GETTING STARTED WITH PYGAME ZERO ON THE RASPBERRY PI

RASPBERRY JAM

MILTON KEYNES

Worksheet
And
Programme Listing

TechnoVisual education

www.technovisualeducation.co.uk

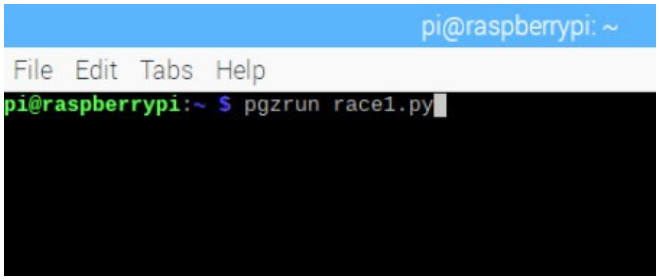
1. RUNNING A PYGAME ZERO PROGRAMME

RASPBERRY JAM MILTON KEYNES

1. First make sure that Pygame Zero is installed.
For instructions go to:
https://pygame-zero.readthedocs.io
/en/stable/installation.html

2. To run a Pygame Zero programme, you will need to first create a blank programme file. First open your favourite Python editor (we will use IDLE for this), create a new file and save it (we will call it 'race1.py').

3. Then from a terminal console window you can run your programme by typing : pgzrun race1.py



You don't even need any lines of code to get Pygame Zero running! You can close the programme window with the top, right close window icon.

1. To start writing our programme we want to define how big the game window is going to be so we can type the following into our programme:

```
# First set the width and height of the window
WIDTH = 700
HEIGHT = 600
```

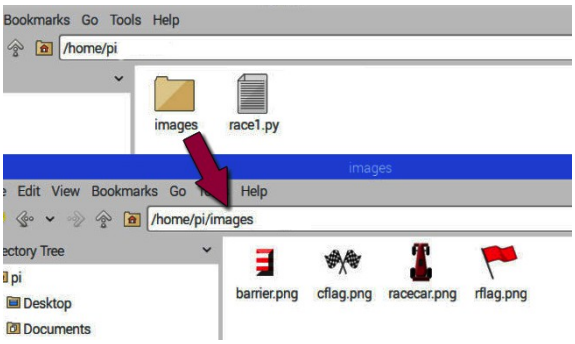
Any lines beginning with # are comments and will be ignored when we run the programme. Save this programme and try running it like before.

Next: Organising Your Graphics

2. ORGANISING YOUR GRAPHICS

RASPBERRY JAM MILTON KEYNES

1. You need to have your images in a place that Pygame Zero can find them. You will need to have a directory called 'images' in the same place that your programme is. You should put all your images inside that directory.



2. Your images should also all be named with lower case letters. The images we will need for this project are:

racecar.png - our racing car image
barrier.png - a barrier for the sides of our track
cflag.png and rflag.png - flag images

If you need to download these files you can get them from: http://www.technovisualeducation.co.uk/pygame-zero-workshop/

3. Once we have our image files in the right place we can make a car sprite. Add the following code to the programme.

```
# Load in the car sprite image as an Actor object
car = Actor("racecar")
car.pos = 250, 500 # Set the car screen position
```

4. We are going to need some variables to hold data about our racing game so add the following:

```
# Some variables to control the track
SPEED = 4
trackCount = 0
trackPosition = 250
trackWidth = 120
trackDirection = False
# The following lists set up the track sprites
trackLeft = []
trackRight = []
# Variable to track the status of the game
gameStatus = 0
```

Next: Drawing To The Screen

3. DRAWING TO THE SCREEN

RASPBERRY JAM MILTON KEYNES

1. Pygame Zero uses a function called draw() to draw items to the screen. Type the following after the variables.

```
# Pygame Zero draw function
def draw():
    global gameStatus
    screen.fill((128, 128, 128))
    if gameStatus == 0:
        car.draw()
```

2. Now try running the programme as before from the terminal window using the pgzrun command. You should see a grey window with a racing car near the bottom.

