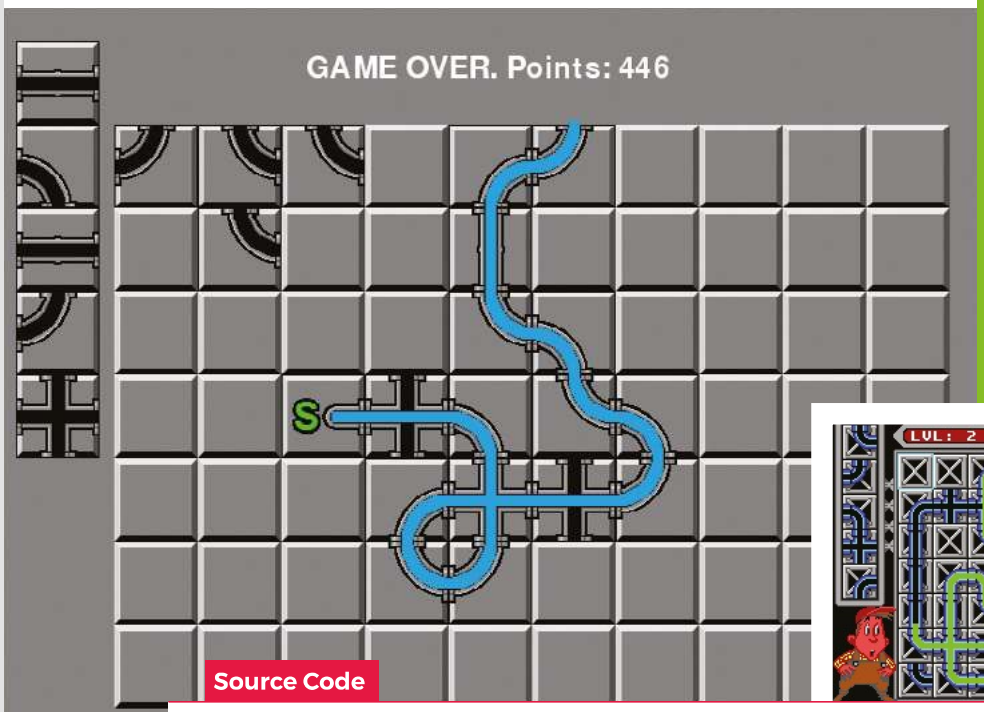

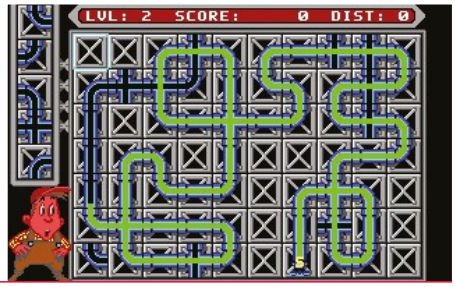

Our *Pipe Mania* homage. Build a pipeline before the water escapes, and see if you can beat your own score.

Pipe Mania's design is so effective, it's appeared in various guises elsewhere – even as a minigame in *BioShock*.

Source Code

Code your own Pipe Mania puzzler

Create a network of pipes before the water starts to flow in our re-creation of a classic puzzler

AUTHOR
JORDI SANTONJA

Pipe Mania, also called *Pipe Dream* in the US, is a puzzle game developed by The Assembly Line in 1989 for Amiga, Atari ST, and PC, and later ported to other platforms, including arcades. The player must place randomly generated sections of pipe onto a grid. When a counter reaches zero, water starts to flow and must reach the longest possible distance through the connected pipes.

Let's look at how to recreate *Pipe Dream* in Python and Pygame Zero. The variable `start` is decremented at each frame. It begins with a value of `60*30`, so it reaches zero after 30 seconds if our monitor runs at 60 frames per second. In that time, the player can place tiles on the grid to build a path. Every time the user clicks on the grid, the last tile from `nextTiles` is placed on the play area and a new random tile appears at the top of the next tiles. `randint(2,8)` computes a random value between 2 and 8.

`grid` and `nextTiles` are lists of tile values, from 0 to 8, and are copied to the screen in the `draw` function with the `screen.blit` operation. `grid` is a two-dimensional list, with sizes `gridWidth=10` and `gridHeight=7`. Every pipe piece is placed in `grid` with a mouse click. This is managed with the Pygame functions `on_mouse_move` and `on_mouse_down`, where the variable `pos` contains the mouse position in the window. `panelPosition` defines the position of the top-left corner of the grid in the window. To get the grid cell, `panelPosition` is subtracted from `pos`, and the result is divided by `tileSize` with the integer division `//`. `tileMouse` stores the resulting cell element, but it is set to `(-1,-1)` when the mouse lies outside the grid.

The **images** folder contains the PNGs with the tile images, two for every tile: the graphical image and the path image. The `tiles` list contains the name of every tile, and adding to it `_block` or `_path` obtains the name of the file. The values stored in `nextTiles` and `grid` are the indexes of the elements in `tiles`.

The image `waterPath` isn't shown to the user, but it stores the paths that the water is going to follow. The first point of the water path is located in the starting tile, and it's stored in `currentPoint`. `update` calls the function `CheckNextPointDeleteCurrent`, when the water starts flowing. That function finds the next point in the water path, erases it, and adds a new point to the `waterFlow` list. `waterFlow` is shown to the user in the `draw` function.

