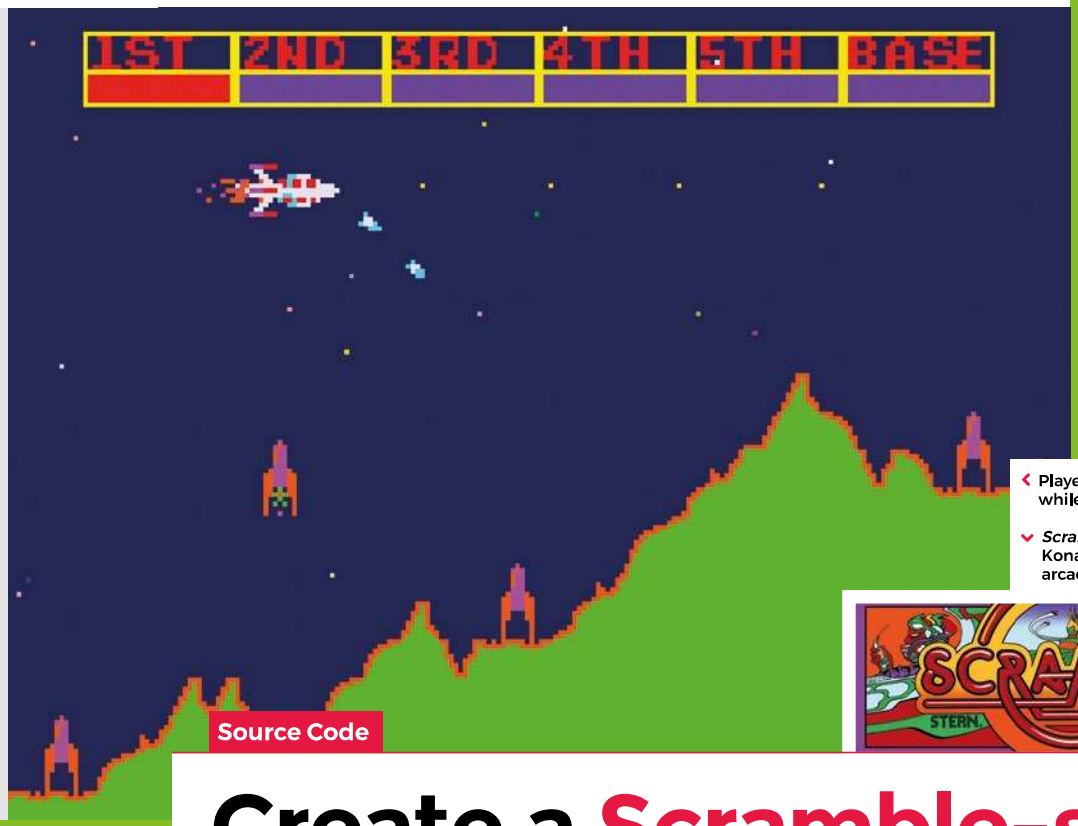

Players must avoid the terrain while dodging enemy rockets.

Scramble was developed by Konami and released in arcades in 1981.

Source Code

Create a Scramble-style scrolling landscape

Weave through a randomly generated landscape in Mark's homage to the classic Scramble

AUTHOR
MARK VANSTONE

In the early eighties, arcades and sports halls rang with the sound of a multitude of video games. Because home computers hadn't yet made it into most households, the only option for the avid video gamer was to get down to the local entertainment establishment and feed the machines with ten pence pieces (which were bigger then). One of these pocket money-hungry machines was Konami's *Scramble* – released in 1981, it was one of the earliest side-scrolling shooters with multiple levels.

The player's jet aircraft flies across a randomly generated landscape (which sometimes narrows to a cave system), avoiding obstacles and enemy planes, bombing targets on the ground, and trying not to crash. As the game continues, the difficulty increases. The player aircraft can only fly forward, so once a target has been passed, there's no turning back for a second go.

In this example code, I'll show you a way to generate a *Scramble*-style scrolling landscape using Pygame Zero and a couple of additional Pygame functions. With early computers, moving a lot of data around the screen was very slow until dedicated video hardware like the blitter chip arrived. Scrolling, however, could be achieved either by a quick shuffle of bytes to the left or right in the video memory – or in some cases, changing the start address of the video memory, which was even quicker.