3. We will need to respond to key presses by the player to make the car move. We can do this with the standard Pygame Zero function called update(). Type in the following after the draw() function:

```
# Pygame Zero update function
def update():
    global gameStatus , trackCount
    if gameStatus == 0:
        if keyboard.left:
            car.x -= 2
        elif keyboard.right:
            car.x += 2
```

If you run the programme again now, you should be able to move the car left and right with the arrow keys on the keyboard.

Next: Making The Race Track

4. MAKING THE RACE TRACK

RASPBERRY JAM MILTON KEYNES

1. Now we have a car that we can move left and right, we need a track for it to race along. Add the following makeTrack() function to your programme.

```
# Function to make a new section of track
def makeTrack():
    global trackCount, trackLeft, trackRight,\
           trackPosition, trackWidth
    trackLeft.append(Actor("barrier", pos =
           (trackPosition-trackWidth,0)))
    trackRight.append(Actor("barrier", pos =
           (trackPosition+trackWidth,0)))
    trackCount += 1
```

2. We also need to make the track move down the screen so we will add an updateTrack() function.

```
# Function to update where the track blocks appear
def updateTrack():
    global trackCount, trackPosition, trackDirection,\
           trackWidth
    if trackLeft[len(trackLeft)-1].y > 32:
        if trackDirection == False:
            trackPosition += 16
        if trackDirection == True:
            trackPosition -= 16

        if randint(0, 4) == 1:
            trackDirection = not trackDirection
        if trackPosition > 700-trackWidth:
            trackDirection = True
        if trackPosition < trackWidth:
            trackDirection = False
        makeTrack()
```

3. In the function above we have used the randint() function so we need to import it by putting the following at the top of our programme.

```
from random import randint
```

Next: Checking For Collisions

5. CHECKING FOR COLLISIONS

RASPBERRY JAM MILTON KEYNES

1. We can see the track being moved if we add a call to the makeTrack() function at the bottom of our code ...

```
makeTrack() # Make first block of track
```

and a call to the updateTrack() function in our update() function.

```
            updateTrack() # Move all the track blocks down
```

2. At the moment we can drive the car through the track barriers so we need to add some code to detect collisions. Below is our updated draw() function including what we typed before.

```
# Pygame Zero draw function
def draw():
    global gameStatus
    screen.fill((128, 128, 128))
    if gameStatus == 0:
        car.draw()
        b = 0
        while b < len(trackLeft):
            if (car.colliderect(trackLeft[b]) or
               car.colliderect(trackRight[b])):
                # Red flag time
                gameStatus = 1
            trackLeft[b].draw()
            trackLeft[b].y += SPEED
            trackRight[b].draw()
            trackRight[b].y += SPEED
            b += 1
    if gameStatus == 1:
        # Red Flag
        screen.blit('rflag', (318, 268))
```

3. So now when the car touches a barrier the gameStatus variable gets changed to 1. Then the draw() function will display a red flag image. Run the programme as before to test this.