`pointsToCheck` contains a list of relative positions, offsets, that define a step of two pixels from `currentPoint` in every direction to find the next point. Why two pixels? To be able to define the 'cross' tile, where two lines cross each other. In a 'cross' tile the water flow must follow a straight line, and this is how the only points found are the next points in the same direction. When no next point is found, the game ends and the score is shown: the number of points in the water path, `playState` is set to `0`, and no more updates are done.

Pipe-wrangling in Python

Here's Jordi's code for a *Pipe Mania*-style puzzler. To get it working on your system, you'll need to install Pygame Zero – full instructions are available at **wfmag.cc/pgzero**.


```
# Pipe Mania
from pygame import image, Color, Surface
from random import randint

gridWidth, gridHeight = 10, 7
grid = [[0 for x in range(gridWidth)] for y in range(gridHeight)]
tileSize = 68
panelPosition = (96, 96)
numberNextTiles = 5
nextTiles = [randint(2, 8) for y in range(numberNextTiles)]
nextTilesPosition = (16, 28)
tileMouse = (-1, -1)

tiles = ['empty', 'start',
        'hori', 'vert', 'cross',
        'bottomleft', 'bottomright',
        'topleft', 'topright']

pathTiles = [image.load('images/'+tiles[i]+'_path.png') for i in
range(1,9)]

waterPath = Surface((gridWidth*tileSize, gridHeight*tileSize))
waterPath.fill(Color('black'))
grid[3][2] = 1 # start tile
waterPath.blit(pathTiles[0], (2 * tileSize, 3 * tileSize))
currentPoint = (2 * tileSize + 43, 3 * tileSize + 34)
waterFlow = []
start = 60*30 # 30 seconds

playState = 1

pointsToCheck = [(2, 0),( 0,2),(-2, 0),( 0,-2),
                 (2, 1),( 1,2),(-2, 1),( 1,-2),
                 (2,-1),(-1,2),(-2,-1),(-1,-2),
                 (2,-2),( 2,2),(-2, 2),(-2,-2)]

def draw():
    screen.blit('background', (0,0))
    for x in range(gridWidth):
        for y in range(gridHeight):
            screen.blit(tiles[grid[y][x]]+'_block', (
                panelPosition[0] + x * tileSize,
                panelPosition[1] + y * tileSize))
    for y in range(numberNextTiles):
        screen.blit(tiles[nextTiles[y]]+'_block', (
            nextTilesPosition[0],
            nextTilesPosition[1] + y * tileSize))
    for point in waterFlow:
        screen.blit('water', point)
    if playState == 1:
        if tileMouse[0] >= 0 and tileMouse[1] >= 0:
            screen.blit(tiles[nextTiles[-1]]+'_block', (
                panelPosition[0] + tileMouse[0] * tileSize,
                panelPosition[1] + tileMouse[1] * tileSize))
        if start > 0:
            screen.draw.text("Start in "
+ str(start // 60), center=(400, 50), fontsize=35)
    else:
        screen.draw.text("GAME OVER. Points:
"+str(len(waterFlow)), center=(400, 50), fontsize=35)

def update():
    global start, playState
    if start > 0:
        start -= 1
    elif playState == 1:
        if not CheckNextPointDeleteCurrent():
            playState = 0

def CheckNextPointDeleteCurrent():
    global currentPoint
    for point in pointsToCheck:
        newPoint = (currentPoint[0] + point[0], currentPoint[1]
+ point[1])
        if newPoint[0] < 0 or newPoint[1] < 0 or newPoint[0] >=
gridWidth*tileSize or newPoint[1] >= gridHeight*tileSize:
            return False # goes outside the screen
        if waterPath.get_at(newPoint) != Color('black'):
            waterPath.set_at(newPoint, Color('black'))
            waterFlow.append((newPoint[0] + panelPosition[0] - 4,
newPoint[1] + panelPosition[1] - 4))
            currentPoint = newPoint
            return True
    return False # no next point found


def on_mouse_down(pos):
    if playState == 1 and tileMouse[0] >= 0 and tileMouse[1] >=
0:
        if grid[tileMouse[1]][tileMouse[0]] != 1: # not start
tile
            grid[tileMouse[1]][tileMouse[0]] = nextTiles[-1]
            waterPath.fill(Color('black'), (tileMouse[0] *
tileSize, tileMouse[1] * tileSize, tileSize, tileSize))
            waterPath.blit(pathTiles[nextTiles[-1] - 1],
(tileMouse[0] * tileSize, tileMouse[1] * tileSize))
            for i in reversed(range(numberNextTiles - 1)):
                nextTiles[i + 1] = nextTiles[i]
            nextTiles[0] = randint(2, 8)

def on_mouse_move(pos):
    global tileMouse
    if playState == 1:
        tileMouse = ((pos[0] - panelPosition[0])//tileSize,
(pos[1] - panelPosition[1])//tileSize)
        if pos[0] < panelPosition[0] or pos[1] < panelPosition[1]
or tileMouse[0] >= gridWidth or tileMouse[1] >= gridHeight:
            tileMouse = (-1, -1) # mouse outside panel
```