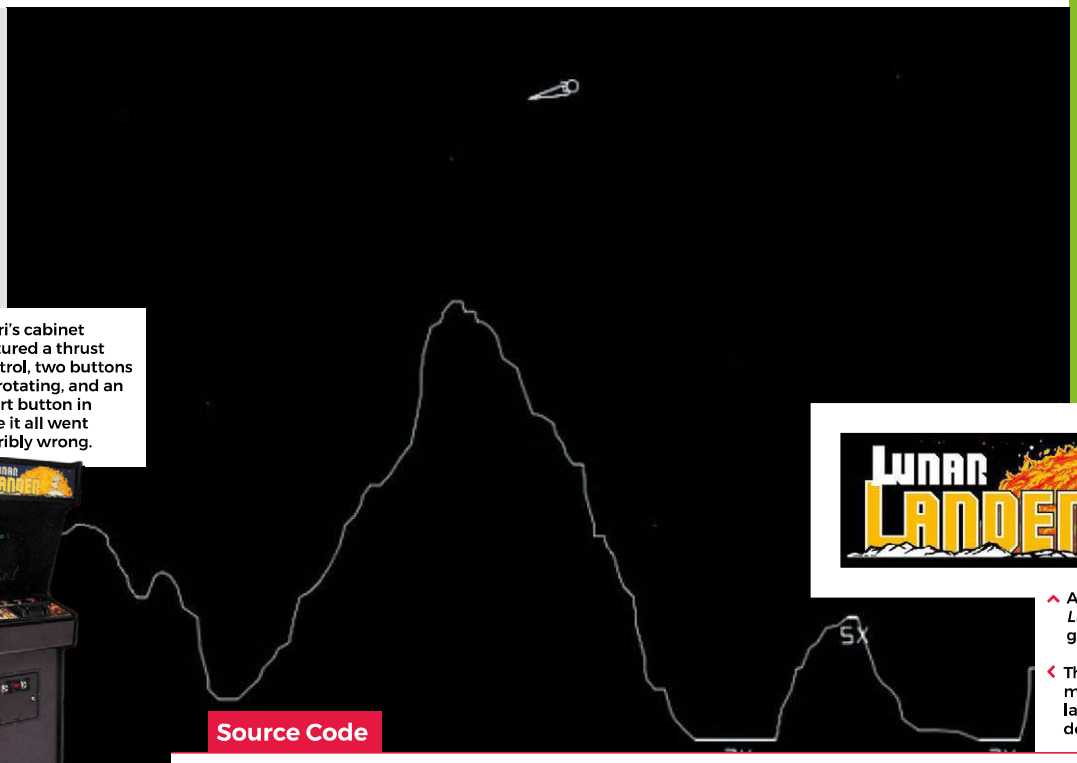

Atari's cabinet featured a thrust control, two buttons for rotating, and an abort button in case it all went horribly wrong.

Atari released the first *Lunar Lander* arcade game in 1979.

The player must manoeuvre the lander to a landing site and carefully descend to the surface.

Source Code

Code a homage to Lunar Lander

Shoot for the moon in Mark's version of an Atari hit

AUTHOR
MARK VANSTONE

First released in 1979 by Atari, *Lunar Lander* was based on a concept created a decade earlier. The original 1969 game (actually called *Lunar*) was a text-based affair that involved controlling a landing module's thrust to guide it safely down to the lunar surface; a later iteration, *Moonlander*, created a more visual iteration of the same idea on the DEC VT50 graphics terminal.

Given that it appeared at the height of the late-seventies arcade boom, though, it was Atari's coin-op that became the most recognisable version of *Lunar Lander*, arriving just after the tenth anniversary of the Apollo 11 moon landing. Again, the aim of the game was to use rotation and thrust controls to guide your craft, and gently set it down on a suitably flat platform. The game required efficient control of the lander, and extra points were awarded for parking successfully on more challenging areas of the landscape.

The arcade cabinet was originally going to feature a normal joystick, but this was changed to a double stalked up-down lever providing variable levels of thrust. The player had to land the craft against the clock with a finite amount of fuel with the Altitude, Horizontal Speed, and Vertical Speed readouts at the top of the screen as a guide. Four levels of difficulty were built into the game, with adjustments to landing controls and landing areas.

To write a game like *Lunar Lander* with Pygame Zero, we can replace the vector graphics with a nice pre-drawn static background and use that as a collision detection mechanism and altitude meter. If our background is just black where the Lander can fly and a different colour anywhere the landscape is, then we can test pixels using the Pygame function `image.get_at()` to see if the lander has landed. We can also test a line of pixels from the Lander down the Y-axis until we hit the landscape, which will give us the lander's altitude.

