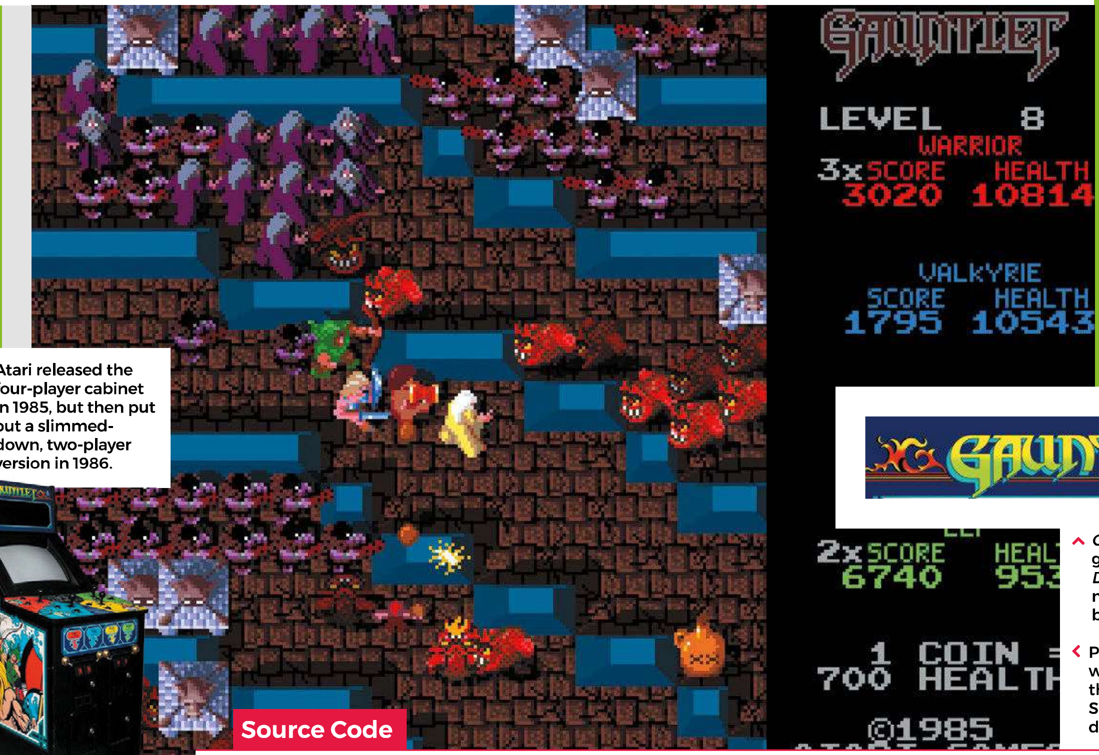
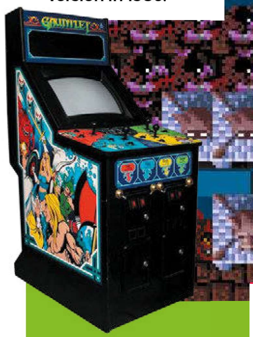




Atari released the four-player cabinet in 1985, but then put out a slimmed-down, two-player version in 1986.

Gauntlet was originally going to be called *Dungeon*, but that name was already taken by the time of release.

Players collected items while battling their way through dungeons. Shooting food was a definite faux pas.

Source Code

Code Gauntlet's four-player co-op

AUTHOR
MARK VANSTONE

Four players dungeon crawling at once? Mark shows you how

Atari's *Gauntlet* was an eye-catching game, not least because it allowed four people to explore its dungeons together. Each player could choose one of four characters, each with its own abilities – there was a warrior, a Valkyrie, a wizard, and an elf – and surviving each dungeon required slaughtering enemies and the constant gathering of food, potions, and keys that unlocked doors and exits.

Designed by Ed Logg, and loosely based on the tabletop RPG Dungeons & Dragons, as well as John Palevich's 1983 dungeon crawler, *Dandy*, *Gauntlet* was a big success. It was ported to most of the popular home systems at the time, and Atari released a sequel arcade machine, *Gauntlet II*, in 1986.

Atari's original arcade machine featured four joysticks, but our example will mix keyboard controls and gamepad inputs. Before we deal with the movement, we'll need some characters and dungeon graphics. For this example, we can make our dungeon from a large bitmap image and use a collision map to prevent our characters from clipping through walls. We'll also need graphics for the characters moving in eight different directions. Each direction has three frames of walking animation, which makes a total of 24 frames per character. We can use a Pygame Zero Actor object for each character and add a few extra properties to keep track of direction and the current animation frame. If we put the character Actors in a list, we can loop through the list to check for collisions, move the player, or draw them to the screen.

We now test input devices for movement controls using the built-in Pygame keyboard object to test if keys are pressed. For example, `keyboard.left` will return `True` if the left arrow key is being held down. We can use the arrow keys for one player and the **WASD** keys for the other keyboard player. If we register x and y movements separately, then if two keys are pressed – for example, up and left – we can read that as a diagonal movement. In this way, we can get all eight directions of movement from just four keys.

