Pygame Zero for Makers

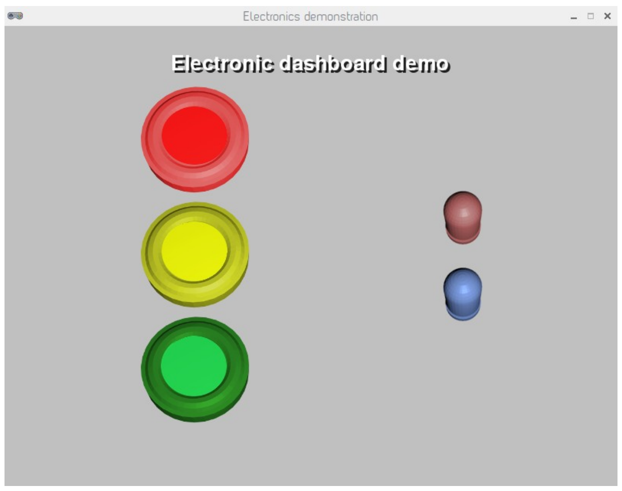
Using Pygame Zero to create novel interfaces for controlling electronic projects

Introduction

Computer interfaces don't need to be boring! In this tutorial I will show you how Pygame Zero can be used to create graphical interfaces that can control the real world through physical computing.

This is a hands-on guide using Pygame Zero and GPIO Zero to provide a graphical interface to electronic circuits. You will create a simple graphical interface to turn some LEDs on and off and to read the state of some switches. This is specifically designed for the Raspberry Pi, although the techniques for creating the graphical interface can also be used with other computers.

After using the simple exercises in this worksheet you should be able to go on to create more elaborate graphical interfaces for controlling many different projects.



In the worksheet I have referred to devices as being virtual when they are displayed on the screen and real or physical when they exist as real devices on the breadboard.

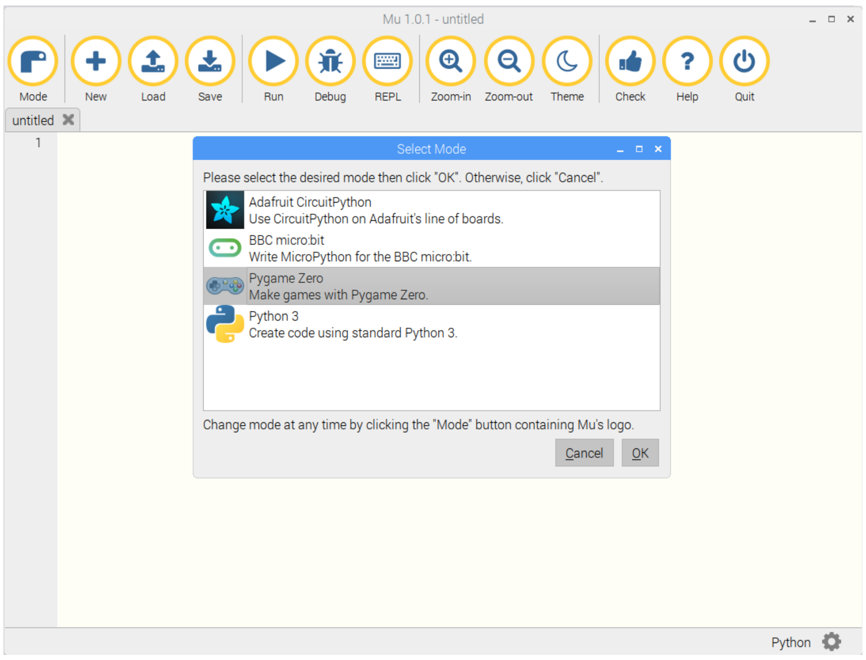
Please note that the code created in this project is not necessarily the most efficient. It's been designed for simplicity to make it easier to understand, but there are opportunities to make the code more efficient.

Getting Started with Mu

When developing programs using the Pygame Zero then it is easiest using the mu editor, as the editor includes the ability to run Pygame Zero programs. If mu is not installed already then it can be installed using

```
sudo apt install mu
```

Mu will then be available from the programming menu. When you first start mu then it will ask what mode you wish to run in. You should select Pygame Zero. If you need to change the mode in future you can change t the mode using the “Mode” button menu (top left).



