Source Code template

Source Code is a regular feature that runs to two pages, with a typical word count of 800-1000 words. The aim is to select a game from history, focus on a significant aspect of it, talk a little bit about its history, and then illustrate how it works with a brief snippet of code.

All copy should be laid out per the following template.

Please submit your content marked up using the @tags listed below.

@authorname

Mac Bowley – image included

@header

Create a verticle scrolling road, straight out of Spy Hunter.

@intro

The arcade game Spy Hunter challenged players to take the role of a secret agent, controlling a futuristic car as it navigates a road full of obstacles and other vehicles. In this week’s Source Code, Mac Bowley shows you how to create a vertical scrolling road in Pygame Zero.

@bodycopy

**Intro**

The 1983 arcade classic *Spy Hunter* was inspired by the popular James Bond film franchise, in fact it was originally supposed to be a James Bond licensed game. The game puts players at the wheel of a fictitious Interceptor vehicle (presumably the work of Q, or the non licensed equivalent), and challenges them to navigate the vertical scrolling road destroying enemy vehicles. The top down view point gives the impression that the player is following the car in a helicopter. The road is generated as the game progresses, this was one of the first games to combine the challenges of driving and shooting into one game.

In this week’s tutorial, I will show you how you can recreate the vertical scrolling road to use in your own driving games. Although there are numerous ways a vertical scrolling road can be generated, in this article I will show you a simple implementation using a queuing system. In true Source Code fashion I will use Python and Pygame in this tutorial, but the mechanics can be achieved in almost any engine/language using the same principles.

The road will be created using the Rect class from Pygame, the rectangles will be 2 pixels high and I will stack them to create the “road”. The first thing to do is to create 2 lists; one to hold the pieces of the road currently being drawn on screen, and another to hold a queue of pieces that will be added as the road scrolls down. I will also set 2 colours, one for the background and another for the road.

@crosshead

A queue of pieces that will be added

~~~python

c\_grass = (0, 153, 76)

c\_road = (204, 136, 0)

road = []

queue = []

block\_size = 2

~~~

At the start of the game the road should cover the entire height of the screen, in my game I have set this to 540 pixels. Each of my rectangles is 2 pixels high (which I have also stored in a variable), meaning a total of 220 rectangles are required for my road. I can use a for loop that counts in twos to populate the road list. As my road scrolls I will be adding blocks to the top of the screen, it is much easier to add items to the end of lists than at the front, so I want my road list to start at the bottom of the screen and move upwards. The last block should be at y = 0.

@crosshead

it is much easier to add items to the end of lists

~~~python

buffer = int(WIDTH/4)

for y in range(HEIGHT-block\_size, -block\_size, -block\_size):

block = Rect((buffer, y), (int(WIDTH/2), block\_size))

road.append(block)

~~~

The buffer is used to set the x co-ordinates of the blocks, I want the road to start in the middle of the screen – so each block is half a screen wide and sits a quarter of the way across the screen. Just this code has set up my road, which we can draw by iterating over the list.

~~~python

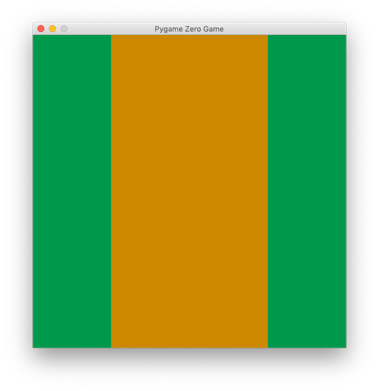
def draw():

screen.fill(c\_grass)

for piece in road:

screen.draw.rect(piece, c\_road)

~~~



That is looking cool, but it is not yet a scrolling road – let’s fix that.

To create the scrolling road effect we want, each of the current pieces of road need to move down the screen and a new piece should be added to the end of the list at position y = 0. Before I can add to my road I need a queue of pieces to add. I can populate the queue list immediately after I create the road, using the same techniques.

Pygame has a built in function that can be used to schedule functions, which will then be called at set intervals – meaning I can scroll my road at a set “frame rate”. The scroll\_road function you can see above will achieve this, first I loop over each piece of the road and move it down by the height of my pieces (2). I then remove the first item in the queue list and append it to the end of the road, making sure it is at y = 0. The pygame clock is then set to call the function at intervals set by a frame\_rate variable, mine is set to 1/60 meaning sixty frames per second.

Essentially, you could now just keep adding pieces to the queue in the same way we have been so far, creating an always straight scrolling road – but where is the fun in that? I also want my road to change direction and produce turns for my player to navigate. Another function is needed to populate the queue, choosing a left or right turn at random. As my road scrolls and it nears the end of the queue the update path function will be called to add more pieces to the queue.