Next: Getting To The Finish Line

6. GETTING TO THE FINISH LINE

RASPBERRY JAM MILTON KEYNES

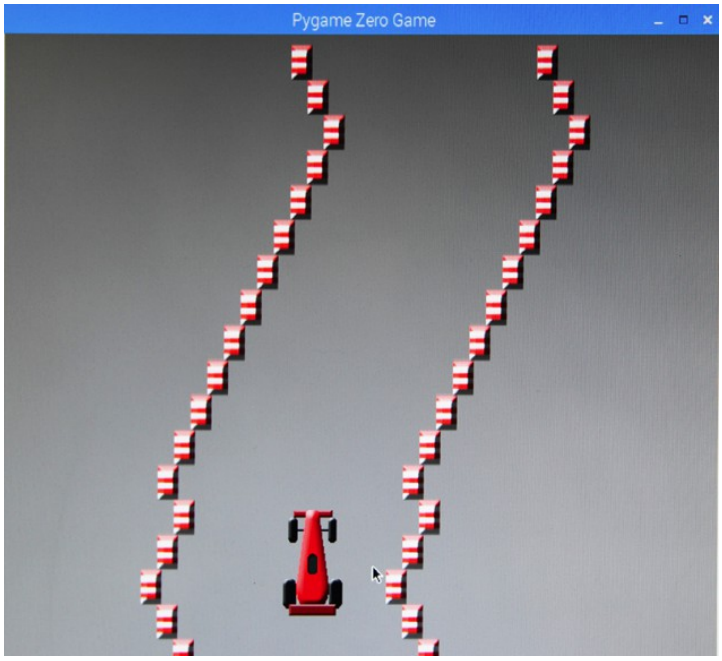
1. If our player gets through 200 track sections without hitting any barriers we can set a race finished gameStatus (chequered flag). Do this by adding the following code in the update() function.

```
    if trackCount > 200:
        # Chequered flag time
        gameStatus = 2
```

2. We will also need a new condition at the bottom of the draw() function.

```
    if gameStatus == 2:
        # Chequered Flag
        screen.blit('cflag', (318, 268))
```

So now we should have a finished racing game. Run it to test that you can get to the end to see the chequered flag but also test to make sure that if the car hits a barrier you get a red flag.



Next: The Whole Programme

THE WHOLE PROGRAMME

RASPBERRY JAM MILTON KEYNES

```
# PyGame Zero Racing Game
from random import randint

# First set the window width and height
WIDTH = 700
HEIGHT = 600

# Load in the car sprite image as an Actor
car = Actor("racecar")
car.pos = 250, 500 #car start position

# Some variables to control the track
SPEED = 4
trackCount = 0
trackPosition = 250
trackWidth = 120
trackDirection = False
# The following set up the track sprites
trackLeft = []
trackRight = []
# Variable to track the status of the game
gameStatus = 0

# Pygame Zero draw function
def draw():
    global gameStatus
    screen.fill((128, 128, 128))
    if gameStatus == 0:
        car.draw()
        b = 0
        while b < len(trackLeft):
            if(car.colliderect(trackLeft[b])
      or car.colliderect(trackRight[b])):
                # Red flag time
                gameStatus = 1
            trackLeft[b].draw()
            trackLeft[b].y += SPEED
            trackRight[b].draw()
            trackRight[b].y += SPEED
            b += 1
    if gameStatus == 1:
        # Red Flag
        screen.blit('rflag', (318, 268))
    if gameStatus == 2:
        # Chequered Flag
        screen.blit('cflag', (318, 268))
```

```
# Pygame Zero update function
def update():
    global gameStatus , trackCount
    if gameStatus == 0:
        if keyboard.left:
            car.x -= 2
        elif keyboard.right:
            car.x += 2
        updateTrack()
    if trackCount > 200:
        # Chequered flag time
        GameStatus = 2

# Our game functions

# Function to make a new section of track
def makeTrack():
    global trackCount, trackLeft,\
      trackRight,trackPosition, trackWidth
    trackLeft.append(Actor("barrier", pos =
        (trackPosition-trackWidth,0)))
    trackRight.append(Actor("barrier", pos =
        (trackPosition+trackWidth,0)))
    trackCount += 1

# Function to update the track blocks
def updateTrack():
    global trackCount, trackPosition,\
      trackDirection, trackWidth
    if trackLeft[len(trackLeft)-1].y > 32:
        if trackDirection == False:
            trackPosition += 16
        if trackDirection == True:
            trackPosition -= 16

        if randint(0, 4) == 1:
            trackDirection = not \
             trackDirection
        if trackPosition > 700-trackWidth:
            trackDirection = True
        if trackPosition < trackWidth:
            trackDirection = False
        makeTrack()

# End of functions

makeTrack() # Make first block of track
```

Note that some lines have a "\" at the end – this is a line continuation mark which means the next line could be put on the same line if the "\" is removed.

A copy of this worksheet and the programme resources are available from: http://www.technovisualeducation.co.uk/pygame-zero-workshop/