

^ Three different cabinet styles were available for Rally-X.

< In Namco's original arcade game, the red cars chased the player relentlessly around each level. Note the handy mini-map on the right.

Code a Rally-X-style mini-map



AUTHOR
MARK VANSTONE

Race around using a mini-map for navigation, just like the arcade classic, Rally-X

The original *Rally-X* arcade game blasted onto the market in 1980, at the same time as *Pac-Man* and *Defender*. This was the first year that developer Namco had exported its games outside Japan thanks to the deal it struck with Midway, an American game distributor. The aim of *Rally-X* is to race a car around a maze, avoiding enemy cars while collecting yellow flags – all before your fuel runs out.


The aspect of *Rally-X* that we'll cover here is the mini-map. As the car moves around the maze, its position can be seen relative to the flags on the right of the screen. The main view of the maze only shows a section of the whole map, and scrolls as the car moves, whereas the mini-map shows the whole size of the map but without any of the maze walls – just dots where the car and flags are (and in the original, the enemy cars). In our example, the mini-map is five times smaller than the main map, so it's easy to work out the calculation to translate large map co-ordinates to mini-map co-ordinates.

To set up our *Rally-X* homage in Pygame Zero, we can stick with the default screen size of 800×600. If we use 200 pixels for the side panel, that leaves us with a 600×600 play area. Our player's car will be drawn in the centre of this area at the co-ordinates 300,300. We can use the in-built rotation of the Actor object by setting the angle property of the car. The maze scrolls depending on which direction the car is pointing, and this can be done by having a lookup table in the form of a dictionary list (**directionMap**) where we define x and y increments for each angle the car can travel. When the cursor keys are pressed, the car stays central and the map moves.

To detect the car hitting a wall, we can use a collision map. This isn't a particularly memory-efficient way of doing it, but it's easy to code. We just use a bitmap the same size as the main map which has all the roads as black and all the walls as white. With this map, we can detect if there's a wall in the direction in which the car's moving by testing the pixels directly in front of it. If a wall is

detected, we rotate the car rather than moving it. If we draw the side panel after the main map, we'll then be able to see the full layout of the screen with the map scrolling as the car navigates through the maze.

We can add flags as a list of Actor objects. We could make these random, but for the sake of simplicity, our sample code has them defined in a list of x and y co-ordinates. We need to move the flags with the map, so in each **update()**, we loop through the list and add the same increments to the x and y co-ordinates as the main map. If the car collides with any flags, we just take them off the list of items to draw by adding a **collected** variable. Having put all of this in place, we can draw the mini-map, which will show the car and the flags. All we need to do is divide the object co-ordinates by five and add an x and y offset so that the objects appear in the right place on the mini-map.

And those are the basics of *Rally-X*! All it needs now is a fuel gauge, some enemy cars, and obstacles – but we'll leave those for you to sort out... 



Download
the code
from GitHub:
[wfmag.cc/
wfmag43](https://wfmag.cc/wfmag43)

Rally-X racing in Python

Here's Mark's code for a *Rally-X*-style racer, complete with mini-map. To get it working on your system, you'll need to install Pygame Zero – full instructions are available at wfmag.cc/pgzero.

```
# Rally X
from pygame import image, Color

car = Actor('car', center=(300, 300))
car.angle = 180
mapx = -100
mapy = 0
directionMap = {(0,1), 90:(1,0), 180:(0,-1), 270:(-1,0)}
speed = 5
collisionmap = image.load('images/collisionmap.png')
count = gameStatus = 0
flagsXY=[(200,190),(300,110),(300,300),
(400,600),(600,1600),(800,350)]
flags = []
for f in range(0, 6):
    flags.append(Actor('flag', center=(0, 0)))
    flags[len(flags)-1].collected = False

def draw():
    screen.blit("colourmap", (mapx, mapy))
    car.draw()
    for f in range(0, 6):
        if not flags[f].collected: flags[f].draw()
    screen.blit("sidepanel", (600, 0))
    drawMiniMap()
    if gameStatus == 1: screen.draw.text("YOU GOT
ALL THE FLAGS!", center = (400, 300), owidth=0.5,
ocolor=(255,255,255), color=(0,0,255), fontsize=80)

def update():
    global mapx, mapy, count, gameStatus
    if gameStatus == 0:
        checkInput()
        testmove = (int((-mapx+300) - ((directionMap[car.
angle][0]*8) * speed)), int((-mapy+300) - ((directionMap[car.
angle][1]*8) * speed)))
        if collisionmap.get_at(testmove) == Color('black'):
            mapx += directionMap[car.angle][0] * speed
            mapy += directionMap[car.angle][1] * speed
        else:
            car.angle += 90
            if car.angle == 360: car.angle = 0
            if collisionmap.get_at((int(-mapx+300), int(-
mapy+300))) == Color('white'): mapx += 1
            if collisionmap.get_at((int(-mapx+270), int(-
mapy+300))) == Color('white'): mapx -= 1
            if collisionmap.get_at((int(-mapx+300), int(-
mapy+330))) == Color('white'): mapy += 1
```

```
            if collisionmap.get_at((int(-mapx+300), int(-
mapy+270))) == Color('white'): mapy -= 1
            flagCount = 0
            for f in range(0, 6):
                flags[f].x = flagsXY[f][0]+mapx
                flags[f].y = flagsXY[f][1]+mapy
                if flags[f].colidepoint(car.pos):
                    flags[f].collected = True
                    if flags[f].collected == True: flagCount += 1
                    count += 1
            if flagCount == 6: gameStatus = 1

def checkInput():
    if keyboard.left: car.angle = 90
    if keyboard.right: car.angle = 270
    if keyboard.up: car.angle = 0
    if keyboard.down: car.angle = 180

def drawMiniMap():
    carRect = Rect((658+(-mapx/5), 208+(-mapy/5)), (4, 4))
    if count%10 > 5:
        screen.draw.filled_rect(carRect, (0, 0, 0))
    else:
        screen.draw.filled_rect(carRect, (100, 100, 100))
    for f in range(0, 6):
        if not flags[f].collected:
            flagRect = Rect((600+(flagsXY[f]
[0]/5), 150+(flagsXY[f][1]/5)), (4, 4))
            screen.draw.filled_rect(flagRect, (255, 255, 0))
```