For joystick or gamepad movement, we need to import the joystick module from Pygame. This provides us with methods to count the number of joystick or gamepad devices that are attached to the computer, and then initialise them for input. When we check for input from these devices, we just need to get the x-axis value and the y- axis value and then make it into an integer. Joysticks and gamepads should return a number between -1 and 1 on each axis, so if we round that number, we will get the movement value we need.

We can work out the direction (and the image we need to use) of the character with a small lookup table of x and y values and translate that to a frame number cycling through those three frames of animation as the character walks. Then all we need to do before we move the character is check they aren't going to collide with a wall or another character. And that's it – we now have a four-player control system. As for adding enemy spawners, loot, and keys – well, that's a subject for another time.

Four-player movement in Python

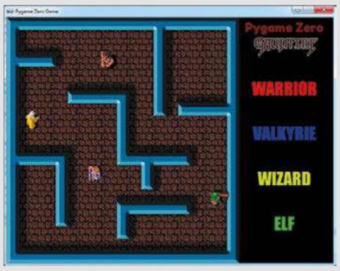
Here's Mark's code for a *Gauntlet*-style four-player mechanic. To get it running on your system, you'll need to install Pygame Zero – full instructions are available at **wfmag.cc/pgzero**.

Download the code from GitHub: **wfmag.cc/wfmag39**

```
import math
from pygame import image, Color, joystick

myChars = []
myDirs = [(0,1),(-1,1),(-1,0),(-1,-1),(0,-1),(1,-
1),(1,0),(1,1)]
collisionmap = image.load(‘images/collisionmap.png’)
joystick.init()
joyin0 = joyin1 = False
if(joystick.get_count() > 0):
    joyin0 = joystick.Joystick(0)
    joyin0.init()
if(joystick.get_count() > 1):
    joyin1 = joystick.Joystick(1)
    joyin1.init()

def makeChar(name,x,y):
    c = len(myChars)
    myChars.append(Actor(name+”_1”,(x, y)))
    myChars[c].name = name
    myChars[c].frame = myChars[c].movex = myChars[c].movey =
myChars[c].dir = 0

def draw():
    screen.blit(“colourmap”,(0,0))
    drawChars()

def drawChars():
    for c in range(len(myChars)):
        myChars[c].image = myChars[c].
name+”_”+str(((myChars[c].dir*3)+1)+math.floor(myChars[c].
frame/10))
        myChars[c].draw()

def update():
    checkInput()
    moveChars()

def checkInput():
    if keyboard.left: myChars[0].movex = -1
    if keyboard.right: myChars[0].movex = 1
    if keyboard.up: myChars[0].movey = -1
    if keyboard.down: myChars[0].movey = 1
    if keyboard.a: myChars[1].movex = -1
    if keyboard.d: myChars[1].movex = 1
    if keyboard.w: myChars[1].movey = -1
    if keyboard.s: myChars[1].movey = 1
    if joyin0:
        myChars[2].movex = round(joyin0.get_axis(0))
        myChars[2].movey = round(joyin0.get_axis(1))
    if joyin1:
        myChars[3].movex = round(joyin1.get_axis(0))
        myChars[3].movey = round(joyin1.get_axis(1))

def moveChars():
    for c in range(len(myChars)):
        getCharDir(myChars[c])
        if myChars[c].movex or myChars[c].movey:
            myChars[c].frame += 1
            if myChars[c].frame >= 30: myChars[c].frame = 0
            testmove = (int(myChars[c].x + (myChars[c].movex
*20)),int(myChars[c].y + (myChars[c].movey *20)))
            if collisionmap.get_at(testmove) == Color(‘black’)
and collideChars(c,testmove) == False:
                myChars[c].x += myChars[c].movex
                myChars[c].y += myChars[c].movey
            myChars[c].movex = 0
            myChars[c].movey = 0

def getCharDir(ch):
    for d in range(len(myDirs)):
        if myDirs[d] == (ch.movex,ch.movey):
            ch.dir = d

def collideChars(c,xy):
    for ch in range(len(myChars)):
         if myChars[ch].collidepoint(xy) and ch != c:
             return True
    return False

makeChar(“warrior”,60,60)
makeChar(“valkyrie”,500,450)
makeChar(“wizard”,460,180)
makeChar(“elf”,100,400)
```

Finding sprites

If you want to reconstruct a retro game like *Gauntlet*, you can often find sprite sheets online – these are bitmaps with all the frames of animation for a character on one sheet. A good source of these files is **spriters-resource.com**, where you'll find sprite sheets for a wide range of retro games. Some coding systems can use the sprite-sheets as they are, but for this example, we have cut them up into separate frames. You can do this with Sprite Sheet Slicer, available at **wfmag.cc/slicer**.



› Our four-player homage to the classic *Gauntlet* arcade game.