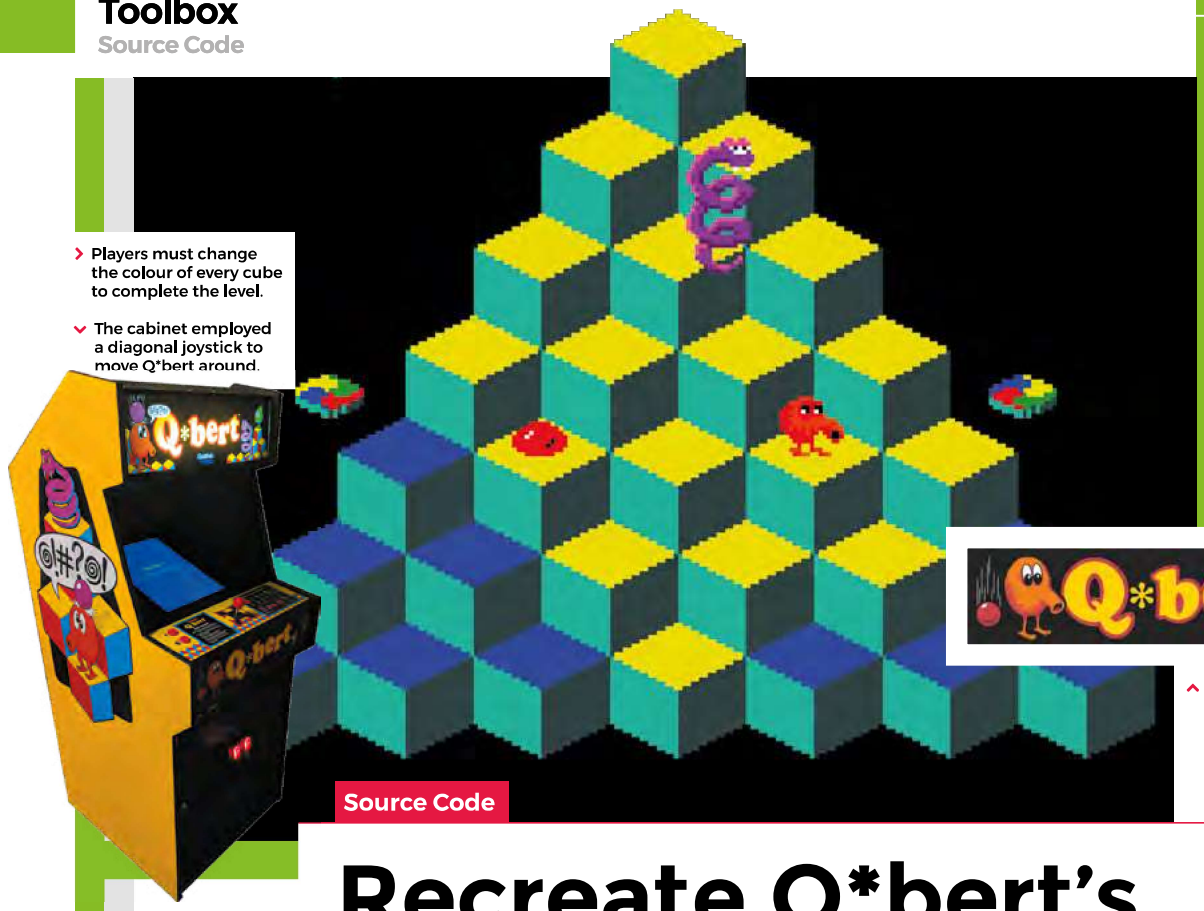


Players must change the colour of every cube to complete the level.

The cabinet employed a diagonal joystick to move Q*bert around.

It was probably just as well, considering how popular the game would become, that Jeff Lee went with *Q*bert* instead of his initial idea: *Snots And Boogers*.

Source Code

Recreate Q*bert's cube-hopping action

Code the mechanics of an eighties arcade hit

AUTHOR
MARK VANSTONE

Late in 1982, a funny little orange character with a big nose landed in arcades. The titular Q*bert's task was to jump around a network of cubes arranged in a pyramid formation, changing the colours of each as they went. Once the cubes were all the same colour, it was on to the next level; to make things more interesting, there were enemies like Coily the snake, and objects which helped Q*bert: some froze enemies in their tracks, while floating discs provided a lift back to the top of the stage.

*Q*bert* was designed by Warren Davis and Jeff Lee at the American company Gottlieb, and soon became such a smash hit that, the following year, it was already being ported to most of the home computer platforms available at the time. New versions and remakes continued to appear for years afterwards, with a mobile phone version appearing in 2003. *Q*bert* was by far Gottlieb's most popular game, and after several changes in company ownership, the firm is now part of Sony's catalogue – *Q*bert*'s main character even made its way into the 2015 film, *Pixels*.

*Q*bert* uses isometric-style graphics to draw a pseudo-3D display – something we can easily replicate in Pygame Zero by using a single cube graphic with which we make a pyramid of Actor objects. Starting with seven cubes on the bottom row, we can create a simple double loop to create the pile of cubes. Our Q*bert character will be another Actor object which we'll position at the top of the pile to start. The game screen can then be displayed in the `draw()` function by looping through our 28 cube Actors and then drawing Q*bert.

We need to detect player input, and for this we use the built-in keyboard object and check the cursor keys in our `update()` function. We need to make Q*bert move from cube to cube so we can move the Actor 32 pixels on the x-axis and 48 pixels on the y-axis. If we do this in steps of 2 for x and 3 for y, we will have Q*bert on the next cube in 16 steps. We can also change his image to point in the right direction depending on the key pressed in our `jump()` function. If we use this linear movement in our `move()` function, we'll see the Actor go in a straight line to the next block. To add a bit of bounce to Q*bert's movement, we add or subtract (depending on the direction) the values in the `bounce[]` list. This will make a bit more of a curved movement to the animation.

