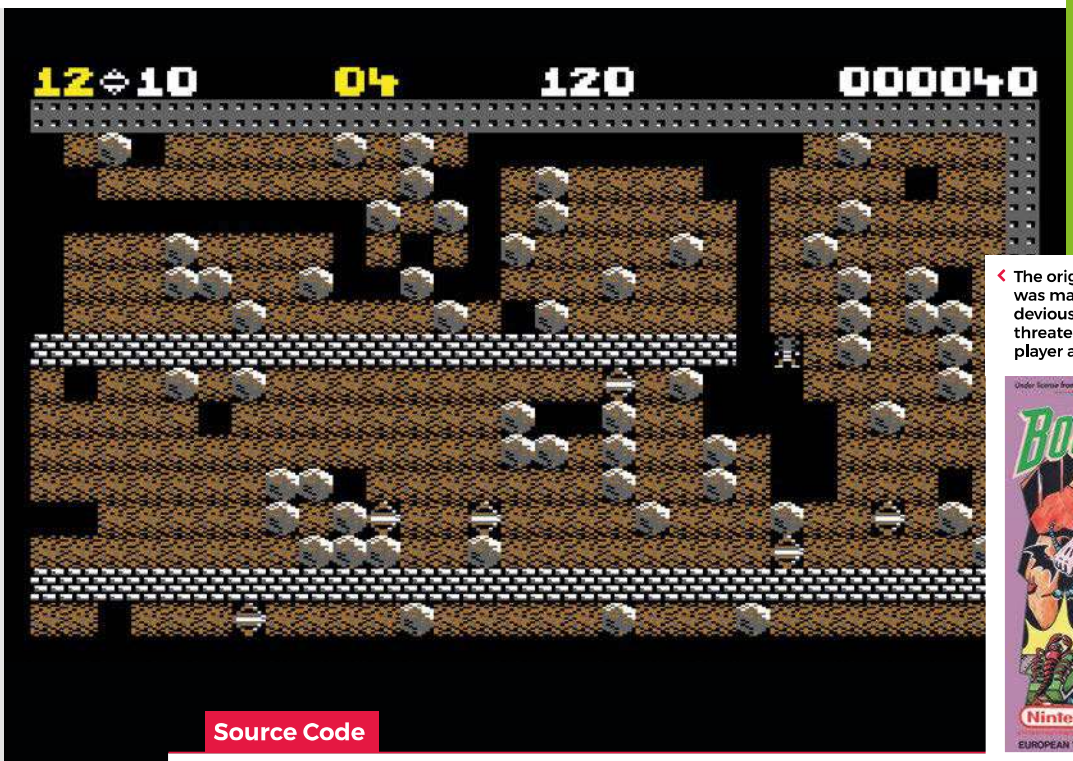


The original *Boulder Dash* was marked out by some devious level design, which threatened to squash the player at every turn.

Source Code

Code a Boulder Dash mining game

Dig through the caves to find gems – but watch out for falling boulders

AUTHOR
MARK VANSTONE

Boulder Dash first appeared in 1984 for the Commodore 64, Apple II, and the Atari 400/800. It featured an energetic gem collector called Rockford who, thanks to some rather low-resolution graphics, looked a bit like an alien. His mission was to tunnel his way through a series of caves to find gems while avoiding falling rocks dislodged by his digging. Deadly creatures also inhabited the caves which, if destroyed by dropping rocks on them, turned into gems for Rockford to collect.

The ingenious level designs were what made *Boulder Dash* so addictive. Gems had to be collected within a time limit to unlock the exit, but some were positioned in places that would need planning to get to, often using the physics of falling boulders to block or clear areas. Of course, the puzzles got increasingly tough as the levels progressed.

Written by Peter Liepa and Chris Gray, *Boulder Dash* was published by First Star Software, which still puts out new versions of the game to this day. Due to its original success, *Boulder Dash* was ported to all kinds of platforms, and the years since have seen no fewer than 20 new iterations of *Boulder Dash*, and a fair few clones, too.

We're going to have a look at the boulder physics aspect of the game, and make a simple level where Rockford can dig out some gems and hopefully not get flattened under an avalanche of rocks. Writing our code in Pygame Zero, we'll automatically create an 800 by 600-size window to work with. We can make our game screen by defining a two-dimensional list, which, in this case, we will fill with soil squares and randomly position the rocks and gems. Each location in the list matrix will have a name: either `wall` for the outside boundary, `soil` for the diggable stuff, `rock` for a round, moveable boulder, `gem` for a collectable item, and finally, `rockford` to symbolise our hero. We can also define an Actor for Rockford, as this will make things like switching images and tracking other properties easier.

