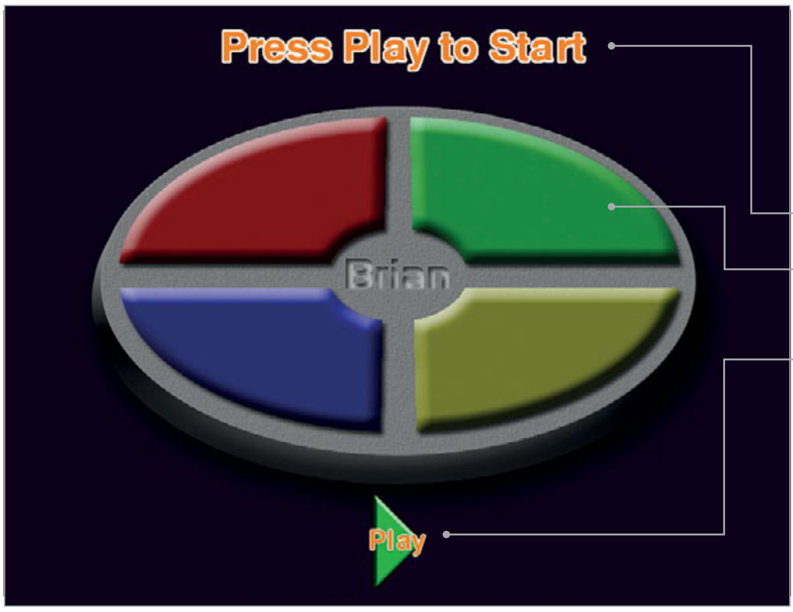
MARK VANSTONE

Educational software author from the nineties, author of the ArcVenture series, disappeared into the corporate software wasteland. Rescued by the Raspberry Pi!
technovisualeducation.co.uk
twitter.com/mindexplorers



Instructions are displayed here

The game has four coloured buttons that light up when they are pressed

Let's get this Brian battle underway

PYGAME ZERO PART 02

VIDEO GAME LOGIC WITH SIMPLE BRIAN

Recreate a classic electronic game using Pygame Zero

You'll Need

- Raspbian Jessie or newer
- An image manipulation program such as GIMP, or images from **magpi.cc/keaiKi**
- The latest version of Pygame Zero (1.2)
- A good memory

Long, long ago, before the Raspberry Pi existed, there was a game. It was a round plastic box with four coloured buttons on top and you had to copy what it did. To reconstruct this classic game using Pygame Zero, we'll first need a name. To avoid any possible trademark misunderstandings and because we are using the Python language, let's call it 'Brian'. The way the game works is that Brian will show you a colour sequence by lighting up the buttons and then you have to copy the sequence by pressing the coloured buttons in the same sequence. Each round, an extra button is added to the sequence and you get a point for each round you complete correctly. The game continues until you get the sequence wrong.

>STEP 01

Run, run as fast as you can

In the first part of this series (**magpi.cc/71**), we ran our Pygame Zero code by typing the **pgzrun** command into a Terminal window, as previously there was no way of running a Pygame Zero program directly from IDLE. With the 1.2 update of Pygame Zero, there is now a way to run your programs directly from IDLE without using a Terminal window. You will need to make sure that you are running version 1.2 or later of Pygame Zero. If the code in **Figure 1** does not run straight from IDLE then you may need to upgrade Raspbian on your Pi.

figure1.py

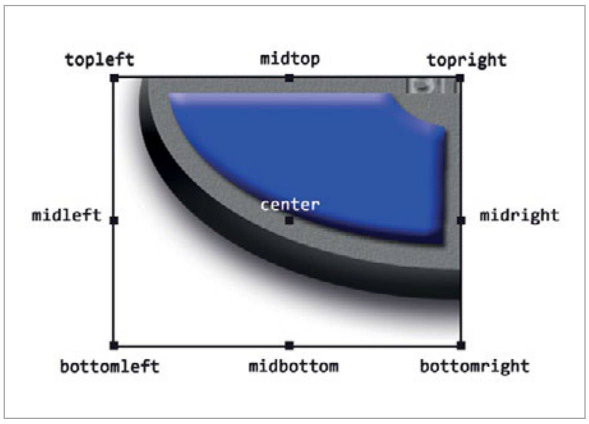
```
import pgzrun

# Your program code will go here

pgzrun.go()
```

Figure 1 With Pygame Zero V1.2 you can now run your program from IDLE with these extra lines top and bottom

>STEP 02

The stage is set

If we are using Pygame Zero, the point is to get things happening quickly, so let's get some stuff on the screen. We are going to need some images that make up the buttons of the Brian game. You can make your own or you can get ours from GitHub at **magpi.cc/keaiKi**. The images will need to be in an **images** directory next to your program file. We have called our starting images **redunlit**, **greenunlit**, **blueunlit**, and **yellowunlit** because all the buttons will be unlit at the start of the game. We have also got a play button so that the player can click on it to start the game.



