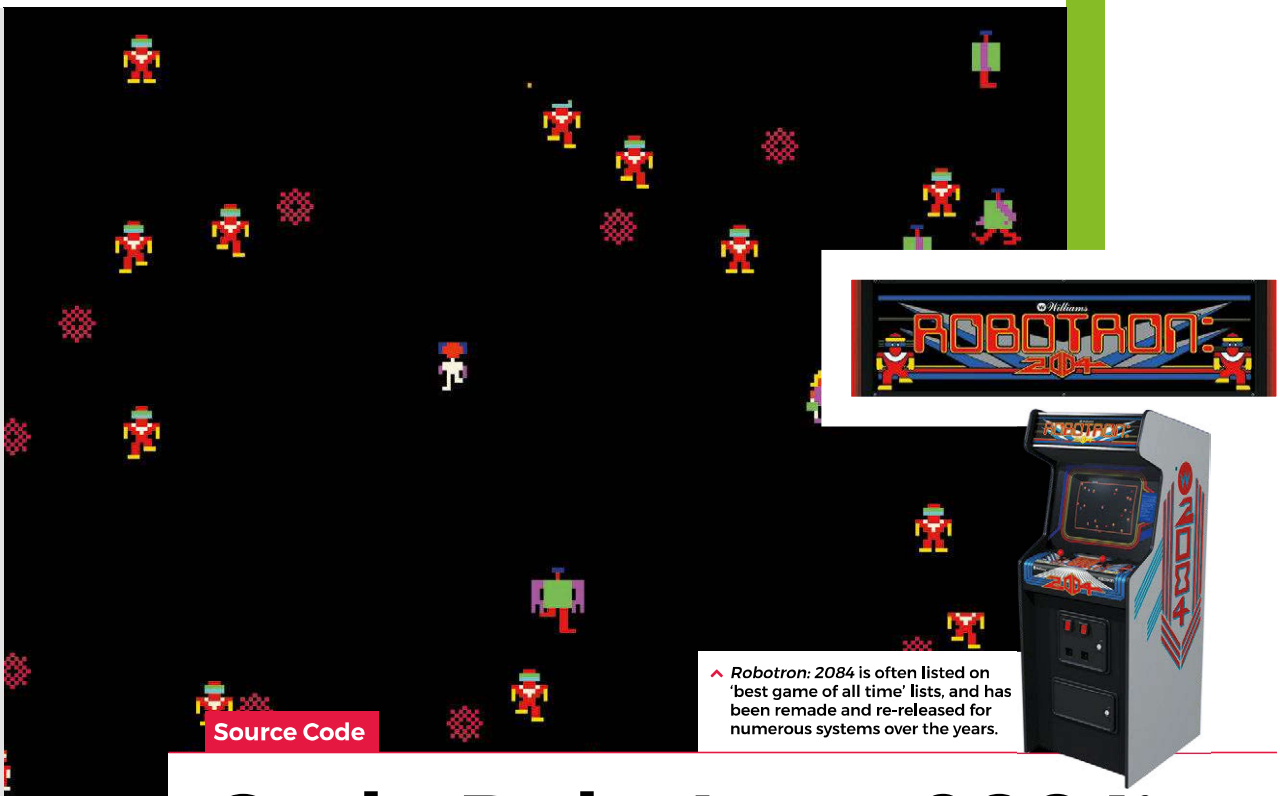

Robotron: 2084 is often listed on 'best game of all time' lists, and has been remade and re-released for numerous systems over the years.

Source Code

Code Robotron: 2084's twin-stick action

Move in one direction and fire in another with Mac's re-creation of an arcade classic

AUTHOR
MAC BOWLEY

Released back in 1982, *Robotron: 2084* popularised the concept of the twin-stick shooter. It gave players two joysticks which allowed them to move in one direction while also shooting at enemies in another. Here, I'll show you how to recreate those controls using Python and Pygame. We don't have access to any sticks, only a keyboard, so we'll be using the arrow keys for movement and **WASD** to control the direction of fire.

The movement controls use a `global` variable, a few `if` statements, and two built-in Pygame functions: `on_key_down` and `on_key_up`. The `on_key_down` function is called when a key on the keyboard is pressed, so when the player presses the right arrow key, for example, I set the x direction of the player to be a positive 1. Instead of setting the movement to 1, instead, I'll add 1 to the direction. The `on_key_down` function is called when a button's released. A key being released means the player doesn't want to travel in that direction anymore and so we should do the opposite of what we did earlier – we take away the 1 or -1 we applied in the `on_key_up` function.

We repeat this process for each arrow key. Moving the player in the `update()` function is the last part of my movement; I apply a move speed and then use a `playArea` rect to clamp the player's position.