The rotation controls of the lander are quite simple, as we can capture the left and right arrow keys and increase or decrease the rotation of the lander; however, when thrust is applied (by pressing the up arrow) things get a little more complicated. We need to remember which direction the thrust came from so that the craft will continue to move in that direction even if it is rotated, so we have a direction property attached to our lander object. A little gravity is applied to the position of the lander, and then we just need a little bit of trigonometry to work out the movement of the lander based on its speed and direction of travel.

To judge if the lander has been landed safely or rammed into the lunar surface, we look at the downward speed and angle of the craft as it reaches an altitude of 1. If the speed is sufficiently slow and the angle is near vertical, then we trigger the landed message, and the game ends. If the lander reaches zero altitude without these conditions met, then we register a crash. Other elements that can be added to this sample are things like a limited fuel gauge and variable difficulty levels. You might even try adding the sounds of the rocket booster noise featured on the original arcade game.

Lunar Lander in Python

Here's Mark's code for a simple, modern take on *Lunar Lander*. To get it running on your system, you'll need to install Pygame Zero – full instructions are available at **wfmag.cc/pgzero**.

Download the code from GitHub: **wfmag.cc/wfmag37**

```
import math
from pygame import image, Color
import time
start_time = time.time()
backgroundImage = image.load('images/background.png')
lander = Actor('lander',(50,30))
lander.angle = lander.direction = -80
lander.thrust = 0.5
gravity = 0.8
lander.burn = speedDown = gameState = gameTime = 0

def draw():
    global gameTime
    screen.blit('background',(0,0))
    screen.blit('space',(0,0))
    r = lander.angle
    if(lander.burn > 0):
        lander.image = "landerburn"
    else:
        lander.image = "lander"
    lander.angle = r
    lander.draw()
    if gameState == 0:
        gameTime = int(time.time() - start_time)
    screen.draw.text("Altitude : "+ str(getAlt()),
topleft=(650, 10), owidth=0.5, ocolor=(255,0,0),
color=(255,255,0) , fontsize=25)
    screen.draw.text("Time : "+ str(gameTime), topleft=(40,
10), owidth=0.5, ocolor=(255,0,0), color=(255,255,0) ,
fontsize=25)
    if gameState == 2:
        screen.draw.text("Congratulations \nThe Eagle Has
Landed", center=(400, 50), owidth=0.5, ocolor=(255,0,0),
color=(255,255,0) , fontsize=35)
    if gameState == 1:
        screen.draw.text("Crashed", center=(400, 50),
owidth=0.5, ocolor=(255,0,0), color=(255,255,0) , fontsize=35)

def update():
    global gameState, speedDown
    if gameState == 0:
        if keyboard.up:
            lander.thrust = limit(lander.thrust+0.01,0,1)
            changeDirection()
            lander.burn = 1
        if keyboard.left: lander.angle += 1
        if keyboard.right: lander.angle -= 1
        oldPos = lander.center
        lander.y += gravity
        newPos = calcNewXY(lander.center, lander.thrust, math.
radians(90-lander.direction))
        lander.center = newPos
        speedDown = newPos[1] - oldPos[1]
        lander.thrust = limit(lander.thrust-0.001,0,1)
        lander.burn = limit(lander.burn-0.05,0,1)
        if speedDown < 0.2 and getAlt() == 1 and lander.angle >
-5 and lander.angle < 5:
            gameState = 2
        if getAlt() == 0:
            gameState = 1

def changeDirection():
    if lander.direction > lander.angle: lander.direction -= 1
    if lander.direction < lander.angle: lander.direction += 1

def limit(n, minn, maxn):
    return max(min(maxn, n), minn)

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

def getAlt():
    testY = lander.y+8
    height = 0;
    while testPixel((int(lander.x),int(testY))) ==
Color('black') and height < 600:
        testY += 1
        height += 1
    return height

def testPixel(xy):
    if xy[0] >= 0 and xy[0] < 800 and xy[1] >= 0 and xy[1] <
600:
        return
backgroundImage.get_at(xy)
    else:
        return
Color('black')
```

› Our homage to the classic *Lunar Lander*. Can you land without causing millions of dollars' worth of damage?

ENGAGE

The direction of thrust could be done in several ways. In this case, we've kept it simple, with one directional value which gradually moves in a new direction when an alternative thrust is applied. You may want to try making an X- and Y-axis direction calculation for thrust so that values are a combination of the two dimensions. You could also add joystick control to provide variable thrust input.