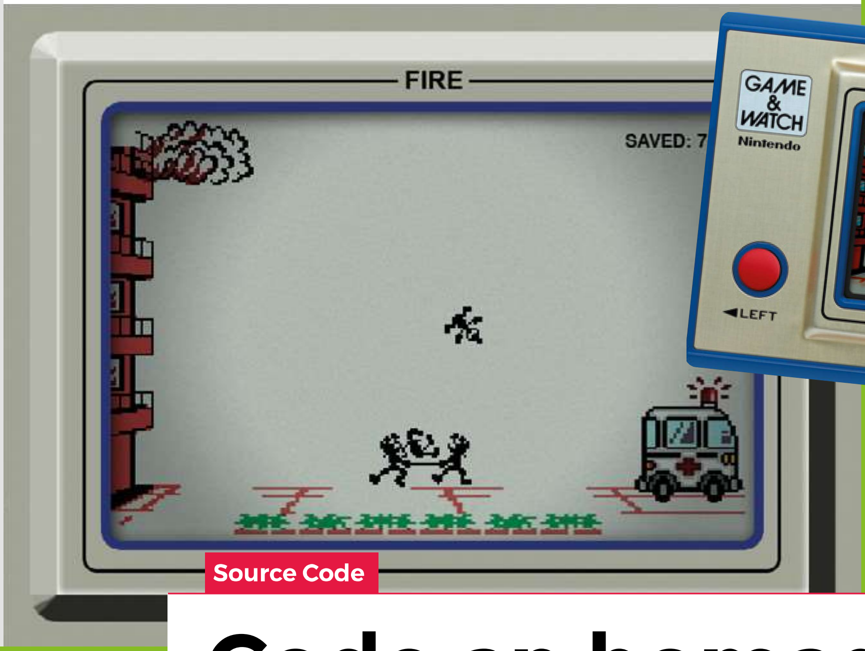

There were predefined places that graphics could be shown on the screen because of the LCD technology.

Our homage to the classic *Fire* Game & Watch handheld.

Source Code

Code an homage to a Game & Watch classic

AUTHOR
MARK VANSTONE

Revisit Nintendo's early gaming years with our homage to Fire

As the seventies turned into the eighties, Nintendo began a line of handheld games: the Game & Watch. Created by Gunpei Yokoi, these featured a liquid crystal screen and simple controls – often just left and right buttons. One such game was *Fire*, where the player had to control a pair of firefighters as they attempted to save people jumping out of a burning building. Although the moving graphics were monochrome, there was a colour overlay showing the burning building, the ground, and the ambulance.

These LCD game graphics weren't drawn with a matrix of pixels like a computer game, but had a defined set of areas on the screen where shapes could appear. This limited the variations of possible graphics in one game, and due to the speed of the electronics behind the handheld, the refresh rate was considerably slower than most computer games: only around two frames per second.

For this example, we're going to recreate the look and feel of the original *Fire* with Pygame Zero. As we have in previous editions of Source Code, we've downloaded our graphics from **spriters-resource.com**.

The first thing we need to do is sort out that frame rate. We want the `draw()` function to draw every update, but we only want the game objects to move every 30 updates, so what we do is have a `count` variable and only fire the `doUpdate()` function if `count%30` is zero. The `%` sign returns the remainder when we divide by 30, so we'll get numbers between 0 and 29 in this case.

We draw the background image first, and then we need to have our two firefighters move between three positions along the ground. We can catch the left and right arrow key presses, but we only want to have one movement per refresh, so we set a `movement` variable to -1 or 1 and then do the movement when the next refresh happens. We actually have three Actors for the firefighters, one for each position, and only draw the one at the current position.

Now for the people jumping out of the building. There are only 22 positions which a jumping person can be in, so we can put them in a list of tuples which represent the x and y co-ordinates of the positions on the screen. We also have a different image for each position on the screen, so as our person moves, their image is changed to reflect the position on the screen. We start the game by making a new jumping person in the form of an actor and add that to a list of jumpers. Then, each update we move all the jumpers in the list along by one frame and change their image. When we get to the `draw()` function, we draw all our jumpers at the co-ordinates for their frame.

So now we have a game that we can control the firefighters left and right, and have people jumping out of the building following a predefined set of positions on the screen. We need to detect if the firefighters are stopping the people from hitting the ground, so we test to see if a jumper is at any of the frames where they need to be caught, and if so, are the firefighters in the right position to catch them? If not, the person will fall to the ground and the game's over. If the firefighters are in the right position to save the person all the way across the screen, however, they'll bounce into the ambulance and the player's score increases.

With that, the game's pretty much complete. The original *Fire* gave you three lives and two difficulty levels, but as always, we'll leave you to add those features in for yourself.

Python & Watch

Here's Mark's code for a *Fire*-style action game in Python. To get it running on your system, you'll first need to install Pygame Zero. You can find full instructions at **wfmag.cc/pgzero**.

Download the code from GitHub: **wfmag.cc/wfmag63**

```
# Fire
import pgzrun
count = catcherPos = moveCatcher = gameState = score = 0
catchers = []
jumpers = []
jumperPositions = [(130,220),(190,260),(210,320),(220,360),(240,410),
(260,360),
                   (270,320),(290,250),(320,220),(340,250),(360,300),
(380,360),
                   (390,410),(420,360),(430,300),(470,250),(500,300),
(520,360),
                   (538,410),(580,360),(600,320),(620,350)]
for c in range(3):
    catchers.append(Actor('catcher'+str(c), center=(240+(c*150), 425)))

def draw():
    screen.blit("background",(0,0))
    for c in range(3):
        if catcherPos == c: catchers[c].draw()
    for j in jumpers:
        if j.state == 0: j.draw()
        if j.state == -1 and count%2 == 0: j.draw()
    screen.draw.text("SAVED: "+str(score), topleft = (580, 120),
color=(0,0,0) , fontsize=25)

def update():
    global count
    count += 1
    if(count%30 == 0) : doUpdate()
    if(count%2000 == 0) : makeJumper()

def doUpdate():
    global catcherPos, moveCatcher, gameState, score
    if gameState == 0:
        catcherPos = limit(catcherPos+moveCatcher, 0, 2)
        moveCatcher = 0
        for j in jumpers:
            if (j.frame < 21 and j.state == 0):
                j.frame += 1
                j.image = "jumper"+str(j.frame)
                j.pos = jumperPositions[j.frame]
            else:
                if j.state == 0:
                    j.state = 1
                    score += 1
                    makeJumper()
            if (j.frame == 4 and catcherPos != 0) or (j.frame == 12 and
catcherPos != 1) or (j.frame == 18 and catcherPos != 2):
                j.state = -1
                j.image = "jumperdropped"
                j.y += 50
                gameState = 1

def on_key_down(key):
    global moveCatcher
    if key.name == "LEFT":
        moveCatcher = -1
    if key.name == "RIGHT":
        moveCatcher = 1

def makeJumper():
    if len(jumpers)%5 == 4:
        jumpers.append(Actor('jumper0', center=(130, 270)))
        jumpers[len(jumpers)-1].frame = 1
    else:
        jumpers.append(Actor('jumper0', center=(130, 220)))
        jumpers[len(jumpers)-1].frame = 0
    jumpers[len(jumpers)-1].state = 0

def limit(n, minn, maxn):
    return max(min(maxn, n), minn)

makeJumper()

pgzrun.go()
```

Nintendo started releasing the Game & Watch series in 1980. *Fire* Game & Watch had two versions: the silver version and the wide-screen version.