Please select the desired mode then click "OK". Otherwise, click "Cancel".

Adafruit CircuitPython
Use CircuitPython on Adafruit's line of boards.

BBC micro:bit
Write MicroPython for the BBC micro:bit.

Pygame Zero
Make games with Pygame Zero.

Python 3
Create code using standard Python 3.

Change mode at any time by clicking the "Mode" button containing Mu's logo.

Cancel OK

To get started and test that everything is setup correctly create a first program file with the following two lines:

```
WIDTH = 800
HEIGHT = 600
```

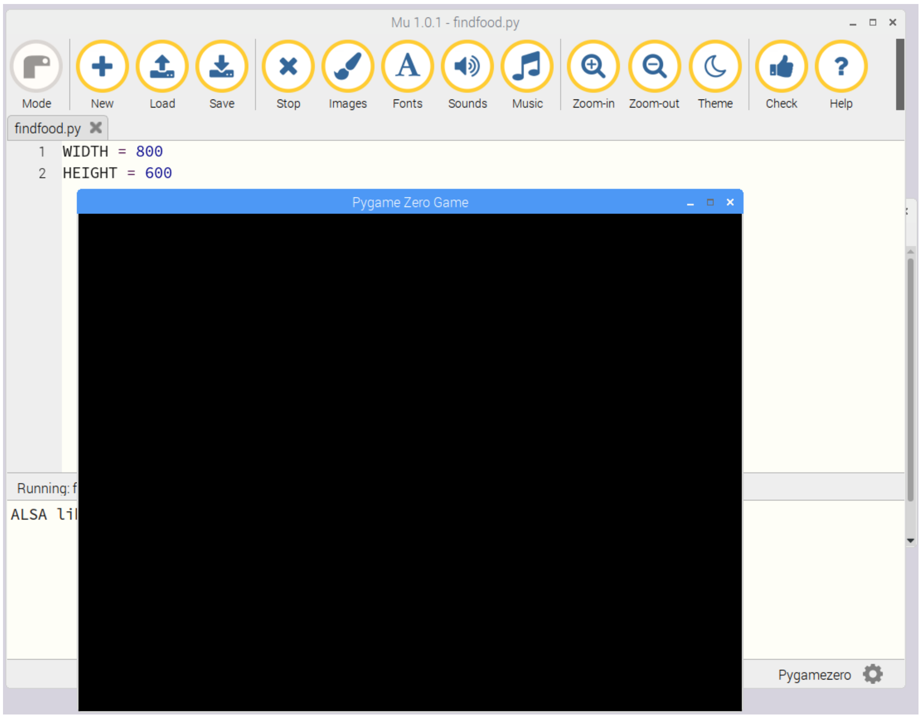
Save it as electronics.py

To test from Mu click on the Play button. You will need to save the file which by default is stored in the mu_code directory.

If you want to run the program from outside of Mu then change to the appropriate directory and enter

```
pgzrun electronics.py
```

When run you the program should create a new windows with a black background.



```
WIDTH = 800
HEIGHT = 600
```

```
ALSA lil
```

You can close the program by clicking on the x in the top right, or by pressing Stop from the Mu menubar.

You've now created your first Pygame Zero file.

Changing the background colour

I think the black background is a bit dark for my console, so I've changed it to a light grey colour. This is done by adding the following code after the WIDTH and HEIGHT values.

```
def draw():
    screen.fill((192, 192, 192))
```

Click on the Play button and you should now see the same screen from before, but it will now have a grey background.

