Spy Hunter was one of the very first games with both driving and shooting.

Source Code

Make a Spy Hunter–style scrolling road

Mac shows you how to make the beginnings of a top-down driving game

AUTHOR
MAC BOWLEY

The 1983 arcade classic *Spy Hunter* put players at the wheel of a fictitious Interceptor vehicle and challenged them to navigate a vertically scrolling road, destroying enemy vehicles.

Here, I'll show you how you can recreate the game's scrolling road to use in your own driving games. The road will be created using the Rect class from Pygame, with the road built from stacked rectangles that are each two pixels high.

First, I create two lists; one to hold the pieces of road currently being drawn on screen, and another to hold a queue of pieces that will be added as the road scrolls. To create the scrolling road effect, each of the current pieces of road will need to move down the screen, while a new piece is added to the end of the list at position y = 0.

Pygame can schedule functions, which can then be called at set intervals – meaning I can scroll my road at a set frame rate. The `scroll_road` function will achieve this. First, I loop over each road piece, and move it down by two pixels. I then remove the first item in the queue list and append it to the end of the road. 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 60 frames per second.

"The game can also be speeded up by decreasing the frame_rate variable"

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. I use a buffer of 50 pixels to keep the road from moving off the edge of my screen – each time a turn is made, I check that the road doesn't go beyond this point. I want the turn amount to be random, so I'm also setting a minimum turn of 200 pixels. If this amount takes my car closer than the buffer, I'll instead set the turn amount so that it takes it up to the buffer but no further. I do this for both directions, 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, so 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.

To make things more exciting, the game can also be speeded up by decreasing the `frame_rate` variable. You could even gradually increase this over time, making the game feel more frantic the further you get.

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

A rolling road in Python

Here's Mac's code snippet, which creates a winding road worthy of *Spy Hunter* in Python. To get it running on your system, you'll need to install Pygame Zero – you can find full instructions at **wfmag.cc/pgzero**

Download the code from GitHub: **wfmag.cc/wfmag31**

```
import random

WIDTH = 540
HEIGHT = 540

c_grass = (0, 153, 76)
c_road = (204, 136, 0)

road = [] # To be drawn on screen
queue = [] # To be added when scrolling

block_size = 2

player = Actor("car.png", (int(WIDTH/2 - 16), 390), anchor =
("left", "top"))
speed = 5

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)

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 # An index of -1 is last item in a list
    if len(queue) < 5:
        update_path() # If queue is getting low, update path

frame_rate = 1/60 # Scroll the road at a set interval
clock.schedule_interval(scroll_road, frame_rate)

min_buffer = 50

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:
        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:
        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
    height = random.randint(200, 400)
    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)
    current_pos_x = queue[-1].left
    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()
```

Our code snippet provides a solid basis for your own top-down driving game. All you need now are weapons. And a few other cars.