Our `draw()` function is just a nested loop to iterate through the list matrix and blit to the screen whatever is indicated in each square. The Rockford Actor is then drawn over the top. We can also keep a count of how many gems have been collected and provide a congratulatory message if all of them are found. In the `update()` function, there are only two things we really need to worry about: the first being to check for keypresses from the player and move Rockford accordingly, and the second to check rocks to see if they need to move.

Rockford is quite easy to test for movement, as he can only move onto an empty square – a soil square or a gem square. It's also possible for him to push a boulder if there's an empty space on the other side. For the boulders, we need to first test if there's an empty space below it, and

Tumbling rocks in Python

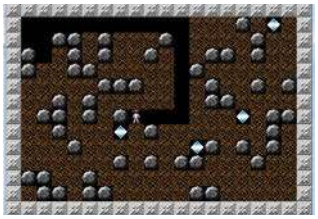
Here's Mark's code snippet, which creates some falling *Boulder Dash* rocks – and an intrepid explorer – in Python. To get it running on your system, you'll need to install Pygame Zero – you can find full instructions at **wfmag.cc/pgzero**

Download the code from GitHub: **wfmag.cc/wfmag30**

```
import random

rockford = Actor('rockford-1', center=(60, 100))
gameState = count = 0
items = [[] for _ in range(14)]
gems = collected = 0
for r in range(0, 14):
    for c in range(0, 20):
        itype = "soil"
        if(r == 0 or r == 13 or c == 0 or c == 19): itype =
"wall"
        elif random.randint(0, 4) == 1: itype = "rock"
        elif random.randint(0, 20) == 1:
            itype = "gem"
            gems += 1
        items[r].append(itype)
items[1][1] = "rockford"

def draw():
    screen.fill((0,0,0))
    if gems == collected: infoText("YOU COLLECTED ALL THE
GEMS!")
    else: infoText("GEMS : "+ str(collected))
    for r in range(0, 14):
        for c in range(0, 20):
            if items[r][c] != "" and items[r][c] !=
"rockford":
                screen.blit(items[r][c], ((c*40), 40+(r*40)))
    if gameState == 0 or (gameState == 1 and count%4 == 0):
rockford.draw()

def update():
    global count
    mx = my = 0
    if count%10 == 0:
        for r in range(13, -1, -1):
            for c in range(19, -1, -1):
                if items[r][c] == "rockford":
                    if keyboard.left: mx = -1
                    if keyboard.right: mx = 1
                    if keyboard.up: my = -1
                    if keyboard.down: my = 1
                if items[r][c] == "rock": testRock(r,c)
        rockford.image = "rockford"+str(mx)
        if gameState == 0: moveRockford(mx,my)
    count += 1

def infoText(t):
    screen.draw.text(t, center = (400, 20), owidth=0.5,
ocolor=(255,255,255), color=(255,0,255) , fontsize=40)

def moveRockford(x,y):
    global collected
    rx, ry = int((rockford.x-20)/40), int((rockford.y-40)/40)
    if items[ry+y][rx+x] != "rock" and items[ry+y][rx+x] !=
"wall":
        if items[ry+y][rx+x] == "gem": collected +=1
        items[ry][rx], items[ry+y][rx+x] = "", "rockford"
        rockford.pos = (rockford.x + (x*40), rockford.y +
(y*40))
    if items[ry+y][rx+x] == "rock" and y == 0:
        if items[ry][rx+(x*2)] == "":
            items[ry][rx], items[ry][rx+(x*2)], items[ry+y]
[rx+x] = "", "rock", "rockford"
            rockford.x += x*40

def testRock(r,c):
    if items[r+1][c] == "":
        moveRock(r,c,r+1,c)
    elif items[r+1][c] == "rock" and items[r+1][c-1] == "" and
items[r][c-1] == "":
        moveRock(r,c,r+1,c-1)
    elif items[r+1][c] == "rock" and items[r+1][c+1] == "" and
items[r][c+1] == "":
        moveRock(r,c,r+1,c+1)

def moveRock(r1,c1,r2,c2):
    global gameState
    items[r1][c1], items[r2][c2] = "", items[r1][c1]
    if items[r2+1][c2] == "rockford": gameState = 1
```

if so, the boulder must move downwards. We also test to see if a boulder is on top of another boulder – if it is, the top boulder can roll off and down onto a space either to the left or the right of the one beneath.

There's not much to add to this snippet of code to turn it into a playable game of *Boulder Dash*. See if you can add a timer, some monsters, and, of course, some puzzles for players to solve on each level.



Our homage to *Boulder Dash* running in Pygame Zero. Dig through the caves to find gems – while avoiding death from above.

BOTTOMS UP

An important thing to notice about the process of scanning through the list matrix to test for boulder movement is that we need to read the list from the bottom upwards; otherwise, because the boulders move downwards, we may end up testing a boulder multiple times if we test from the beginning to the end of the list. Similarly, if we read the list matrix from the top down, we may end up moving a boulder down and then when reading the next row, coming across the same one again, and moving it a second time.