Above There are nine positions that an Actor's co-ordinates can be aligned to when the Actor is created

>STEP 03

Getting the actors on stage

As we saw in part one, we can create Actors by simply supplying an image name and a position on the screen for it to go. There are several ways of supplying the position information. This time we are going to use position handles to define where the character appears. We will use the same co-ordinates for each quadrant of the whole graphic, but we will change the handle names we use. For these Actors we can use **`bottomright`**, **`bottomleft`**, **`topright`**, and **`topleft`**, as well as the co-ordinates (400,270), which is the centre point of our whole graphic. Have a look at **Figure 2** and see how this is written.

>STEP 04

Look at the state of that

We now need to add some logic to determine if each button is on or off and show it lit or unlit accordingly. We can do this by adding a variable to each of our Actors. We want it to be either on or off, so we can set this variable as a Boolean value, i.e. True or False. If we call this variable 'state', we can add it to the Actor by writing (for the first button): **`myButtons[0].state = False`**. We then do the same for each of the button Actors with their list numbers 1, 2, and 3 because we defined them as a list.

>STEP 05

Light goes on, light goes off

We have defined a state for each of our buttons, but now we have to write some code to react to that state. First let's make a couple of lists which hold the names of the images we will use for the two states. The first list will be the images we use for the buttons being lit, which would be: **`buttonsLit = ['redlit', 'greenlit', 'bluelit', 'yellowlit']`**. We then need a list of the unlit buttons: **`buttonsUnlit = ['redunlit', 'greenunlit', 'blueunlit', 'yellowunlit']`**.

Language

>PYTHON

DOWNLOAD:
magpi.cc/hPLqsi

figure2.py

```python
import pgzrun

myButtons = []
myButtons.append(Actor('redunlit', bottomright=(400,270)))
myButtons.append(Actor('greenunlit', bottomleft=(400,270)))
myButtons.append(Actor('blueunlit',topright=(400,270)))
myButtons.append(Actor('yellowunlit',topleft=(400,270)))
playButton = Actor('play', pos=(400,540))

def draw(): # Pygame Zero draw function
    screen.fill((30, 10, 30))
    for b in myButtons: b.draw()
    playButton.draw()

pgzrun.go()
```

Figure 2 Set up the screen by creating Actors and draw them in the `draw()` function

figure3.py

Figure 3 Adding a state to the buttons means that we can change the image in our `update()` function and draw them as lit

```python
import pgzrun

myButtons = []
myButtons.append(Actor('redunlit', bottomright=(400,270)))
myButtons[0].state = False
myButtons.append(Actor('greenunlit', bottomleft=(400,270)))
myButtons[1].state = False
myButtons.append(Actor('blueunlit',topright=(400,270)))
myButtons[2].state = False
myButtons.append(Actor('yellowunlit',topleft=(400,270)))
myButtons[3].state = False
buttonsLit = ['redlit', 'greenlit', 'bluelit', 'yellowlit']
buttonsUnlit = ['redunlit', 'greenunlit', 'blueunlit', 'yellowunlit']
playButton = Actor('play', pos=(400,540))

def draw(): # Pygame Zero draw function
    screen.fill((30, 10, 30))
    for b in myButtons: b.draw()
    playButton.draw()

def update(): # Pygame Zero update function
    bcount = 0
    for b in myButtons:
        if b.state == True: b.image = buttonsLit[bcount]
        else: b.image = buttonsUnlit[bcount]
        bcount += 1

pgzrun.go()
```

GLOBALS

You can read global variables inside a function, but if you change the value of the variable, you must declare it as global in the function.

Then we can use these lists in an **`update()`** function to set the image of each button to match its state. See **Figure 3** to see how we can do this.

>STEP 06

Switching images

We can see from **Figure 3** that each time our **`update()`** function runs, we will loop through our button list. If the button's state is **`True`**, we set the image of the button to the image in the **`buttonsLit`** list. If not (i.e. the state variable is **`False`**) then we set the image of the button to the image in the **`buttonsUnlit`** list. At this stage you can test to see if your logic is working by setting the state variables at the top of the code to **`True`** and see if when you run the program, the buttons display lit.

>STEP 07

What happens if I press this button?