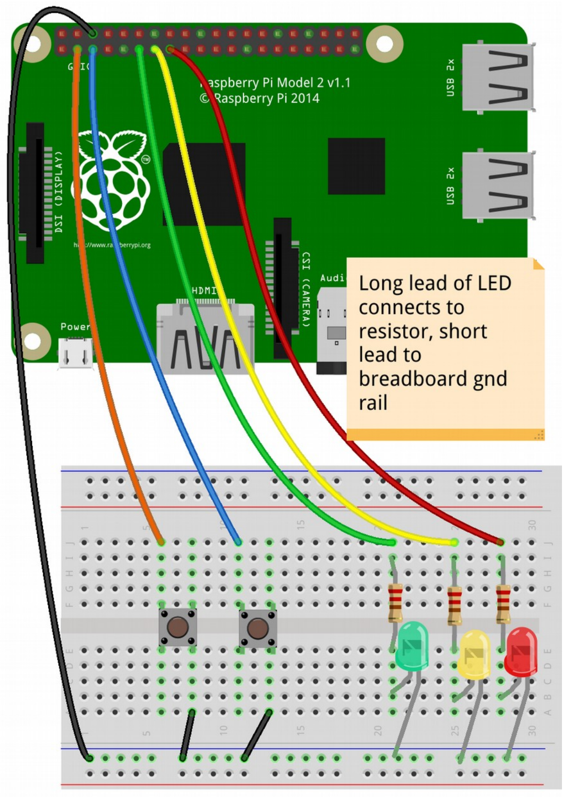
This works by filling the screen with the rgb colour values. These are numbers representing the red, green and blue components. If you want a different colour then you can change these values, the easiest way is to use an online colour picker and look for the RGB values listed against your preferred colour.

The line
```
def draw():
```
is defining the draw function. This is a Python function that Pygame Zero calls approximately 60 times per second. It should be used to tell Pygame Zero what should be displayed on the screen.

In this case the code calls screen.fill() which fills the screen with the colour listed. The double brackets is intentional, without those the code will not run. This is because the RGB value needs to be stored in a tuple (which is a collection of numbers) which is defined by the inner set of brackets and the outer brackets are required for the fill method.

Electronics

The diagram below shows the electronic circuit for this project.


Raspberry Pi Model 2 v1.1
© Raspberry Pi 2014
GPIO
USB 2x
USB 2x
DSI (DISPLAY)
CSI (CAMERA)
HDMI
Power
Audio
Long lead of LED connects to resistor, short lead to breadboard gnd rail

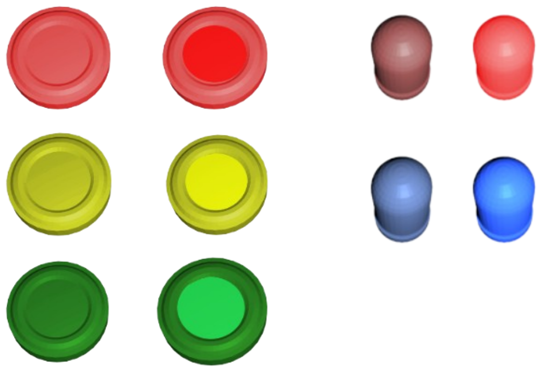
Port Number	GPIO Number	Input / Output
6	gnd	Ground
3	2	Switch 1
5	3	Switch 2
11	17	Green LED
13	27	Yellow LED
15	22	Red LED

The pins for the LEDs have been chosen so that you could use an LED traffic light module instead of the 3 separate LEDs.

Creating a Sprite

A sprite is an image used in a computer game that is often created from a bitmap image. In this case the sprites will be used as buttons to control our circuits and some virtual LEDs that can be switched on and off using the physical buttons.

The images you will need for this were created as simple 3D objects in blender. There are 3 button sprites which are used to turn on and off LEDs, and two LED sprites which are controlled by the physical buttons. I have created two images of each, one with the button / LED turned off and one with it turned on. The images are shown below.



They are available to download from http://www.penguintutor.com/downloads/pgzero-makers.tgz

Extract it using:

tar -xvzf pgzero-makers.tgz

You should then copy the images into an images folder where you are writing the code.
```
cp pgzero-makers/code/images/* /home/pi/mu_code/images

[In the workshop this will normally have been done for you]
```

To add the first image update the code as shown:

```
WIDTH = 800
HEIGHT = 600

red_button = Actor("red-button-1", (150,200))

def draw():
    screen.fill((192, 192, 192))
    red_button.draw()
```