My road can either turn left or right, a random choice made whenever the queue is populated. Whichever way the road turns, it has to start from the same spot as the last piece in my queue. I can grab the last item in a list using -1 as an index and then store the x position, building from here will make sure my road is continuous. The road should stay on screen and I would like a buffer of 50 pixels from the edge of my screen – each time a turn is made I check that the road will not go beyond this point. I would like the turn amount to be random but a turn of 10 pixels, or another small amount, would not be very fun so I am also setting a minimum turn of 200 pixels. If this minimum turn amount would take my car closer than the buffer, I will instead set the turn amount so it takes it up to the buffer but no further. I do this for both directions, and as well as setting a modifier to apply to my turn amount (-1 to turn left and 1 to turn right), which will save me duplicating my code. I also want to randomly choose how many pieces will be involved in my turn, each piece is a step in the scroll, the more pieces the longer my turn will take. This will make sure I have a good mix of sharp and elongated turns in my road, keeping the player engaged. Finally, I add a little more straight to my road after the turn – using another for loop to add pieces in the same x co-ordinate of the last piece of the turn.

@crosshead

You should include one or two crossheads for every page of content. This is a short (2-5 word) interjection in the text. Think of it as a break for the eyes.

@bodyends

This tag marks the end of the main chunk of text.

## **Boxouts**

You should aim to have one boxout in Source Code.

@boxout

You could easily add difficulty to your game by decreasing the frame rate variable – which would make the road scroll faster. You could even gradually increase this over time, making the game feel more frantic the further you get.

Another nice improvement would be to make the turns more curvy, make sure you are comfortable with algebra before you do this!

@endboxout

This tag is used to close the boxout.

## **The code**

@code

import random

# Set the screen dimensions

WIDTH = 540

HEIGHT = 540

# Colour variables

c\_grass = (0, 153, 76)

c\_road = (204, 136, 0)

# Lists to hold pieces

road = [] # To be drawn on screen

queue = [] # To be added when scrolling

# Height of my road pieces

block\_size = 2

# Player actor and a speed to move it at

player = Actor("car.png", (int(WIDTH/2 - 16), 390))

speed = 5

# Initial population of the road and queue

buffer = int(WIDTH/4)

for i in range(HEIGHT-block\_size, -block\_size, -block\_size):

block = Rect((buffer, i), (int(WIDTH/2), block\_size))

road.append(block)

for i in range(0, 200, block\_size):

block = Rect((buffer, 0), (int(WIDTH/2), block\_size))

queue.append(block)

# Function that will scroll the road

def scroll\_road():

global road, queue

for piece in road: # Move all the pieces down by 2

piece.top += block\_size

road.append(queue.pop(0)) # Move piece from queue to road

road.pop(0) # Remove the bottom road piece

road[-1].top = 0

if len(queue) < 5:

update\_path() # If the queue is getting low update the path

# Scroll the road at a set interval - 60 scrolls (frames) per second

frame\_rate = 1/60

clock.schedule\_interval(scroll\_road, frame\_rate)

# Road should not go any closer than 50 pixels to the edge

min\_buffer = 50

# Make sure the road doesn't exceed my buffer

def clamp\_road(x):

if x < min\_buffer:

x = min\_buffer

if x > int(WIDTH/2) - min\_buffer:

x = int(WIDTH/2) - min\_buffer

return x

min\_turn = 200

turn\_gap = 200

def update\_path():

global road, queue

choice = random.randint(0, 1) # Right or left turn

current\_pos\_x = queue[-1].left

if choice == 0:

# Turn left

modifier = -1

if current\_pos\_x - min\_turn > min\_buffer:

turn = random.randint(min\_turn, current\_pos\_x - 5)

else:

turn = current\_pos\_x - min\_buffer

else:

# Turn right

modifier = 1

if int(WIDTH/2) - current\_pos\_x - min\_buffer > min\_turn:

turn = random.randint(min\_turn, int(WIDTH/2) - current\_pos\_x - min\_buffer)

else:

turn = int(WIDTH/2) - current\_pos\_x - min\_buffer

# Choose how long my turn will take

height = random.randint(200, 400)

# Move a percentage of the turn for each block in height

for y in range(block\_size, height, block\_size):

x = turn/height \* y \* modifier

new\_x = clamp\_road(current\_pos\_x + x)

block = Rect((new\_x, 0), (int(WIDTH/2), block\_size))

queue.append(block)

# Find the last x co-ordinate of the turn

current\_pos\_x = queue[-1].left

# Add some straight to the road

for i in range(0, turn\_gap, block\_size):

block = Rect((current\_pos\_x, 0), (int(WIDTH/2), block\_size))

queue.append(block)

def update():

# Player movement

global player

player\_momentum = 0

if keyboard.left:

player\_momentum = -speed

elif keyboard.right:

player\_momentum = speed

else:

player\_momentum = 0

new\_pos = player.x + player\_momentum

collision = False

for i in range(16):

if new\_pos > road[75+i].x and new\_pos + player.width < road[75+i].x + road[75+i].width:

collision = True

if collision == True:

player.x = new\_pos

def draw():

screen.fill(c\_grass)

for piece in road:

screen.draw.rect(piece, c\_road)

player.draw()

## 

## **Images and captions**

You need to include a minimum of two pictures per page.

Any problems or questions about images, just contact your friendly editor: [ryan.lambie@raspberrypi.org](mailto:ryan.lambie@raspberrypi.org)

@captions

All images need captions. Captions should be 15 - 25 words and should state something interesting that’s not already mentioned in the text.

For each image, you should include the following two tags:

@image -- the filename of the picture

@caption -- the text to go along with the image. If you’ve referred to this image in the text, you should start the caption with “figure 1:” (or figure 2, etc).

@endcaptions