Workshop 14b: Minecraft and the Sense HAT

Written by CREATIV3.1Pi

This workshop focuses on some more advanced Python and using it with Minecraft and the Sense HAT.

Difficulty: Intermediate workshop

Contents

0 Introduction

Python is a computer programming language designed for general-purpose use and can be found in use almost everywhere. Python was designed to be easy to read and write, making it the perfect language to start off with!

Python has been around since 1991 and grew to be popular in 2003, where it was ranked in the top ten most popular programming languages (and it's still there!). It was

the programming language of the year in 2007 and 2010 and even comes pre-installed on your Pi!

This is an intermediate workshop, it will help to have some previous experience with Python.

How to use these booklets

The aim of these booklets is to help you attempt these workshops at home, and to explain concepts in more detail than at the workshop. You don't need to refer to use these booklets during the workshop, but you can if you'd like to.

Code listings & asides

When you need to make changes to your code, they'll be presented in boxes like the following:

```
1 #This is a comment
2 while True:
3         print("Hello, World!")
```

You might not need to copy everything, so check the line numbers to make sure you're not copying something twice. Occasionally, something will be explained in greater detail in asides, like the one below. You can read these as you wish.

Aside

Red lines beginning with number signs (#) are called **comments**. They are ignored by the program, so we can use them to label pieces of code.

What you'll need

For this workshop, you'll just need a working Raspberry Pi running a Linux operating system like Raspbian with Python installed. Luckily for us, Raspbian (and many other Linux distributions) include Python right out of the box.

Everything else

These booklets were created using LaTeX, an advanced typesetting system used for several sorts of books, academic reports and letters.

 Latex source documents are available at http://github.com/McrRaspJam/booklet-workshops

1 Python in Minecraft

1.1 What is Minecraft?

Minecraft is a popular sandbox open-world building game, a free version of which is pre-installed with Raspbian on your Pi. Minecraft Pi Edition provides a powerful API for development, allowing people to build within the game using Python and making it a great way to get to grips with Python.

1.2 Getting Started

You can find Minecraft under `Games > Minecraft PI` from the Raspberry Pi logo in the top left of your screen. Opening it should present you with a screen allowing you to start a new game or join a game, clicking "Start Game" will allow you to create a new world, which we will be using throughout the workshop.

We will also need Python's IDLE tool, which allows us to write and run Python code. You can find this under `Programming > IDLE`. There may be more than one version available and whilst we will be using Python 3 throughout this workshop, the code may still run under Python 2.

Now that we have all of our software open, we can start programming! The first thing we need to do is import Minecraft's API (the code that they provide so that we can control Minecraft) into Python so we can use it.

```
from mcpi import minecraft
```

After we've imported their code, we can use it to connect to Minecraft using the `Minecraft.create()` function.

```
from mcpi import minecraft
mc = minecraft.Minecraft.create()
```

We need to remember our connection so that we can use it later, so we stored it in the variable `mc`. This variable can be called anything, we just need to keep track of our connection to the game.

We're now connected to the game and can start playing with the game itself.

2 Manipulating the World

2.1 Chat

Minecraft Pi Edition provides a simple function for posting messages into the game's chat called `postToChat`. This function allows us to pass in some text and have the text appear in-game.

```
from mcpi import minecraft
mc = minecraft.Minecraft.create()
mc.postToChat("Hello, World!")
```

Extra: Why not try changing what is said in chat or even try changing how many times it is said?

2.2 Positions

Before we start placing blocks in the world, we need to know where we're placing them! Positions in Minecraft are represented by three numbers, `x`, `y` and `z`. The `x` and `z` values are your horizontal position relative to the centre of the world and the `y` value is how high up you are from the bottom of the world.

You can get the player's current position using `player.getPos()`, which returns an object containing the player's current position. This can be useful for moving the player around using code or modifying blocks relative to the player.

```
from mcpi import minecraft
mc = minecraft.Minecraft.create()
pos = mc.player.getPos()
print(pos)
print(pos.x)
print(pos.y)
print(pos.z)
```

2.3 Blocks

The Minecraft world consists of a set of blocks, each with their own ID which represents the type of block.