Now for the aiming and rotating. When my player aims, I want them to set the direction the bullets will fire, which functions like the movement. The difference this time is that when a player hits an aiming key, I set the direction directly rather than adjusting the values. If my player aims up, and then releases that key, the shooting will stop. Our next challenge is changing this direction into a rotation for the turret. Actors in Pygame can be rotated in degrees, so I have to find a way of turning a pair of x and y directions into a rotation. To do this, I use the math module's `atan2` function to find the arc tangent of two points. The function returns a result in radians, so it needs to be converted. (You'll also notice I had to adjust mine by 90 degrees. If you want to avoid having to do this, create a sprite that faces right by default.)

To fire bullets, I'm using a flag called 'shooting' which, when set to `True`, causes my turret to turn and fire. My bullets are dictionaries; I could have used a class, but the only thing I need to keep track of is an actor and the bullet's direction.

You can look at the `update` function and see how I've implemented a fire rate for the turret as well. You can edit the `update` function to take a single parameter, `dt`, which stores the time since the last frame. By adding these up, you can trigger a bullet at precise intervals and then reset the timer.

This code is just a start – you could add enemies and maybe other player weapons to make a complete shooting experience.

Twin-stick shooting in Python

Here's Mac's code snippet. 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/wfmag38**

```
import pygame as pg
import math

WIDTH = 860
HEIGHT = 540

bg = pg.image.load("images/arena.png").
convert()
play_Area = Rect((150, 75), (560, 390))

player = Actor("treads.png",
center=(WIDTH//2, HEIGHT//2),
anchor=('center', 'center'))
turret = Actor("turret.png",
center=(player.x, player.y),
anchor=('center', 'center'))
pl_movement = [0, 0]
pl_move_speed = 5

pl_rotation = [0, 0]
turn_speed = 5
shooting = False
bullets = []
bullet_speed = 150
fire_rate = 0.15
fire_timer = 0

def on_key_down(key, unicode):
    global shooting

    # Movement
    if key == keys.RIGHT:
        pl_movement[0] += 1
    if key == keys.LEFT:
        pl_movement[0] += -1
    if key == keys.UP:
        pl_movement[1] += -1
    if key == keys.DOWN:
        pl_movement[1] += 1
    if key == keys.D:
        pl_rotation[0] = 1
    if key == keys.A:
        pl_rotation[0] = -1
    if key == keys.W:
        pl_rotation[1] = -1
    if key == keys.S:
        pl_rotation[1] = 1
    print(pl_rotation)

def on_key_up(key):
    global shooting

    # Movement
    if key == keys.RIGHT:
        pl_movement[0] = 0
    if key == keys.LEFT:
        pl_movement[0] = 0
    if key == keys.UP:
        pl_movement[1] = 0
    if key == keys.DOWN:
        pl_movement[1] = 0
    if key == keys.D:
        pl_rotation[0] = 0
    if key == keys.A:
        pl_rotation[0] = 0
    if key == keys.W:
        pl_rotation[1] = 0
    if key == keys.S:
        pl_rotation[1] = 0
def update(dt):
    global shooting, bullets, fire_timer

    # Movement every frame
    player.x += pl_movement[0] * pl_
move_speed
    player.y += pl_movement[1] * pl_
move_speed

    # Clamp the position
    if player.y - 16 < play_Area.top:
        player.y = play_Area.top + 16
    elif player.y + 16 > play_Area.
bottom:
        player.y = play_Area.bottom - 16
    if player.x - 16 < play_Area.left:
        player.x = play_Area.left + 16
    elif player.x + 16 > play_Area.
right:
        player.x = play_Area.right - 16

    turret.pos = player.pos

    if any([keyboard[keys.W],
keyboard[keys.A], keyboard[keys.S],
keyboard[keys.D]]):
        shooting = True
    else:
        shooting = False
        fire_timer = fire_rate

    if shooting == True:

        # Rotate the turret
        desired_angle = (math.atan2(-
pl_rotation[1], pl_rotation[0]) / (math.
pi/180)) - 90
        turret.angle = desired_angle
        fire_timer += dt
        if fire_timer > fire_rate:
            bullet = {}
            bullet["actor"] =
Actor("bullet.png", center=player.pos,
anchor=('center', 'center'))
            bullet["direction"] = pl_
rotation.copy()
            bullet["actor"].x += pl_
rotation[0] * 4
            bullet["actor"].y += pl_
rotation[1] * 4
            bullets.append(bullet)
            fire_timer = 0

    bullets_to_remove = []
    for b in bullets:
        b["actor"].x += b["direction"]
[0] * bullet_speed * dt
        b["actor"].y += b["direction"]
[1] * bullet_speed * dt
        if not b["actor"].
colliderect(play_Area):
            bullets_to_remove.append(b)

    for b in bullets_to_remove:
        bullets.remove(b)

def draw():
    screen.blit(bg, (0, 0))
    player.draw()
    turret.draw()
    for b in bullets:
        b["actor"].draw()
```

The arena background and tank sprites were created in Piskel (**piskelapp.com**). Separate sprites for the tank allow the turret to rotate separately from the tracks.