The idea of a button is that you can press it and something happens, so the next thing we need to write is a way to allow the user to press the buttons and make them light up. We can do this with the Pygame Zero functions **`on_mouse_down()`** and **`on_mouse_up()`**. What we need to test in these functions is if the mouse has been clicked down; if it has, we should set our button state to **`True`**. We also need to test if the mouse button has been released, in which case, all the buttons should be set to **`False`**. We can test the value we are passed (**`pos`**) into these functions with the method **`collidepoint()`**, which is part of the Actor object.

figure4.py

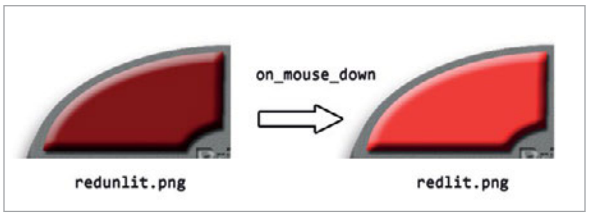
Figure 4 Check for mouse clicks on mouse down, and set all buttons back to unlit on mouse up

```
def on_mouse_down(pos):
    global myButtons
    for b in myButtons:
        if b.collidepoint(pos): b.state = True

def on_mouse_up(pos):
    global myButtons
    for b in myButtons: b.state = False
```

figure5.py

Figure 5 addButton() generates a random number between 0 and 3, then adds this number to the end of the list called buttonList. Then we call the playAnimation() function

```
def playAnimation():
    global playPosition, playingAnimation
    playPosition = 0
    playingAnimation = True

def addButton():
    global buttonList
    buttonList.append(randint(0, 3))
    playAnimation()
```



Above When the mouse is clicked on a button, we switch the unlit image for the lit image

>STEP 08

Ups and downs

We can write a test in **`on_mouse_down()`** for each button, to see if it has been pressed, and then change the state of the button if it has been pressed. We can then write code to set all the button states to **`False`** in the **`on_mouse_up()`** function, and our **`update()`** and **`draw()`** functions will now reflect what we need to see on the screen from those actions. Look at **Figure 4** and you will see how we can change the state of the buttons as a response to mouse events. When you have added this to your program, test it to make sure that the buttons light up correctly when clicked.

>STEP 09

Write a list

Now that we have our buttons working, we will need to make a way to use them in two different ways. The first will be for the game to display a sequence for the player to follow, and the second is to receive input from the player when they repeat the sequence. For the first task we will need to build a list to represent the sequence and then play that sequence to the player. Let's define our list at the top of the code with **`buttonList = []`** and then make a function **`def addButton():`** which will create an additional entry into the sequence each round.

>STEP 10

That's a bit random

We can generate our sequence by generating random integers using the **`random`** module. We can use this module by importing it at the top of our code with: **`from random import randint`**. We will only need the **`randint()`** function, so we can import that function specifically. To add a new number to the sequence in the **`addButton()`** function, we can write **`buttonList.append(randint(0, 3))`**, which will add a number between 0 and 3 to our list. Once we have added our new number, we will want to show the player the sequence, so add a line in the **`addButton()`** function: **`playAnimation()`**.

>STEP 11

Playing the animation

We have set up our function to create the sequence. Now we need a system to play that sequence so that the player can see it. We will do this with a counter variable called **`playPosition`**. We define this at the start of our code: **`playPosition = 0`**. We will also need a variable to show that our animation is playing: **`playingAnimation = False`**, also written at the start of the code. We can then define our **`playAnimation()`** function that we used in the previous step. Look at **Figure 5** to see how the **`addButton()`** and **`playAnimation()`** functions are written.

>STEP 12

Are we playing?

So, once we have set our animation going, we will need to react to that in our **`update()`** function. We know that all we need to do is change the state of the buttons and the **`draw()`** function will handle the visuals for us. In our **`update()`** function, we have to say: "If the animation is playing then increment our animation counter, check that we haven't reached the end of the animation and if not then light the button (change its state to True) which is indicated by our sequence list." This is a bit of a mouthful, so in **Figure 6** we can see how this translates into code.

>STEP 13

Getting a bit loopy

We can see from **Figure 6** that we are incrementing the play position each time **`update()`** is called. What we want to do is keep each button in the sequence lit for several refreshes, so we divide the **`playPosition`** by a predefined number (**`LOOPDELAY`**) to get our list position that we want to display. We round the result downwards with the **`math.floor()`** function. To use this function, you will have to import the **`math`** module at the top of your code. So if **`LOOPDELAY`** is 80 then we will move from one list position (**`listpos`**) to the next every 80 times **`update()`** is called.

