LINUX VOICE
TUTORIAL

RASPBERRY PI: MAKE GAMES WITH A PIBRELLA AND SCRATCH

LES POUNDER

Use a physical device to create interactive Python and Scratch programs for fun and profit, but mostly for fun.

WHY DO THIS?

- Discover how simple it is to make hardware obey your commands.
- Use the graphical Scratch programming language in a practical application.
- Reprise Benny Hill's role in the Italian Job.

INSTALLING THE PIBRELLA BOARD

The Pibrella board is designed to fit over all of the Raspberry Pi GPIO pins. The board should simply push on with little resistance and the black rubber pad should rest on the capacitor next to the micro USB power socket.

Understanding and predicting how a program works is part of the new Computing curriculum which is being introduced in September of this year. It is also a key part in understanding how a computer thinks and how it uses logic. Children across the UK will need to understand two types of programming languages; one must be a visual language, the other a textual language.

For this issues tutorial we will explore two projects using Pimoroni and Cyntech's latest board, the Pibrella, which we reviewed in Issue 3. The projects for this issue are used to highlight the basic aspect of control, and our first project – a simulation of traffic lights – is an ideal starting point for a commonly seen aspect of our lives. Our second project is a dice simulator, where we can control the main part of the program but we introduce a random element to spice things up.

Python

To use the Pibrella board with our Raspberry Pi, we first need to install an extra module that will enable Python and Scratch to talk to the board. To do this we are going to use a Python packaging tool called **pip**.

First, open a terminal. In the terminal we need to ensure that our list of packages is up to date, so type the following, remembering to press Enter afterwards.

```
sudo apt-get update
```

You will now see lots of on-screen activity, which means that your list of software packages is being updated. When this is complete, type the following to install **pip**, the Python package management tool. If you're asked to confirm any changes or actions, please read the instructions carefully and only answer Yes if you are happy.

```
sudo apt-get install python-pip
```

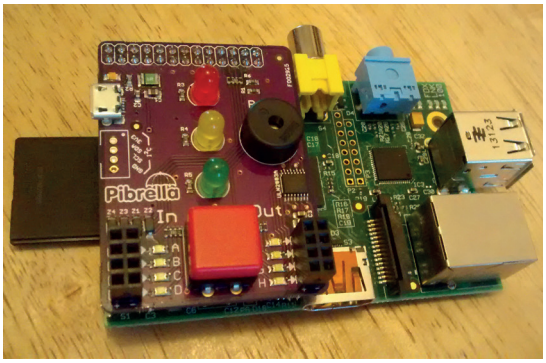
Now it's time to use **pip** to install the **pibrella** package for Python, so type the following:

```
sudo pip install pibrella
```

After a few minutes the **pibrella** module for Python should be installed on your Raspberry Pi.

We'll be using Python 2 for our Python code in this tutorial, and we need to use an editor called Idle to write our code. The Pibrella board attaches to the GPIO (General Purpose Input/Output), and in order for us to use it with Idle we need to launch the Idle application using the sudo command, like so:

```
sudo idle
```



The Pibrella board has a big red button, three LEDS and a buzzer. It's perfect for basic hardware interface messing.

After a few moments you will see the Idle Python editor on your screen.

Scratch

The standard version of Scratch does not have the capability to interact with external components, but this special version maintained by Simon Walters (found on Twitter as **@cymplecy**) enables you to use many different add-on boards and the GPIO directly.

To install ScratchGPIO5plus on your Raspberry Pi, type in the following in a terminal:

```
wget http://goo.gl/Pthh62 -O isgh5.sh
```

This will download a shell script that contains the instructions to install Scratch on your Raspberry Pi. Now that we have the shell script, we need to use it, so in the same terminal type in the following.

```
sudo bash isgh5.sh
```

You will be prompted for your password, once you have typed this in press Enter and you will see lots of commands and actions appear on the screen. Let it complete these tasks, perhaps pop off for a cup of tea and then return when it is done.

When everything is installed, you will see two new icons on your desktop: ScratchGPIO5 and ScratchGPIO5plus. What we are interested in is ScratchGPIO5plus, as this is the version of Scratch that will enable us to use Pibrella. Double-click on the icon to launch ScratchGPIO5plus.

Simulate traffic lights

In the real world, traffic lights are used to control the flow of traffic through a town or city. On your Pibrella you will see three Light Emitting Diodes (LED) that we can use to simulate our own traffic lights using

Scratch and Python. Traffic lights control traffic via the red, amber and green lights which are programmed in these two sequences.

