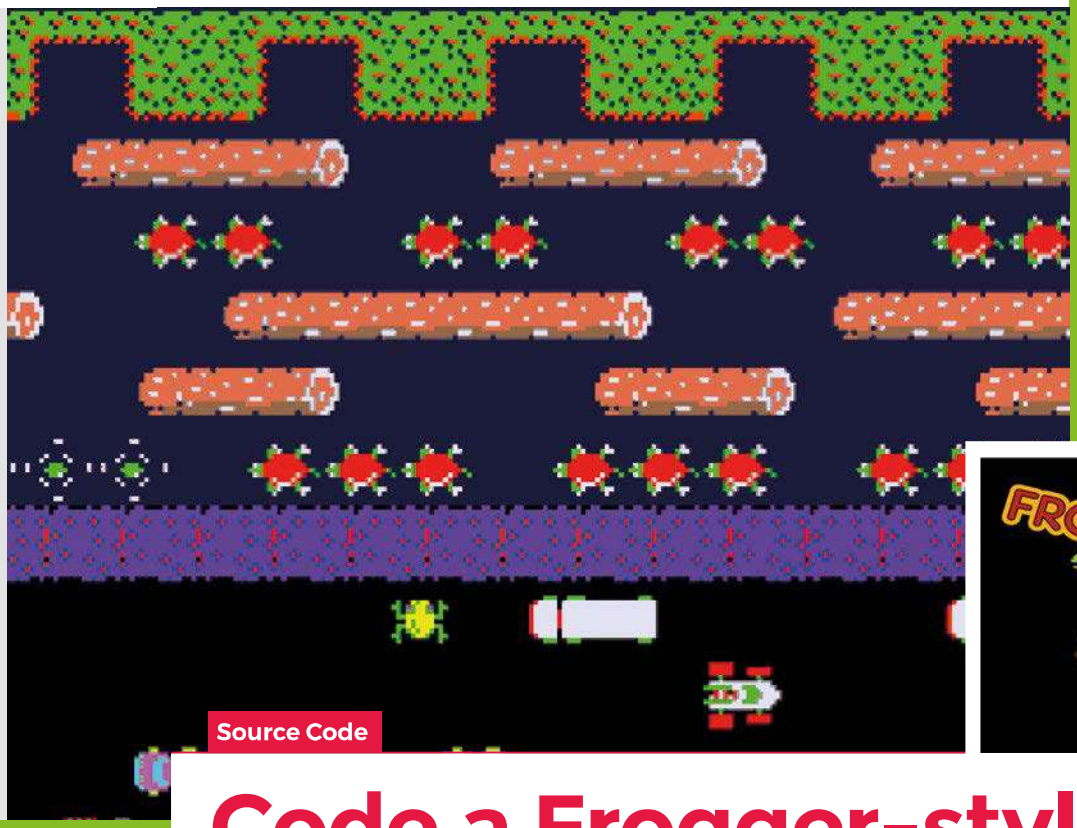


Source Code

Code a Frogger-style road-crossing game

Save the frog from busy roads and rivers with a simple remake of Konami's classic arcade game

AUTHOR
RIK CROSS

Why did the frog cross the road? Because *Frogger* would be a boring game if it didn't. Released in 1981 by Konami, the game appeared in assorted bars, sports halls, and arcades across the world, and became an instant hit. The concept was simple: players used the joystick to move a succession of frogs from the bottom of the screen to the top, avoiding a variety of hazards – cars, lorries, and later, the occasional crocodile. Each frog had to be safely manoeuvred to one of five alcoves within a time limit, while extra points were awarded for eating flies along the way.

Before *Frogger*, Konami mainly focused on churning out clones of other hit arcade games like *Space Invaders* and *Breakout*; *Frogger* was one of its earliest original ideas, and the simplicity of its concept saw it ported to just about every home system available at the time. (Ironically, Konami's game would fall victim to repeated cloning by other developers.) Decades later, developers still take inspiration from it; Hipster Whale's *Crossy Road* turned *Frogger* into an endless running game; earlier this year, Konami returned to the creative well with *Frogger in Toy Town*, released on Apple Arcade.

"We can recreate *Frogger* in just a few lines of Pygame Zero code"

We can recreate much of *Frogger*'s gameplay in just a few lines of Pygame Zero code. The key elements are the frog's movement, which use the arrow keys, vehicles that move across the screen, and floating objects – logs and turtles – moving in opposite directions. Our background graphic will provide the road, river, and grass for our frog to move over. The frog's movement will be triggered from an `on_key_down()` function, and as the frog moves, we switch to a second frame with legs outstretched, reverting back to a sitting position after a short delay. We can use the inbuilt Actor properties to change the image and set the angle of rotation.

