Source Code

The original arcade machine had three steering wheels and three accelerator pedals.

Recreate Super Sprint's top-down racing

Making player and computer-controlled cars race round a track isn't as hard as it sounds

AUTHOR
MARK VANSTONE

Decades before the advent of more realistic racing games like *Sega Rally* or *Gran Turismo*, Atari produced a string of popular arcade racers, beginning with *Gran Trak 10* in 1974 and gradually updated via the *Sprint* series, which appeared regularly through the seventies and eighties. By 1986, Atari's *Super Sprint* allowed three players to compete at once, avoiding obstacles and collecting bonuses as they careened around the track.

The original arcade machine was controlled with steering wheels and accelerator pedals, computer-controlled cars added to the racing challenge. Tracks were of varying complexity, with some featuring flyover sections and shortcuts, while oil slicks and tornadoes provided obstacles to avoid. If a competitor crashed really badly, a new car would be airlifted in by helicopter.

So how can we make our own *Super Sprint*-style racing game with Pygame Zero? To keep this example code short and simple, I've created a simple track with a few bends. In the original game, the movement of the computer-controlled cars would have followed a set of coordinates round the track, but as computers have much more memory now, I have used a bitmap guide for the cars to follow. This method produces a much less predictable movement for the cars as they turn right and left based on the shade of the track on the guide.

"This snippet shows how you can get a top-down racing game working"

With Pygame Zero, we can write quite a short piece of code to deal with both the player car and the automated ones, but to read pixels from a position on a bitmap, we need to borrow a couple of objects directly from Pygame: we import the Pygame image and Color objects and then load our guide bitmaps. One is for the player to restrict movement to the track, and the other is for guiding the computer-controlled cars around the track.

The cars are Pygame Zero Actors, and are drawn after the main track image in the `draw()` function. Then all the good stuff happens in the `update()` function. The player's car is controlled with the up and down arrows for speed, and the left and right arrows to change the direction of movement. We then check to see if any cars have collided with each other. If a crash has happened, we change the direction of the car and make it reverse a bit. We then test the colour of the pixel where the car is trying to move to. If the colour is black or red (the boundaries), the car turns away from the boundary.

The car steering is based on the shade of a pixel's colour read from the guide bitmap. If it's light, the car will turn right, if it's dark,

Top-down racing in Python

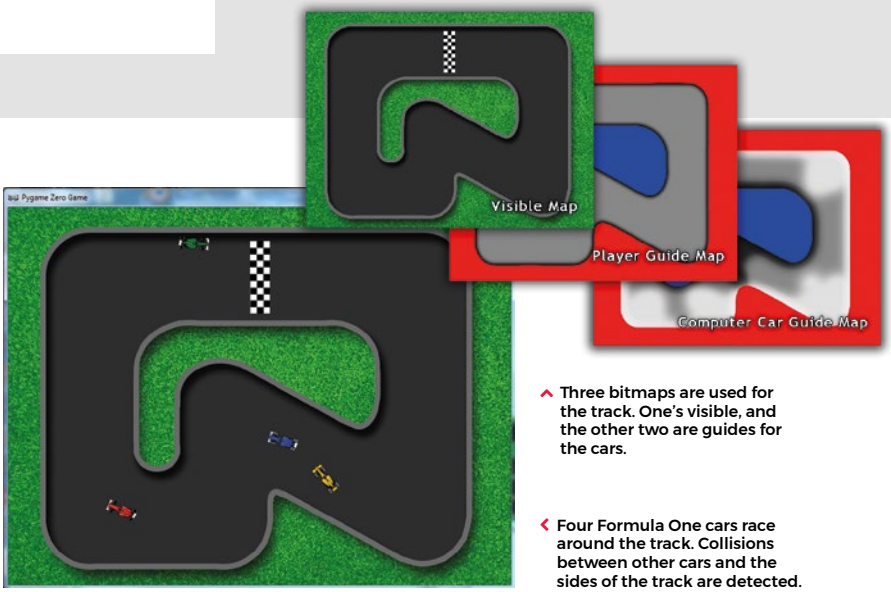
Here's a code snippet that creates a *Super Sprint*-style racer in Python. To get it running on your system, you'll first need to install Pygame Zero – you can find full instructions at **wfmag.cc/pgzero**

Download the code from GitHub: **wfmag.cc/wfmag21**

```
import math
from random import randint
from pygame import image, Color
controlimage1 = image.load('images/guide1.png')
controlimage2 = image.load('images/guide2.png')
cars = []
for c in range(4):
    cars.append(Actor('car'+str(c), center=(400, 70+(30*c))))
    cars[c].speed = 0

def draw():
    screen.blit("track", (0, 0))
    for c in range(4):
        cars[c].draw()

def update():
    if keyboard.up: cars[0].speed += .15
    if keyboard.down: cars[0].speed -= .15
    if(cars[0].speed != 0):
        if keyboard.left: cars[0].angle += 2
        if keyboard.right: cars[0].angle -= 2
    for c in range(4):
        crash = False
        for i in range(4):
            if cars[c].collidepoint(cars[i].center) and c != i:
                crash = True
                cars[c].speed = -(randint(0,1)/10)
        if crash:
            newPos = calcNewXY(cars[c].center, 2, math.
radians(randint(0,360)-cars[c].angle))
        else:
            newPos = calcNewXY(cars[c].center, cars[c].
speed*2, math.radians(180-cars[c].angle))
        if c == 0:
            ccol = controlimage1.get_
at((int(newPos[0]),int(newPos[1])))
        else:
            ccol = controlimage2.get_
at((int(newPos[0]),int(newPos[1])))
        if cars[c].speed != 0:
            if ccol != Color('blue') and ccol != Color('red'):
                cars[c].center = newPos
            else:
                if c > 0:
                    if ccol == Color('blue'):
                        cars[c].angle += 5
                    if ccol == Color('red'):
                        cars[c].angle -= 5
                cars[c].speed = cars[c].speed/1.1
        if c > 0 and cars[c].speed < 1.8+(c/10):
            cars[c].speed += randint(0,1)/10
            if crash:
                cars[c].angle += ((ccol[1]-
136)/136)*(2.8*cars[c].speed)
            else:
                cars[c].angle -= ((ccol[1]-
136)/136)*(2.8*cars[c].speed)
        else:
            cars[c].speed = cars[c].speed/1.1

def calcNewXY(xy,speed,ang):
    newx = xy[0] - (speed*math.cos(ang))
    newy = xy[1] - (speed*math.sin(ang))
    return newx, newy
```

the car will turn left, and if it's mid-grey, the car continues straight ahead. We could make the cars stick more closely to the centre by making them react quickly, or make them more random by adjusting the steering angle more slowly. A happy medium would be to get the cars mostly sticking to the track but being random enough to make them tricky to overtake.

Our code will need a lot of extra elements to mimic Atari's original game, but this short snippet shows how easily you can get a top-down racing game working in Pygame Zero.



^ Three bitmaps are used for the track. One's visible, and the other two are guides for the cars.

< Four Formula One cars race around the track. Collisions between other cars and the sides of the track are detected.