- Green to Amber and then to Red.
- Red and Amber together, and then to Green.

You will see that the sequence is different in reverse, and this enables drivers who are colour blind to know where they are in the sequence. So how can we create this in our code? Well let's first see how it can be achieved in Scratch.

Scratch

In Scratch we can see three columns. These are:

- The palette of blocks, where all of the blocks that make up our program can be found; they are colour-coded to enable children to quickly identify where a block is from.
- A script-building area, where we can drag our blocks of code to build our program.
- Finally there is a stage area that shows the output of our program.

In our code we need to first tell Scratch that we are using the Pibrella board. You'll find this in the Variables palette; it's called **Set AddOn to 0**. Change this to **Set AddOn to PiBrella** and leave it in the palette for now. Now we need to create a piece of code that tells Scratch that when the green flag is clicked, the add-on Pibrella board is to be used, and that all of the inputs and outputs should be turned off. The **When Green Flag Clicked** block is located in the Control palette. We next need to drag the **Set AddOn to PiBrella** from the Variables palette and place it under the Green Flag block. Lastly for this section we need to create a Broadcast block called **AllOff** that tells Pibrella that all of its inputs and outputs should be turned off.

Now that we've told Scratch that we're using the Pibrella board, we need to write the code that controls the main part of our program. Our logic is as follows

```
When the green flag is clicked
Wait until the big red button is pressed
Repeat the following 3 times
        Turn on the Green LED
        Wait 10 seconds
        Turn off the Green LED
        Turn on the Amber LED
        Wait 2 seconds
        Turn off the Amber LED
        Turn on the Red LED
        Wait 10 seconds
        Turn on the Amber LED
        Wait 2 seconds
        Turn off the Amber LED
        Turn off the Red LED
```

Most of the blocks necessary to complete this project are in the Control palette, except for our switch sensor block, which is located in the Sensors palette, and the green six sided-block, which is a comparison block from the Operators panel. The green block that you're looking for is the **=** block in the Operators palette.

You will see a large number of broadcast blocks, and we use those blocks to communicate with the components on the Pibrella. For example, to turn on the Red LED we use **broadcast RedOn** and to turn it off we use **broadcast RedOff**. Each of these broadcast blocks will need to be created as they are not already in the Control palette. Have a go at creating your own sequence.

This is the main functionality that will control our Pibrella and recreate a typical UK traffic light using Scratch. You can see that there are other code snippets in the column; these relate to the output visible in the stage area. Our car can drive across the screen when the light is green, but when the light changes to amber the car will slow down and finally when the light is red the car will come to a stop. I added these elements to introduce another element of control, in that our Pibrella board can influence the car on screen. To take this further, see if you can work out how to add another car to the stage and replicate the code that we created, but alter the speed of the car to make it faster or slower.

Python

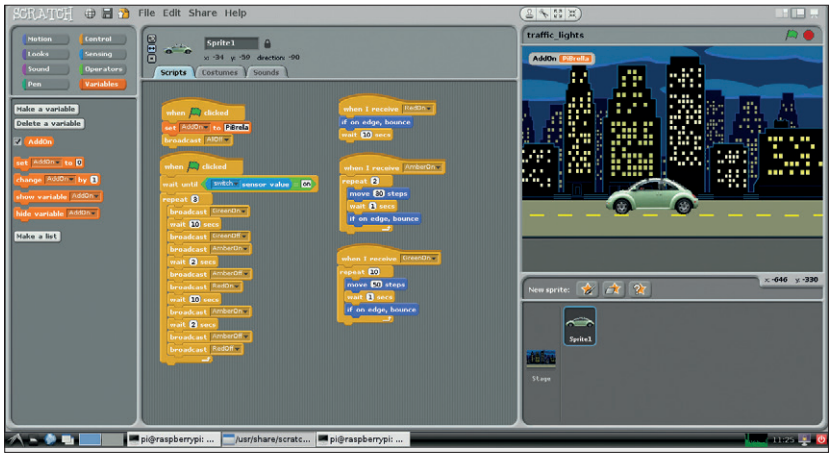
Our Python code for this project is quite similar to the Scratch code that we earlier created. Let's take a look at the code, section by section. We start with importing some external libraries, in this case pibrella and time. Pibrella's library enables us to use the board in our Python code. The time library enables us to control the speed at which our program runs.

```
import pibrella
import time
```

Next we have two variables that control the delays in our code. Delay is used to control the time that the green and the red LED are illuminated for. Sequence is used to control the time that the amber LED is illuminated for.

```
delay = 10
sequence = 2
```

We now move on to a function that groups together all of the steps necessary to turn on and off our LED and control how long this is to be done for. A function is a group of instructions that can be called by using



LV PRO TIP

In these tutorials we used the Pip package manager for Python. Pip takes a lot of the hard work out of managing your Python packages and enables you to quickly update your packages should a new version be made available. You can learn more about Pip and the Python Package Index at **https://pypi.python.org/pypi**.