>STEP 14

A dramatic pause

Still in **Figure 6**, we check to see if we have reached the end of the **`buttonList`** with **`listpos`**. If we have then we stop the animation. Otherwise, if we are still running the animation, we work out which button should be lit from our **`buttonList`**. We could just say "light that button and set the rest to unlit", but before we do that we have a line that basically says: "If we are in the second half of our button lighting loop, set all the buttons to unlit." This means that we will get a pause in between each button being lit when no buttons are lit. We can then just loop through our buttons and set their state according to the value of **`litButton`**.

>STEP 15

Testing the animation

Now, ignoring the fact that we have a play button ready and waiting to do something, we can test our animation by calling the **`addButton()`** function.

figure6.py

```
def update(): # Pygame Zero update function
    global myButtons, playingAnimation, playPosition
    if playingAnimation:
        playPosition += 1
        listpos = math.floor(playPosition/LOOPDELAY)
        if listpos == len(buttonList):
            playingAnimation = False
            clearButtons()
        else:
            litButton = buttonList[listpos]
            if playPosition%LOOPDELAY > LOOPDELAY/2: litButton = -1
            bcount = 0
            for b in myButtons:
                if litButton == bcount: b.state = True
                else: b.state = False
                bcount += 1
```

Figure 6 The `update()` function with a check to see if the animation is playing. If it is, we need to work out which buttons to light and move the animation on

This function adds a random button number to the list and sets the animation in motion. To test it, we can call it a few times at the bottom of our code, just above the **`pgzrun.go()`**. If we call the **`addButton()`** function three times then three numbers will be added to the **`buttonList`** list and the animation will start. If this all works, we are ready to add the code to capture the player's response.

> "We can collect the player's clicks on the buttons just by adding another list"

>STEP 16

I need input

We can collect the player's clicks on the buttons just by adding another list, **`playerInput`**, to the definitions at the top of the code and adding a few lines into our **`on_mouse_down()`** function. Add a counter variable **`bcount = 0`** at the top of the function and then add one to **`bcount`** at the end of the loop. Then, after **`if b.collidepoint(pos):`** we add **`playerInput.append(bcount)`**. We can then test the player input to see if it matches the **`buttonList`** list we are looking for. We will write this as a separate function called **`checkPlayerInput()`** and call it at the end of our **`on_mouse_down()`** function. As we now have the basis of our game, refer to the full listing to see how the rest of the code comes together as we go through the final steps.

MODULO OR %

The % symbol is used to get the remainder after a division calculation. It's useful for creating smaller repeats within a larger loop.

CHANGING THE DELAY

We have used the constant LOOPDELAY for timing our loops, if the game is running too slow, decrease this value at the top of the code.

>STEP 17

Game over man

The **checkPlayerInput()** function will check the buttons that the player has clicked against the list held in **buttonList** which we have been building up with the **addButton()** function. So we need to loop through the **playerInput** list with a counter variable – let's call it **ui**, and write **if playerInput[ui] != buttonList[ui]: gameOver()**. If we get to the end of the list and both **playerInput** and **buttonList** are the same length then we know that the player has completed the sequence and we can signal that the score needs to be incremented. The score variable can be defined at the top of the code as **score = 0**. In our **on_mouse_up()** function, we can then respond to the score signal by incrementing the score and setting the next round in motion.

>STEP 18

Just press play

We still haven't done anything with that play button Actor that we set up at the beginning. Let's put some code behind that to get the game started. Make sure you have removed any testing calls at the bottom of your code to **addButton()** (Step 15). We will need a variable to check if the game is started, so put **gameStarted = False**

> "Let's put some code behind that to get the game started"

at the top of the code with the other variables and then in our **on_mouse_up()** function we can add a test: **if playButton.collidepoint(pos) and gameStarted == False:** and then set the **gameStarted** variable to **True**. We can set a countdown variable when the play button is clicked so that there is a slight pause before the first animation starts.

>STEP 19

Finishing touches

We're nearly there with our game: we have a way to play a random sequence and build that list round by round, and we have a way to capture and check user input. The last things we need are some instructions for the player, which we can do with the Pygame Zero **screen.draw.text()** function. We will want an initial 'Press Play to Start' message, a 'Watch' message for when the animation is playing, a 'Now You' message to prompt the player to respond, and a score message to be displayed when the game is over. Have a look in the **draw()** function in the complete listing to see how these fit in. There are many ways we can enhance this game; for example, the original electronic game had sound too, but that will be covered in another part of this series.

