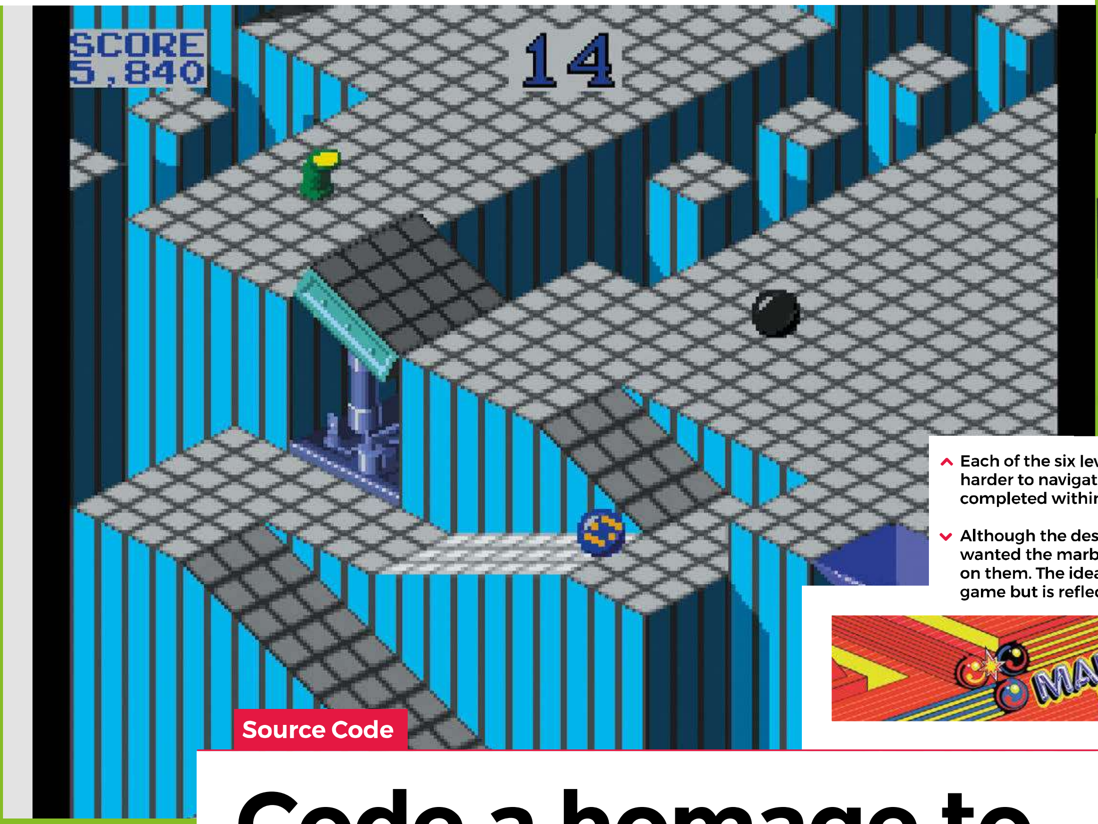


^ Each of the six levels got progressively harder to navigate and had to be completed within a time limit.

v Although the designer was against it, Atari wanted the marbles to have smiley faces on them. The idea didn't make it to the game but is reflected in the game logo.

Source Code

Code a homage to Marble Madness

Code the map and movement basics of the innovative marble-rolling arcade game

AUTHOR
MARK VANSTONE

Hitting arcades in 1984, Atari's *Marble Madness* presented a rather different control mechanism than other games of the time. The original arcade cabinet provided players with a trackball controller rather than a conventional joystick, and the aim was to guide a marble through a three-dimensional course in the fastest possible time. This meant that a player could change the angle and speed of the marble as it rolled and avoid various obstacles and baddies.

During development, designer Mark Cerny had to shelve numerous ideas for *Marble Madness*, since the hardware just wasn't able to achieve the level of detail and interaction he wanted. The groundbreaking 3D display was one idea that made it through to the finished game: its pre-rendered, ray-traced isometric levels.

Marble Madness was the first game to use Atari's System 1 upgradeable hardware platform, and also boasted the first use of an FM sound chip produced by Yamaha to create its distinctive stereo music. The game was popular in arcades to start with, but interest appeared to drop off after a few months – something Cerny attributed to the fact that the game didn't take long to play.

"The ball physics are calculated from the grey-shaded heightmap"

Marble Madness's popularity endured in the home market, though, with ports made for most computers and consoles of the time – although inevitably, most of these didn't support the original's trackball controls.

For our version of *Marble Madness*, we're going to use a combination of a rendered background and a heightmap in Pygame Zero, and write some simple physics code to simulate the marble rolling over the terrain's flats and slopes. We can produce the background graphic using a 3D modelling program such as Blender. The camera needs to be set to Orthographic to get the forced perspective look we're after. The angle of the camera is also important, in that we need an X rotation of 54.7 degrees and a Y rotation of 45 degrees to get the lines of the terrain correct. The heightmap can be derived from an overhead view of the terrain, but you'll probably want to draw the heights of the blocks in a drawing package such as GIMP to give you precise colour values on the map.