Perhaps you could also add a horn sound to this car, so that it waits impatiently while the light is red. You can find the Sound palette in Scratch, coloured light purple.

the name of the function (I like to think of a function as similar to a macro). It can automate a lot of steps and makes debugging our code easier as we only have to look at the function and not search through our code for any issues. To create a function called **traffic_lights** you would do as follows.

```
def traffic_lights():
```

Now that we have named our function we have to tell it what to do when it is used, and this is what the code looks like.

```
    #Create the sequence
        #Green on for 10 seconds
        print("GREEN")
        pibrella.light.green.on()
        time.sleep(delay)
        pibrella.light.green.off()
        #Amber for 2 seconds
        print("AMBER")
        pibrella.light.amber.on()
        time.sleep(sequence)
        pibrella.light.amber.off()
        #Red for 10 seconds.
        print("RED")
        pibrella.light.red.on()
        time.sleep(delay)
        # Don't turn off the red light until the end of the amber
        # sequence.
        print("AMBER & RED TOGETHER")
        pibrella.light.amber.on()
        time.sleep(sequence)
        pibrella.light.amber.off()
        pibrella.light.red.off()
```

You'll notice that the code under the name of our function is indented – this is correct. Python uses indentation to show what code belongs to the function, it even applies to loops.

Expansion activity

In our project we used two variables to control the delay in our light sequence. We can change these quite easily, but why not ask the user to change them interactively? Python 2 has a great function called **raw_input** that can add interactive content to your project (in Python 3 it is renamed to **input**). To use **raw_input** in place of our current values we can try this.

```
delay = raw_input("How long should the green and red light be on? ")
sequence = raw_input("How long should the amber light be on? ")
```

So now once we start our program we will be asked two questions relating to how long the lights should be on. Answer each of the questions and press enter after entering your answer.

2 GENERATING RANDOM OUTPUT: DICE GAME

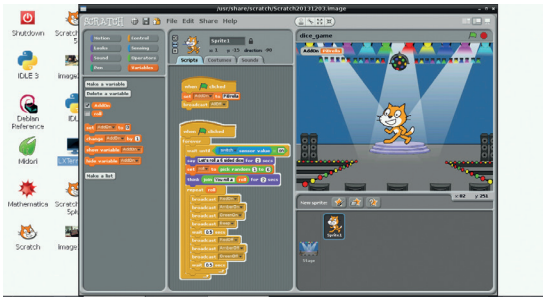
In the first project our expected result and our actual result matched, because we designed our program that way. In the second project, while we will still have an element of control to our program, our actual result will differ as we will be using a random number to simulate throwing a die.

```
When the big red button is pressed
    Say that the program will pick a random number between 1 and 6
    On the screen tell the player what the number is
    Flash all of the LED on your Pibrella the same number of times as the random number
    For each flash of the LED the on board buzzer will buzz
```

Looking at this logic sequence we can control everything apart from the number that is chosen at random – let's build this in Scratch:

Scratch

We start with telling Scratch that we're using the Pibrella board and that all of the inputs and outputs should be off. This is exactly the same start as Project 1. Next we see a forever loop, that is watching for us to press the big red button; as soon as we press the button our on-screen cat will say "Let's roll a 6 sided dice". Next our code will set a variable called **roll**, but where does this variable live in Scratch? Well if you click on the Variables palette you will see a button called "Make a variable". clicking on this will trigger a pop-up asking you to name your variable, so name it



To add bells and whistles, can you can think of a way to trigger the cat to dance if you roll a six?

roll. There will also be two options asking if this variable applies to all sprites or just this one. For this project either option is applicable, so the choice is yours. Now that we have a variable called **roll**, let's recreate the dice throw logic.

