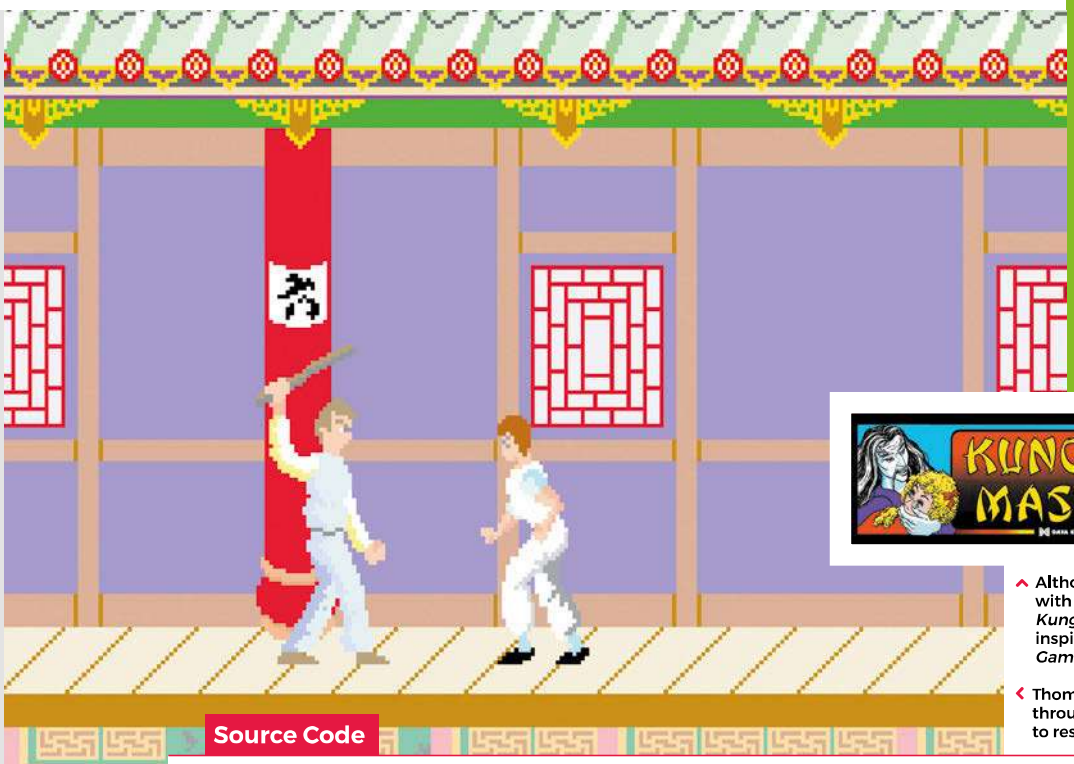

Although released to tie in with *Jackie Chan's Spartan X*, *Kung-Fu Master* was originally inspired by the Bruce Lee film, *Game of Death*.

Thomas battles his way through the martial arts mob to rescue his girlfriend, Sylvia.

Source Code

Code a Kung-Fu Master style beat-'em-up

Punch and kick your way through a rabble of bad dudes in a simple scrolling beat-'em-up

AUTHOR
MARK VANSTONE

Kung-Fu Master hit arcades in 1984. Its side-scrolling action, punching and kicking through an army of knife-throwing goons, helped create the beat-'em-up genre. In fact, its designer, Takashi Nishiyama, would go on to kickstart the *Street Fighter* series at Capcom, and later start up the *Fatal Fury* franchise at SNK.

In true eighties arcade style, *Kung-Fu Master* distils the elements of a chop-socky action film to its essentials. Hero Thomas and his girlfriend are attacked, she's kidnapped, and Thomas fights his way through successive levels of bad guys to rescue her. The screen scrolls from side to side, and Thomas must use his kicks and punches to get from one side of the level to the other and climb the stairs to the next floor of the building.

To recreate this classic with Pygame Zero, we'll need quite a few frames of animation, both for the hero character and the enemies he'll battle. For a reasonable walk cycle, we'll need at least six frames in each direction. Any fewer than six won't look convincing, but more frames can achieve a smoother effect. For this example, I've used the 3D package Poser, since it has a handy walk designer which makes generating sequences of animation much easier.