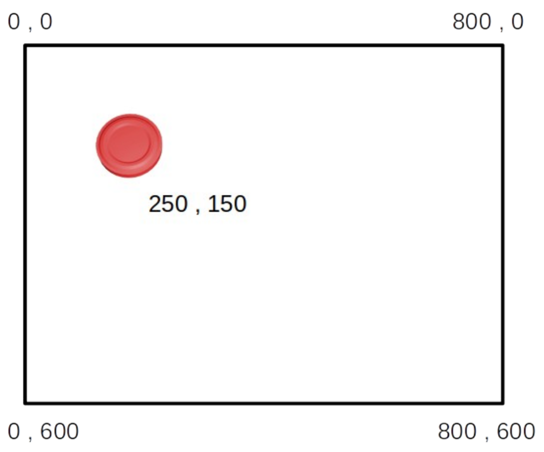
The code that is shown in grey is existing code and the new lines are in black / coloured text.

In Pygame Zero a sprite is known as an actor. You can think of it as an actor on a stage (which is similar terminology that Scratch uses), although in Pygame Zero terminology it is an Actor on the screen. The actor takes an image as it’s argument.

The image should be in the images sub-directory and when entered in the Actor method should not include any path information or an extension (although if using a recent version of pygame zero you can include both path and extension if you like).

The other argument used when creating the Actor, is a tuple that defines the position of the actor on the screen. Setting this to a specific x and y position will position the actor on the screen. The first value is the x co-ordinate increasing from left to right and the second is the y co-ordinate from top to bottom.

The image below shows the co-ordinate system with the red button positioned at 250, 150.



Detecting the button is pressed

Now we have the button we need to know when it's pressed. Initially you can make the button change image as though it has an internal light. This is done using the alternate image.

red-button-1.png represents the button off (not lit up)
red-button-2.png represents the button on (lit up)

To change the image just update the image property eg.

display_led_red.image = "red-led-on"

Then to change it back

display_led_red.image = "red-led-off"

But first you need to detect when it's pressed and in this case we will handle the button press differently to how most computer interfaces work and more like how real button switches work.

In the “real” world if you press a physical button switch then the moment you push the button you make the contact and it will cause something to happen. Whereas when you are on a computer and click on most buttons using a mouse then it normally waits until you release the button before carrying out an action (there are exceptions such as the application menu in both Windows and on the Raspberry Pi, which activates on the first click).

In this example we are creating a computer representation for a physical button so will treat any click as a press. It also happens that the easiest approach, as additional code is required to detect the release.

The way to detect whether something is clicked is to first detect whether the mouse button is pressed and see if the mouse is over the button at the time. This is done using collidepoint which determines if a particular Actor covers a particular position.

This is best explained using some code.

```
WIDTH = 800
HEIGHT = 600

red_button = Actor("red-button-1", (150,200))
red_button.status = False

def draw():
    screen.fill((192, 192, 192))
    red_button.draw()

def on_mouse_down(pos, button):
    if button == mouse.LEFT:
        if red_button.collidepoint(pos):
            if red_button.status == False:
                red_button.status = True
                red_button.image = "red-button-2"
            else:
                red_button.status = False
                red_button.image = "red-button-1"
```

The first thing that's been added is the following line:

```
red_button.status = False
```

This creates a new attribute for the switch which keeps track of whether the switch is in the on state (True) or the off state (False). This isn't actually needed as you could just check what image is being displayed, but I think it makes the code easier to follow later on.

Next is the new function on_mouse_down. This is a feature of pygame zero that if one of the mouse buttons is pressed and you have created this function in your code then it will call that function. It provides the position that the mouse was in (pos) and which button is pressed.

If the button that was pressed is the left button then it looks to see if the position of the mouse cursor collides (overlaps) with the red button Actor.

If it is that particular button that has been pressed then it looks at the status of the button. If it's False (not pressed) then it changes it to True and replaces the image to show the button lit up. Otherwise it sets the status to False and changes the image to the one showing the button in the off state.