To assign a value to a variable in Scratch we need to set a value to the variable. So in our project we **Set roll to...** but what do we set it to? Well, we use a block from the Operators palette that will pick a random number. We drag that block and drop it into the Set block, so now we have a method to randomly pick a number and store it in our **roll** variable. Our code now moves to show the randomly chosen value via a block in the Looks palette. This block, called "Think" creates a thought bubble type effect on the screen, just like those found in cartoons and comics. We then drop

another block from the Operators palette called "Join" into the Think block. This now gives us a method to join our "You roll a" string with the value stored in our roll variable.

The last section of our code is a loop that will iterate the same number of times as the value of our dice **roll** variable. Each time the loop goes round it turns all of the Pibrella LEDs on and beeps the buzzer, then waits for half a second before turning the LED off, then lastly waiting for half a second before repeating the loop.

So that's Project 2, our dice game in Scratch. Try it out and see if it works. For extra points, see if you can work out how to change our dice to a higher or lower number of sides?

Python

The structure of our Python code for Project 2 is quite similar to Project 1.

We first import the libraries that will add extra functionality to the code. There will be two libraries that we have used before, namely **pibrella** and **time**. But you can see a new library called **random**. The **random** library enables us to add an element of surprise to our dice game, and I'll show you how that works later in the code.

```
import random
import time
import pibrella
```

Now that the imports are completed, we next create a function that will handle the main process of the game. This function called **dice()** is made up of a few sections, I'll break it down and explain what happens in each section.

In this section we create a variable, called **guess**, which will store the output of **random.randint(1,6)**. What does that mean? Well, we earlier imported a library called **random**, and from that library we want to use a function called **randint**, or random integer in plain English. In Python the syntax is

```
guess = random.randint(1,6)
```

Next we want to tell the player what the program will achieve, and to do this I print a string of text for the player to read.

```
print("I'm going to think of a number between 1 and 6 and I will flash the lights on your Pibrella to indicate the number I chose")
```

Now that the program has picked a random number and stored it as a variable we want to tell the player what the number was. The reason for this is two fold: one, the game would be no fun if the player were not told the result; and two, we can use this to debug the code later on in the project.

In the **print** function you can see **"The number is "+str(guess)**, what this is demonstrating is something called concatenation, or in other words joining together. The problem that we have is that the text is a string in Python, but the contents of the variable **guess** is an integer. In order to concatenate two things they must be of the same type, and that's where **str** comes in to play. What **str** does is temporarily convert the contents of our variable from an integer into a string, which we can then join to the string **"The number is"**

```
print("The number is "+str(guess))
```

Let's move on to the next section of the function. Here we use a **for** loop that instructs the Pibrella to flash all of the LEDs and play the buzzer to match the guessed number. This provides a great audio/visual output to our game and enables us to explore different methods of output. A loop works by checking to see if a condition is true and if that is correct it looks to see what code should be run.

We start the **for** loop by saying "for every time the loop goes round" and then we tell Python how many times the loop should go round by saying "start at 0 and finish before you get to the number guessed". In Python this is how it looks.

```
for i in range(0,(guess)):
```

We start at 0 rather than 1 because a range will end before it gets to the chosen number. So if we started at 1, the number of flashes would be 1 less than the guess due to the range ending. So we would then have to add 1 to our guess variable, so it's easier to start at 0 and work from there.

So now that we have our **for** loop we need to write the code to flash our LED and beep our buzzer. We start with a half-second delay to help our loop run smoothly.

```
    time.sleep(0.5)
```

Our next part of the sequence controls turning all of the LED on and playing a note on the buzzer, waiting for half a second and then turning the LED and buzzer off. Pibrella has a special function that turns on all of the LEDs without having to individually call them by their names. Pibrella also has a special function to control the note played on the buzzer, but this function is different to those that we have encountered before. This function can take an argument – in other words we can tell the function what note to play, which in this case is **(1)**. Here is all the code to control our LED and buzzer.

```
    pibrella.light.on()
    pibrella.buzzer.note(1)
    time.sleep(0.5)
    pibrella.light.off()
    pibrella.buzzer.off()
```

Project files

All of the files used in these projects are available via my GitHub repository. GitHub is a marvellous way of storing and collaborating on code projects. You can find my GitHub repo at **https://github.com/lesp/LinuxVoice_Pibrella**.

If you're not a Github user, don't worry you can still obtain a zip file that contains all of the project files. The zip file can be found at **https://github.com/lesp/LinuxVoice_Pibrella/archive/master.zip**.

Les Pounder is a maker and hacker specialising in the Raspberry Pi and Arduino. Les travels the UK training teachers in the new computing curriculum and Raspberry Pi.