Now that we have our long-nosed friend jumping around, we need to check where he's landing. We can loop through the cube positions and check whether Q*bert is over each one. If he is, then we change the image of the cube to one with a yellow top. If we don't detect a cube under Q*bert, then the critter's jumped off the pyramid, and the game's over. We can then do a quick loop through all the cube Actors, and if they've all been changed, then the player has completed the level. So those are the basic mechanics of jumping around on a pyramid of cubes. We just need some snakes and other baddies to annoy Q*bert – but we'll leave those for you to add. Good luck!

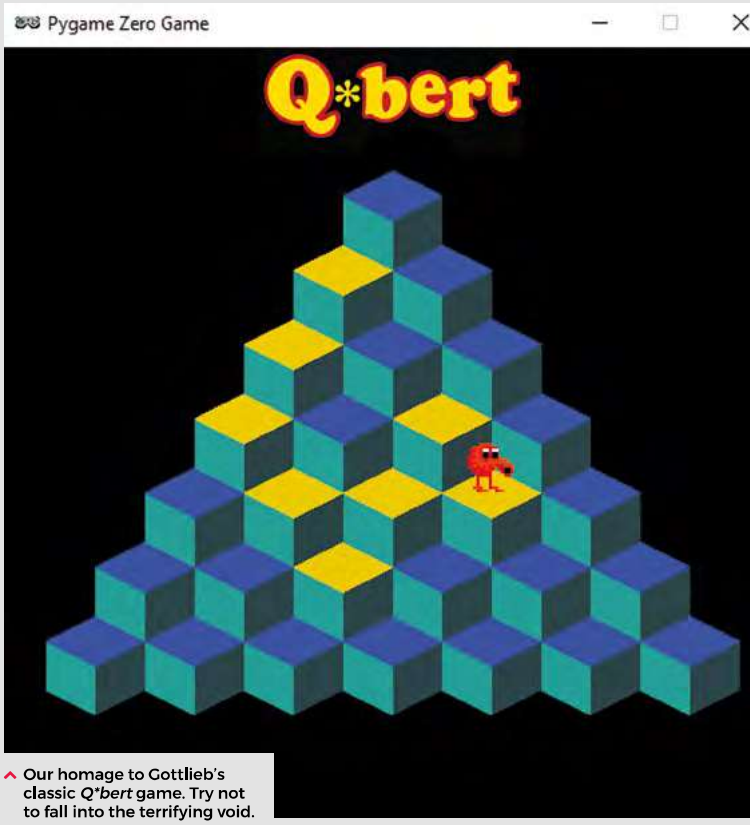
Bouncing between cubes in Python

Here's Mark's code for a *Q*bert*-style, cube-hopping platform game. 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/wfmag42**

```
# Q*bert

WIDTH = HEIGHT = 500

gameState = 0
blocks = []
qbert = Actor('qbert2', center=(250, 80))
qbert.movex = qbert.movey = qbert.frame = count = 0;
bounce = [-6,-4,-2,-1,0,0,0,0,0,0,0,1,2,4,6]

for r in range(0, 7):
    for b in range(0, 7-r):
        blocks.append(Actor('block0', center=(60+(b*64)+(r*32),
400-(r*48))))

def draw():
    screen.blit("background", (0, 0))
    for b in range(0, 28): blocks[b].draw()
    if gameState == 0 or (gameState == 1 and count%4 == 0):
qbert.draw()
    if gameState == 2 : screen.draw.text("YOU CLEARED THE
LEVEL!", center = (250, 250), owidth=0.5, ocolor=(255,255,255),
color=(255,0,255) , fontsize=40)

def update():
    global gameState, count
    if gameState == 0:
        if qbert.movex == 0 and qbert.movey == 0 :
            if keyboard.left: jump(32,48,3)
            if keyboard.right: jump(-32,-48,1)
            if keyboard.up: jump(-32,48,0)
            if keyboard.down: jump(32,-48,2)
        if qbert.movex != 0 : move()
    count += 1;

def move():
    if qbert.movex > 0 :
        qbert.x -=2
        qbert.movex -=2
    if qbert.movex < 0 :
        qbert.x +=2
        qbert.movex +=2
    if qbert.movey > 0 :
        qbert.y -=3 - bounce[qbert.frame]
        qbert.movey -=3
    if qbert.movey < 0 :
        qbert.y +=3 + bounce[qbert.frame]
        qbert.movey +=3
    qbert.frame +=1
    if qbert.movex == 0 :
        checkBlock()
```

```
def checkBlock():
    global gameState
    block = -1
    curBlock = 0
    numSelected = 0
    for r in range(0, 7):
        for b in range(0, 7-r):
            x = 60+(b*64)+(r*32) -2
            y = 400-(r*48) -32
            if qbert.x == x and qbert.y == y :
                block = curBlock
                blocks[block].image = "block1"
            curBlock +=1
    if block == -1 : gameState = 1
    for b in range(0, 28):
        if blocks[b].image == "block1" : numSelected += 1
    if numSelected == 28 : gameState = 2

def jump(x,y,d):
    qbert.movex = x
    qbert.movey = y
    qbert.image = "qbert"+str(d)
    qbert.frame = 0
```



Our homage to Gottlieb's classic *Q*bert* game. Try not to fall into the terrifying void.