ID	Type
0	Air
1	Stone
2	Grass
3	Dirt
4	Cobblestone
5	Wooden Planks
	...
247	Nether Reactor Core

There are lots of types of blocks in Minecraft and it would be a pain to have to look up their IDs every time you want to place a block. This is where the Minecraft API comes in again, as it provides us a list of block names and their respective IDs.

```
from mcpi import block
print(block.TNT.id)
```

Note that we imported `block` from `mcpi` as opposed to `minecraft`. If you are looking to use both Minecraft and the blocks (as you would do if you were writing a normal program), you can import them both by separating their names by a comma.

```
from mcpi import minecraft, block
```

We can place and remove blocks using `setBlock` and specifying the position and type of block to place. An easy way to try out placing blocks is by placing a block over the player's head.

```
from mcpi import minecraft, block
mc = minecraft.Minecraft.create()
x, y, z = mc.player.getPos()
mc.setBlock(x, y + 2, z, block.GRASS.id)
```

Extra: Why not try changing the block from grass to a different block?

Extra: How would you place a block beneath the player instead of above?

Extra: How would you keep placing blocks wherever the player moves? *Hint: Loops allow us to run the same lines of code more than once.*

Hitting Blocks

Block hit events are only emitted by the game when the player *right clicks* a block with a sword, so if your code doesn't seem to be working make sure you're using a sword!

2.4 Events

Events are actions that the player has performed, such as hitting a block or jumping in the air. We can watch what events are happening from Python and act upon them in our code. To do this we need a loop that keeps checking for events and some code to handle the event that is happening.

In this section we'll be writing some code to turn everything the player touches into gold!

Firstly we need the basic framework for handling events, which consists of a loop that runs forever and a function that gets a list of all of the events that we have not processed yet.

```
from mcpi import minecraft

mc = minecraft.Minecraft.create()
```

```
while True:
    for hit in mc.events.pollBlockHits()
        print(hit.pos)
```

This code prints out the position of the block any time the player hits a block, which we can use to turn that block into gold! We can use the function `setBlock` that we learned about earlier to change the block the player hit into a gold block.

```
from mcpi import minecraft, block

mc = minecraft.Minecraft.create()

while True:
    for hit in mc.events.pollBlockHits():
        mc.setBlock(hit.pos.x, hit.pos.y, hit.pos.z, block.
            GOLD_BLOCK.id)
```

Our final code simply takes the position of the block that was hit and turns it into a gold block.

Extra: Why not try changing the block from gold to a different block?

Extra: How would you make it so that the block that was clicked is removed instead of replaced? *Hint: Every empty block in the world is actually an air block.*

3 Integrating the Sense HAT

3.1 Joystick

The Sense HAT has a small joystick on it that allows you to move in 4 directions but also "press" the joystick as a button. We will be using the joystick when integrating the HAT with Minecraft, allowing us to control certain elements of the game using the stick.

By the end of this section we will have created a program that allows us to move the player around using the joystick.

Handling Input

To take input from the joystick, we need a loop that runs forever and some code that waits until the joystick was moved. We also need to be able to tell what type of event happened, such as "pressed" or "released", as well as the direction that the stick was pushed in.

We use the `wait_for_event` function from the Sense HAT's API to wait until the joystick is moved. This function will stop any code from after it running until the stick is moved, which is exactly what we need. When the joystick *is* moved, this function returns information about the movement and allows the rest of the code after it to run.

```
from sense_hat import SenseHat

sense = SenseHat()

while True:
    event = sense.stick.wait_for_event()
    print(event)
```

This code will print out information about the action performed when the joystick is moved, including when the button is pressed.

Moving the Player

Now we need to take the input from the joystick and apply it to the player. We can do this by adding 1 to the position of the player on a specific axis (either `x`, `y` or `z`), depending on how the joystick was moved.