The ball rolling physics are calculated from the grey-shaded heightmap graphic. We've left a debug mode in the code; by changing the debug variable to `True`, you can see how the marble moves over the terrain from the overhead viewpoint of the heightmap. The player can move the marble left and right with the arrow keys – on a level surface it will gradually slow down if no keys are pressed. If the marble is on a gradient

Download the code from GitHub: **wfmag.cc/wfmag34**

Rolling marbles in Python

To get Mark's code running on your system, you'll need to install Pygame Zero – all instructions are at **wfmag.cc/pgzero**.

```
# Marble Madness
from pygame import image

HEIGHT = 570
WIDTH = 600
gameState = 0
marble = Actor(‘marble’, center=(300, 45))
marbleh = Actor(‘marbleh’, center=(300, 60))
marble.dir = marble.speed = 0
heightmap = image.load(‘images/height45.png’)
# set debug variable below to True for debug mode
debug = False

def draw():
    if(debug):
        screen.blit(“height45”, (0, 0))
        marbleh.draw()
    else:
        screen.blit(“map”, (0, 0))
        if gameState == 0:
            marble.draw()
        else:
            if gameState == 2:
                screen.draw.text(“YOU WIN!”, center = (300,
300), owidth=0.5, ocolor=(255,255,255), color=(0,0,255) ,
fontsize=80)
                marble.draw()
            else:
                screen.draw.text(“GAME OVER!”, center = (300,
300), owidth=0.5, ocolor=(255,255,255), color=(0,0,255) ,
fontsize=80)
        screen.blit(“overlay”, (0, 0))
```

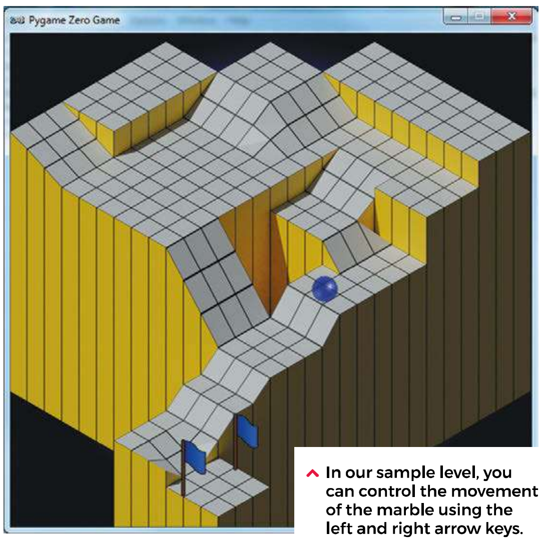
```
def update():
    if gameState == 0:
        if keyboard.left:
            marble.dir = max(marble.dir-0.1,-1)
            marble.speed = min(1,marble.speed + 0.1)
        if keyboard.right:
            marble.dir = min(marble.dir+0.1,1)
            marble.speed = min(1,marble.speed + 0.1)
        moveMarble()
        marble.speed = max(0,marble.speed - 0.01)

def moveMarble():
    global gameState
    ccol = getHeight(marbleh.x,marbleh.y)
    lcol = getHeight(marbleh.x-10,marbleh.y+10)
    rcol = getHeight(marbleh.x+10,marbleh.y+10)
    if ccol.r == 0:
        gameState = 1
    if (lcol.r < ccol.r or rcol.r < ccol.r):
        marble.y += marble.speed
        marble.speed += 0.03
    marbleh.x += marble.speed*marble.dir
    marbleh.y += marble.speed
    marble.x = marbleh.x
    marble.y = (marbleh.y*0.6)+((255-ccol.r)*1.25)
    marble.angle = marble.angle + marble.speed*marble.dir*-10
    if marble.angle > 0 : marble.angle = -50
    if marble.angle < -50 : marble.angle = 0
    if marbleh.y > 610: gameState = 2

def getHeight(x,y):
    return heightmap.get_at((int(x),int(y)))
```

on the heightmap, it will increase speed in the direction of the gradient. If the marble hits a section of black on the heightmap, it falls out of play, and we stop the game.

That takes care of the movement of the marble in two dimensions, but now we have to translate this to the rendered background's terrain. The way we do this is to translate the Y coordinate of the marble as if the landscape was all at the same level – we multiply it by 0.6 – and then move it down the screen according to the heightmap data, which in this case moves the marble down 1.25 pixels for each shade of colour. We can use an overlay for items the marble always rolls behind, such as the finish flag. And with that, we have the basics of a *Marble Madness* level.



In our sample level, you can control the movement of the marble using the left and right arrow keys.

Module Madness

We use the image module from Pygame to sample the colour of the pixel directly under the marble on the heightmap. We also take samples from the left diagonal and the right diagonal to see if there is a change of height. We are only checking for left and right movement, but this sample could be expanded to deal with the two other directions and moving up the gradients, too. Other obstacles and enemies can be added using the same heightmap translations used for the marble, and other overlay objects can be added to the overlay graphic.