For all the other moving elements, we can also use Pygame Zero Actors; we just need to make an array for our cars with different graphics for the various rows, and an array for our floating objects in the same way.

In our `update()` function, we need to move each Actor according to which row it's in, and when an Actor disappears off the screen, set the x coordinate so that it reappears on the opposite side. Handling the logic of the frog moving across the road is quite easy; we just check for collision with each of the

Frogger in Python

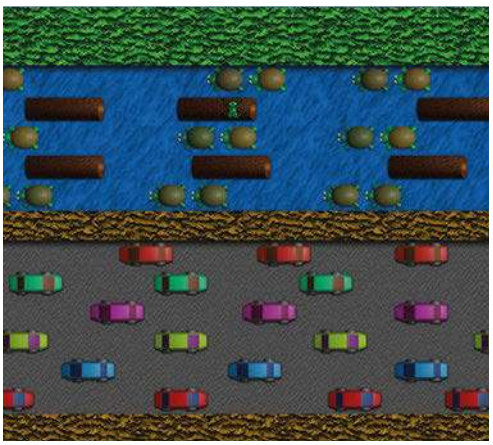
Here's Mark's code snippet, which recreates *Frogger* in Python. To get it running on your system, you'll first need to install Pygame Zero – you can find full instructions at **wfmag.cc/pgzero**

Download the code from GitHub: **wfmag.cc/wfmag27**

```
frog = Actor('frog1', center=(400, 580))
frog.direction = frog.delay = 0
frog.onBoard = -1
cars = []
floats = []
gameState = count = 0
for r in range(0, 6):
    for c in range(0, 4):
        cars.append(Actor('car'+str(r+1),
center=((r*20)+(c*(240-(r*10))), 540-(r*40))))
        if r < 5: floats.append(Actor('float'+str(r+1),
center=((r*20)+(c*(240-(r*10))), 260-(r*40))))

def draw():
    global count
    screen.blit("background", (0, 0))
    for c in range(0, 20):
        floats[c].draw()
    if gameState == 0 or (gameState == 1 and count%2 == 0):
frog.draw()
    for c in range(0, 24):
        cars[c].draw()
    count += 1

def update():
    global gameState
    if gameState == 0:
        frog.onBoard = -1
        for r in range(0, 6):
            s = -1
            if r%2 == 0: s = 1
            for c in range(0, 4):
                i = (r*4)+c
                cars[i].x += s
                if cars[i].x > 840: cars[i].x = -40
                if cars[i].x < -40: cars[i].x = 840
                if cars[i].colliderect(frog): gameState = 1
                if r < 5:
                    floats[i].x -= s
                    if floats[i].x > 880: floats[i].x = -80
                    if floats[i].x < -80: floats[i].x = 880
                    if floats[i].colliderect(frog):
                        frog.onBoard = i
                        frog.x -= s
        if frog.delay > 0:
            frog.delay += 1
            if frog.delay > 10:
                frog.image = "frog1"
                frog.angle = frog.direction
        if frog.y > 60 and frog.y < 270 and frog.onBoard ==
-1: gameState = 1

def on_key_down(key):
    if gameState == 0:
        if key.name == "UP": frogMove(0,-40,0)
        if key.name == "DOWN": frogMove(0,40,180)
        if key.name == "LEFT": frogMove(-40,0,90)
        if key.name == "RIGHT": frogMove(40,0,270)

def frogMove(x,y,d):
    if 800 > frog.x+x > 0: frog.x += x
    if 600 > frog.y+y > 0: frog.y += y
    frog.image = "frog2"
    frog.delay = 1
    frog.angle = frog.direction = d
```

cars, and if the frog hits a car, then we have a squashed frog. The river crossing is a little more complicated. Each time the frog moves on the river, we need to make sure that it's on a floating Actor. We therefore check to make sure that the frog is in collision with one of the floating elements, otherwise it's game over.

There are lots of other elements you could add to the example shown here: the original arcade game provided several frogs to guide to their alcoves on the other side of the river, while crocodiles also popped up from time to time to add a bit more danger. Pygame Zero has all the tools you need to make a fully functional version of Konami's hit.



Amphibious machines

The *Frogger* arcade machine was groundbreaking in that it used two CPUs: a pair of Z80 processors, one to run the main game and one to handle the sound. Konami used the same Z80 CPU in many of its other arcade hits in the early eighties, including *Konami's Ping Pong*, *Time Pilot*, and the surreal pigs-versus-wolves action game, *Pooyan*. Along with *Scramble*, also released in 1981, *Frogger* was by far Konami's most influential game of the period, though, with its ports selling an estimated 20 million units across all systems.