"For a walk cycle animation, we'll need at least six frames in each direction"

Once we have the animation frames for our characters, including a punch, kick, and any others you want to add, we need a background for the characters to walk along. The image we're using is 2000×400 pixels, and we start the game by displaying the central part so our hero can walk either way. By detecting arrow key presses, the hero can 'walk' one way or the other by moving the background left and right, while cycling through the walk animation frames. Then if we detect a **Q** key press, we change the action string to kick; if it's **A**, it's punch. Then in our `update()` function, we use that action to set the Actor's image to the indicated action frame.

Our enemy Actors will constantly walk towards the centre of the screen, and we can cycle through their walking frames the same way we do with the main hero. To give kicks and punches an effect, we put in collision checks. If the hero strikes while an enemy collides with him, we register a hit. This could be made more precise to require more skill, but once a strike's registered, we can switch the enemy to a different status that will cause them to fall downwards and off the screen.

This sample is a starting point to demonstrate the basics of the beat-'em-up genre. With the addition of flying daggers, several levels, and a variety of bad guys, you'll be on your way to creating a Pygame Zero version of this classic game.

A basic beat-'em-up basis

Here's Mark's code to make a classic 1980s-style scrolling beat-'em-up in Python. To get up and running you'll need to install Pygame Zero – full instructions are available at **wfmag.cc/pgzero**

Download the code from GitHub: **wfmag.cc/wfmag32**

```
# Kung-Fu Master
import random

HEIGHT = 450
gameState = count = 0
bloke = Actor('walkl_0001', center=(400, 250))
blokeDir = "l"
backPos = -500
dudes = []
action = ""
actioncount = 0

def draw():
    screen.fill((0,0,0))
    screen.blit("background", (backPos, 30))
    screen.draw.text("Pygame Zero Kung-Fu Master", center =
(400, 15), owidth=1, ocolor=(255,0,0), color=(255,255,0) ,
fontsize=30)
    if gameState != 1 or (gameState == 1 and count%2 ==
0):bloke.draw()
    for d in dudes:
        d.draw()

def on_key_down(key):
    global action, actioncount
    actioncount = 10
    if gameState == 0:
        if key.name == "A": action = "punch"
        if key.name == "Q": action = "kick"

def update():
    global count, backPos, blokeDir, action, actioncount
    if gameState == 0:
        bloke.image = 'stand' + blokeDir
        if action == "punch": bloke.image = 'punch'+blokeDir
        if action == "kick": bloke.image = 'kick'+blokeDir
        if actioncount <= 0: action = ""
        if keyboard.left: moveBloke(3,"l")
        elif keyboard.right: moveBloke(-3,"r")
        if random.randint(0, 100) == 0: makeDude()
        updateDudes()
    count += 1
    actioncount -= 1

def moveBloke(x,d):
    global backPos, blokeDir
    frame = int((count%48)/8) + 1
    if backPos + x < -3 and backPos + x > -1197:
        backPos += x
        moveDudes(x)
        bloke.image = 'walk'+d+'_000'+str(frame)
        blokeDir = d

def makeDude():
    d = len(dudes)
    if random.randint(0, 1) == 0:
        dudes.append(Actor('duder_0001', center=(-50,250)))
    else:
        dudes.append(Actor('dudel_0001', center=(850, 250)))
    dudes[d].status = 0

def updateDudes():
    global gameState
    frame = int((count%48)/8) + 1
    for d in dudes:
        if (bloke.image == 'punch'+blokeDir or bloke.image ==
'kick'+blokeDir) and bloke.collidepoint((d.x, d.y)):
            d.status += 1
        if d.x <=400:
            if d.status > 10:
                d.image = 'dudefallr'
                d.y += 5
            else:
                d.x += 2
                d.image = 'duder_000'+str(frame)
        if d.x >400:
            if d.status > 10:
                d.image = 'dudefalll'
                d.y += 5
            else:
                d.x -= 2
                d.image = 'dudel_000'+str(frame)
        if d.x > 398 and d.x < 402 and d.status == 0:
gameState = 1

def moveDudes(x):
    for d in dudes:
        d.x += x
```

Our *Kung-Fu Master* homage features punches, kicks, and a host of goons to use them on.

THE GENERATION GAME

Because we're moving the background when our hero walks left and right, we need to make sure we move our enemies with the background, otherwise they'll look as though they're sliding in mid-air – this also applies to any other objects that aren't part of the background.

The number of enemies can be governed in several ways: in our code, we just have a random number deciding if a new enemy will appear during each update, but we could use a predefined pattern for the enemy generation to make it a bit less random, or we use a combination of patterns and random numbers.