> **"With early computers, moving a lot of data around the screen was very slow"**

For our scrolling, we can use a Pygame surface the same size as the screen. To get the scrolling effect, we just call the `scroll()` function on the surface to shift everything left by one pixel and then draw a new pixel-wide slice of the terrain. The terrain could just be a single colour, but I've included a bit of maths-based RGB tinkering to make it more colourful. We can draw our terrain surface over a background image, as the SRCALPHA flag is set when we create the surface. This is also useful for detecting if the jet has hit the terrain. We can test the pixel from the surface in front of the jet; if it's not transparent, kaboom!

The jet itself is a Pygame Zero Actor and can be moved up and down with the arrow keys. The left and right arrows increase and decrease the speed. We generate the landscape in the `updateLand()` and `drawLand()` functions where `updateLand()` decides if the landscape is inclining or declining (and the same with the roof), making sure that the roof and floor don't get too close, then it scrolls everything left. The `drawLand()` function then draws pixels

Scrolling caverns in Python

Here's a code snippet that recreates *Scramble*'s scrolling terrain 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/wfmag22**

```
from random import randint
from pygame import Surface
from pygame.locals import *
import math

scrambleSurface = Surface((800,600),SRCALPHA)
landLevel = 600
roofLevel = 10
landChange = -3
roofChange = 3
jet = Actor('jet',(400,300))
speed = 3
crash = False

def draw():
    if crash: # remove the next line if you are affected by
flashing lights
        screen.fill((randint(100,200),0,0))
        screen.blit(scrambleSurface, (0, 0))
        jet.draw()
    else:
        screen.blit('space',(0,0))
        screen.blit(scrambleSurface, (0, 0))
        jet.draw()

def update():
    global speed, crash
    if crash == False:
        if keyboard.up: jet.y -= speed
        if keyboard.down: jet.y += speed
        if keyboard.left: speed = limit(speed-0.1,1,10)
        if keyboard.right: speed = limit(speed+0.1,1,10)
        jet.x = 310 + (speed * 30)
        for _ in range(math.ceil(speed)):
            updateLand()
        if scrambleSurface.get_at((math.ceil(jet.x+32),math.
ceil(jet.y))) != (0,0,0,0):
            crash = True;

def updateLand():
    global landLevel, landChange, roofLevel, roofChange
    if randint(0,10) == 3: roofChange = randint(0,6) - 3
    if randint(0,10) == 3: landChange = randint(0,6) - 3
    roofLevel += roofChange
    landLevel += landChange
    landLevel = limit(landLevel,200,590)
    roofLevel = limit(roofLevel,10,400)
    if roofLevel > landLevel-200: roofLevel = landLevel-200
    scrambleSurface.scroll(-1,0)
    drawLand()

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

def drawLand():
    for i in range(0, 600):
        c = (0,0,0,0)
        if i > landLevel:
            g = limit(i-landLevel,0,255)
            c = (255,g,0)
        else:
            if i < roofLevel:
                r = limit(roofLevel-i,0,255)
                c = (255,r,0)
        scrambleSurface.set_at((799,i),c)
```

at the right-hand edge of the surface from y coordinates 0 to 600, drawing a thin sliver of roof, open space, and floor. The speed of the jet determines how many times the landscape is updated in each draw cycle, so at faster speeds, many lines of pixels are added to the right-hand side before the display updates.

The use of `randint()` can be changed to create a more or less jagged landscape, and the gap between roof and floor could also be adjusted for more difficulty. The original game also had enemy aircraft, which you could make with Actors, and fuel tanks on the ground, which could be created on the right-hand side as the terrain comes into view and then moved as the surface scrolls. *Scramble* sparked a wave of horizontal shooters, from both Konami and rival companies; this short piece of code could give you the basis for making a decent *Scramble* clone of your own.

Avoid the roof and the floor with the arrow keys. Jet graphic courtesy of TheSource4Life at **opengameart.org.**

COLLECTING SCRAMBLE

In 1982, TOMY released a handheld version of *Scramble*, while Grandstand sold a chunky tabletop edition that you can still find for sale on auction sites. If you'd like something a little more up-to-date, though, Konami released *Scramble* as part of their *Konami Collector's Series: Arcade Classics* compilation for the Game Boy Advance. Or, if you have plenty of spare space, you could buy one of the original arcade machines – you'll find working examples for sale at around £1600. With prices like that, you might want to use our code and create your own *Scramble* – it'll save you a fortune.