Changing the status of the LED

Having set the on-screen button to show whether it is on and off, you can also set the status of the pin controlling the physical LED connected to the breadboard. This is fairly simple thanks to the GPIO Zero library. this worksheet focusses on the graphical interface, but you can find out more about the GPIO Zero library in a previous worksheet www.penguintutor.com/space-asteroids or from the GPIO Zero documentation https://gpiozero.readthedocs.io .

```
from gpiozero import Button, LED

WIDTH = 800
HEIGHT = 600

PIN_LED_RED = 22

led_red = LED(PIN_LED_RED)
red_button = Actor("red-button-1", (150,200))
red_button.status = False

def draw():
    screen.fill((192, 192, 192))
    red_button.draw()

def on_mouse_down(pos, button):
    if button == mouse.LEFT:
        if red_button.collidepoint(pos):
            if red_button.status == False:
                red_button.status = True
                red_button.image = "red-button-2"
                led_red.on()
            else:
                red_button.status = False
                red_button.image = "red-button-1"
                led_red.off()
```

There are only a few additional lines of code needed to control the LEDs. The first line is an import statement to include the GPIO Zero library. Next a constant[1] is created with the value 22 which is the GPIO port that the LED is connected to. This is defined as an LED which is then turned on and off each time that the button changes state.

1 The entry for PIN_LED_RED is actually a variable, by putting it in capital letters it is a convention that this should be treated like a constant and the value should not be changed.

Adding 2 more virtual buttons and LEDs

Adding the additional virtual buttons for yellow and green, and their corresponding LEDs is as simple as duplicating the code for each of the LEDs.

During the first part of the program add the following lines

```
PIN_LED_YELLOW = 27
PIN_LED_GREEN = 17

led_yellow = LED(PIN_LED_YELLOW)
led_green = LED(PIN_LED_GREEN)

yellow_button = Actor("yellow-button-1", (250,300))
green_button = Actor("green-button-1", (250,450))
```

Then add the following to the bottom of the on_mouse_down function.

```
    if yellow_button.collidepoint(pos):
        if yellow_button.status == False:
            yellow_button.status = True
            yellow_button.image = "yellow-button-2"
            led_yellow.on()
        else:
            yellow_button.status = False
            yellow_button.image = "yellow-button-1"
            led_yellow.off()
    if green_button.collidepoint(pos):
        if green_button.status == False:
            green_button.status = True
            green_button.image = "green-button-2"
            led_green.on()
        else:
            green_button.status = False
            green_button.image = "green-button-1"
            led_green.off()
```

Controlling virtual LEDs using real switches

So far you have created virtual buttons to control real LEDs. Next you can do the opposite and have real switches control virtual LEDs.

You can add the virtual LEDs to the display by adding the following code to the first block:

```
display_led_red = Actor("red-led-off", (600,250))
display_led_blue = Actor("blue-led-off", (600,350))
```

and then displaying them on the screen inside the draw function.

```
    display_led_red.draw()
    display_led_blue.draw()
```

This is the same as how the virtual buttons were created previously.

To detect when the switch buttons are pressed then there are a few different options. For this example I chose to poll the switches looking to see if they are pressed or not. This uses the update() function which is a feature of Pygame Zero. If you create an update function in your code the it is run approximately every 60 seconds and can be used to handle any changes.

First add the GPIO Zero setup to the top of the code similar to how the physical LEDs were created earlier.

```
PIN_BUTTON_1 = 2
PIN_BUTTON_2 = 3

button_1 = Button(PIN_BUTTON_1)
button_2 = Button(PIN_BUTTON_2)
```

Then create the update function with the following:

```
def update():
    if button_1.is_pressed:
        display_led_red.image = "red-led-on"
    else:
        display_led_red.image = "red-led-off"
    if button_2.is_pressed:
        display_led_blue.image = "blue-led-on"
    else:
        display_led_blue.image = "blue-led-off"
```

This checks each of the two buttons in turn by looking at their “is_pressed” attribute. If the button is pressed then it sets the image to the on button, and if it's not pressed then it sets the image to the off button.

I won't repeat the code at this stage, but if you need to see how the code should look flip to the back where the final code is included.

The draw() vs the update() function

It is worth restating the purpose of the two different functions which are common to most Pygame Zero programs.

```
# Draw function – updates the screen, include draw of any actors.
def draw():

# Update function – use to handle updates to the players and check the status of the keyboard.
def update():
```

As long as you define these in your code they will be run approximately 60 times per second.

Note that the timing of the movement is only approximate. Although the update function should run 60 times every second the actual timing will depend upon a number of factors including whether the system has sufficient resources available to meet that requirement. For more details see: https://pygame-zero.readthedocs.io/en/stable/hooks.html#update

Display text

Finally you can add some text to the screen to give it a title.