USING TEXT

If you are going to use text, it's a good idea to display some on the first screen as it can take a while to load fonts for the first time. You may get unwanted delays if you load it later.

brian.py

```python
import pgzrun
from random import randint
import math
WIDTH = 800
HEIGHT = 600

myButtons = []
myButtons.append(Actor('redunlit', bottomright=(400,270)))
myButtons[0].state = False
myButtons.append(Actor('greenunlit', bottomleft=(400,270)))
myButtons[1].state = False
myButtons.append(Actor('blueunlit', topright=(400,270)))
myButtons[2].state = False
myButtons.append(Actor('yellowunlit', topleft=(400,270)))
myButtons[3].state = False
buttonsLit = ['redlit', 'greenlit', 'bluelit', 'yellowlit']
buttonsUnlit = ['redunlit', 'greenunlit', 'blueunlit', 'yellowunlit']
playButton = Actor('play', pos=(400,540))
buttonList = []
playPosition = 0
playingAnimation = False
gameCountdown = -1
LOOPDELAY = 80
score = 0
playerInput = []
signalScore = False
gameStarted = False

def draw(): # Pygame Zero draw function
    global playingAnimation, score
    screen.fill((30, 10, 30))
    for b in myButtons: b.draw()
    if gameStarted:
        screen.draw.text("Score : " + str(score), (310, 540), owidth=0.5, ocolor=(255,255,255), color=(255,128,0) , fontsize=60)
    else:
        playButton.draw()
        screen.draw.text("Play", (370, 525), owidth=0.5, ocolor=(255,255,255), color=(255,128,0) , fontsize=40)
        if score > 0:
            screen.draw.text("Final Score : " + str(score), (250, 20), owidth=0.5, ocolor=(255,255,255),
```

Language
>PYTHON

DOWNLOAD:
magpi.cc/hPlqsl

```
color=(255,128,0) , fontsize=60)
        else:
            screen.draw.text("Press Play to Start", (220, 20), owidth=0.5, ocolor=(255,255,255), color=(255,128,0) , fontsize=60)
    if playingAnimation or gameCountdown > 0:
        screen.draw.text("Watch", (330, 20), owidth=0.5, ocolor=(255,255,255), color=(255,128,0) , fontsize=60)
    if not playingAnimation and gameCountdown == 0:
        screen.draw.text("Now You", (310, 20), owidth=0.5, ocolor=(255,255,255), color=(255,128,0) , fontsize=60)

def update(): # Pygame Zero update function
    global myButtons, playingAnimation, playPosition, gameCountdown
    if playingAnimation:
        playPosition += 1
        listpos = math.floor(playPosition/LOOPDELAY)
        if listpos == len(buttonList):
            playingAnimation = False
            clearButtons()
        else:
            litButton = buttonList[listpos]
            if playPosition%LOOPDELAY > LOOPDELAY/2: litButton = -1
            bcount = 0
            for b in myButtons:
                if litButton == bcount: b.state = True
                else: b.state = False
                bcount += 1
    bcount = 0
    for b in myButtons:
        if b.state == True: b.image = buttonsLit[bcount]
        else: b.image = buttonsUnlit[bcount]
        bcount += 1
    if gameCountdown > 0:
        gameCountdown -=1
        if gameCountdown == 0:
            addButton()
            playerInput.clear()

def gameOver():
    global gameStarted, gameCountdown, playerInput, buttonList
    gameStarted = False
    gameCountdown = -1
    playerInput.clear()
    buttonList.clear()
    clearButtons()

def checkPlayerInput():
    global playerInput, gameStarted, score, buttonList, gameCountdown, signalScore
    ui = 0
    while ui < len(playerInput):
        if playerInput[ui] != buttonList[ui]: gameOver()
        ui += 1
    if ui == len(buttonList): signalScore = True

def on_mouse_down(pos):
    global myButtons, playingAnimation, gameCountdown, playerInput
    if not playingAnimation and gameCountdown == 0:
        bcount = 0
        for b in myButtons:
            if b.collidepoint(pos):
                playerInput.append(bcount)
                b.state = True
            bcount += 1
        checkPlayerInput()

def on_mouse_up(pos):
    global myButtons, gameStarted, gameCountdown, signalScore, score
    if not playingAnimation and gameCountdown == 0:
        for b in myButtons: b.state = False
    if playButton.collidepoint(pos) and gameStarted == False:
        gameStarted = True
        score = 0
        gameCountdown = LOOPDELAY
    if signalScore:
        score += 1
        gameCountdown = LOOPDELAY
        clearButtons()
        signalScore = False

def clearButtons():
    global myButtons
    for b in myButtons: b.state = False

def playAnimation():
    global playPosition, playingAnimation
    playPosition = 0
    playingAnimation = True

def addButton():
    global buttonList
    buttonList.append(randint(0, 3))
    playAnimation()

pgzrun.go()
```