We'll need the input code from earlier again, as well as some of the Minecraft code for getting the player's position.

```
from sense_hat import SenseHat
from mcpi import minecraft

sense = SenseHat()
mc = minecraft.Minecraft.create()

while True:
```

```
    event = sense.stick.wait_for_event()
    print(mc.player.getPos())
```

This code will print the player's position every time the joystick is moved — we're almost there already!

Up next is changing the player's position depending on the direction pressed on the joystick. The easiest way to do this is compare the event's `direction` value against known directions using an `if` statement.

```
from sense_hat import SenseHat
from mcpi import minecraft

sense = SenseHat()
mc = minecraft.Minecraft.create()

while True:
    event = sense.stick.wait_for_event()
    pos = mc.player.getPos()

    if event.direction == "up":
        pos.z -= 1
    if event.direction == "down":
        pos.z += 1
    if event.direction == "left":
        pos.x -= 1
    if event.direction == "right":
        pos.x += 1

    print(pos)
```

This code will print the new position for the player every time the joystick is moved, but without actually moving the player. The `pos.x += 1` lines mean "take the x position of the player and add 1", which is a shorter version of `pos.x = pos.x + 1`. This code will make the up, right, down and left directions of the joystick move the player's position north, east, south and west.

All we need to do now is apply the player's new position using `setPos`.

```
from sense_hat import SenseHat
from mcpi import minecraft

sense = SenseHat()
mc = minecraft.Minecraft.create()

while True:
    event = sense.stick.wait_for_event()
    pos = mc.player.getPos()

    if event.direction == "up":
        pos.z -= 1
    if event.direction == "down":
        pos.z += 1
    if event.direction == "left":
        pos.x -= 1
```

```
    if event.direction == "right":
        pos.x += 1

    mc.player.setPos(pos)
```

And that's it! We now have a working piece of code that allows you to move the player around with the joystick.

You may notice that there are some issues with this, mainly the fact that you can go through walls! Sadly this is not a trivial thing to fix, as we'd need to check if the player is about to move into a block or not before moving them there. Another issue is that the movement is not in the direction the player is looking, which looks rather odd! There isn't much we can do about this, as Minecraft Pi edition doesn't give us a way to work out which way the player is looking, meaning we can't move the player depending on where they are looking.

Extra: How would you make the player move faster when you move the joystick?

Extra: How would you make the player move up when you press the joystick in? *Hint: The direction value will be "push" when the button is pushed in.*

3.2 Display

The Sense HAT's display allows us to display graphics and text, making it useful for outputting information from the computer. In this section we will be outputting the ID of the block that the player hit whenever the player hits a block.

Showing Text

The Sense HAT makes it easy to show text, providing the easy-to-use `show_message` function, which scrolls text across the HAT's display.

```
from sense_hat import SenseHat
sense = SenseHat()
sense.show_message("Hello, World!")
```

Converting Positions to Text

By default, positions in Minecraft are in the form `Vec3(x, y, z)`. This is no good for our screen, as it has quite a bit of text that is *not* the position. We can convert the position to a shorter piece of text by joining the positions into a string using the `+` operator.

```
from mcpi import minecraft

mc = minecraft.Minecraft.create()

while True:
    for hit in mc.events.pollBlockHits()
```

```
        text = hit.pos.x + ", " + hit.pos.y + ", " + hit.pos.z
        print(text)
```

In this example, we converted the `x`, `y` and `z` values into strings and joined them together with commas, giving us the form `x, y, z`.

We have already seen how to listen for events in section 2, so we will skip the explanation behind that code and go straight to outputting the position of blocks that have been hit.

```
from sense_hat import SenseHat
from mcpi import minecraft

sense = SenseHat()
mc = minecraft.Minecraft.create()

while True:
    for hit in mc.events.pollBlockHits():
        text = hit.pos.x + ", " + hit.pos.y + ", " + hit.pos.z
        sense.show_message(text)
```

And that's it! Right clicking on blocks with a sword should show the position of the block on the Sense HAT's screen.

Extra: Why not try changing the colour of the text shown on the HAT's screen?

Extra: How about changing the colour of the text depending on the type of block hit? *Hint: Use `getBlock` alongside `hit.pos` to get the type of block that was hit.*