The method to display the text is screen.draw.text. This needs to be within the draw function rather than update.

```
screen_title = “Pygame Zero for Makers Demo“

screen.draw.text(screen_title, fontsize=40, center=(400,50), shadow=(1,1), 
color=(255,255,255), scolor="#202020")
```

The arguments are mostly self explanatory. The color value is the colour of the text and scolor the shadow. The color is set using RGB values representing the amount of Red,Green,Blue respectively and the scolor using a string based on the hexadecimal value (similar to how colours are defined in HTML / CSS). I have deliberately used both to show that you can use either, but you can use whichever colour format you prefer.

You can also change the title at the top of the window by adding a TITLE entry along with the WIDTH and HEIGHT entries.

Complete code

The code for the complete game is shown below:

```
from gpiozero import Button, LED

screen_title = "Pygame Zero for Makers Demo"

# Set Pygame Zero screen size and title
WIDTH = 800
HEIGHT = 600
TITLE = screen_title

# PINs on the GPIO for the LEDs and Buttons
# pins are valid for a Raspberry pi revision 2 or later
PIN_LED_RED = 22
PIN_LED_YELLOW = 27
PIN_LED_GREEN = 17
PIN_BUTTON_1 = 2
PIN_BUTTON_2 = 3

# GPIO Zero objects for the LEDs and buttons
led_red = LED(PIN_LED_RED)
led_yellow = LED(PIN_LED_YELLOW)
led_green = LED(PIN_LED_GREEN)
button_1 = Button(PIN_BUTTON_1)
button_2 = Button(PIN_BUTTON_2)

# Pygame Zero objects for the buttons (which control the physical LEDs)
# and LEDs which are (controlled by the physical buttons)
red_button = Actor("red-button-1", (250,150))
yellow_button = Actor("yellow-button-1", (250,300))
green_button = Actor("green-button-1", (250,450))
display_led_red = Actor("red-led-off", (600,250))
display_led_blue = Actor("blue-led-off", (600,350))

# Set status of the Pygame Zero buttons so you can toggle between on and off
red_button.status = False
yellow_button.status = False
green_button.status = False

def draw():
    screen.fill((192, 192, 192))
    screen.draw.text(screen_title, fontsize=40, center=(400,50), shadow=(1,1),
color=(255,255,255), scolor="#202020")
    red_button.draw()
    yellow_button.draw()
    green_button.draw()
    display_led_red.draw()
    display_led_blue.draw()


def update():
    if button_1.is_pressed:
        display_led_red.image = "red-led-on"
    else:
        display_led_red.image = "red-led-off"
    if button_2.is_pressed:
        display_led_blue.image = "blue-led-on"
    else:
        display_led_blue.image = "blue-led-off"


def on_mouse_down(pos, button):
    if button == mouse.LEFT:
        if red_button.collidepoint(pos):
            if red_button.status == False:
                red_button.status = True
                red_button.image = "red-button-2"
                led_red.on()
            else:
                red_button.status = False
                red_button.image = "red-button-1"
                led_red.off()
        if yellow_button.collidepoint(pos):
            if yellow_button.status == False:
                yellow_button.status = True
                yellow_button.image = "yellow-button-2"
                led_yellow.on()
            else:
                yellow_button.status = False
                yellow_button.image = "yellow-button-1"
                led_yellow.off()
        if green_button.collidepoint(pos):
            if green_button.status == False:
                green_button.status = True
                green_button.image = "green-button-2"
                led_green.on()
            else:
                green_button.status = False
                green_button.image = "green-button-1"
                led_green.off()
```

Other interfaces

Once you understand the basics you can go on to design other interfaces to integrate with your maker projects. That could include touch interfaces, speedometer displays, bar graphs or anything else you can think of.

Updates and Other Projects

For a related project see the Space Asteroids game from the 2018 Raspberry Pi Birthday. The game is also based around Pygame Zero and GPIO Zero http://www.penguintutor.com/projects/space-asteroids

For future updates on this and other projects follow PenguinTutor on Twitter, Facebook or YouTube.