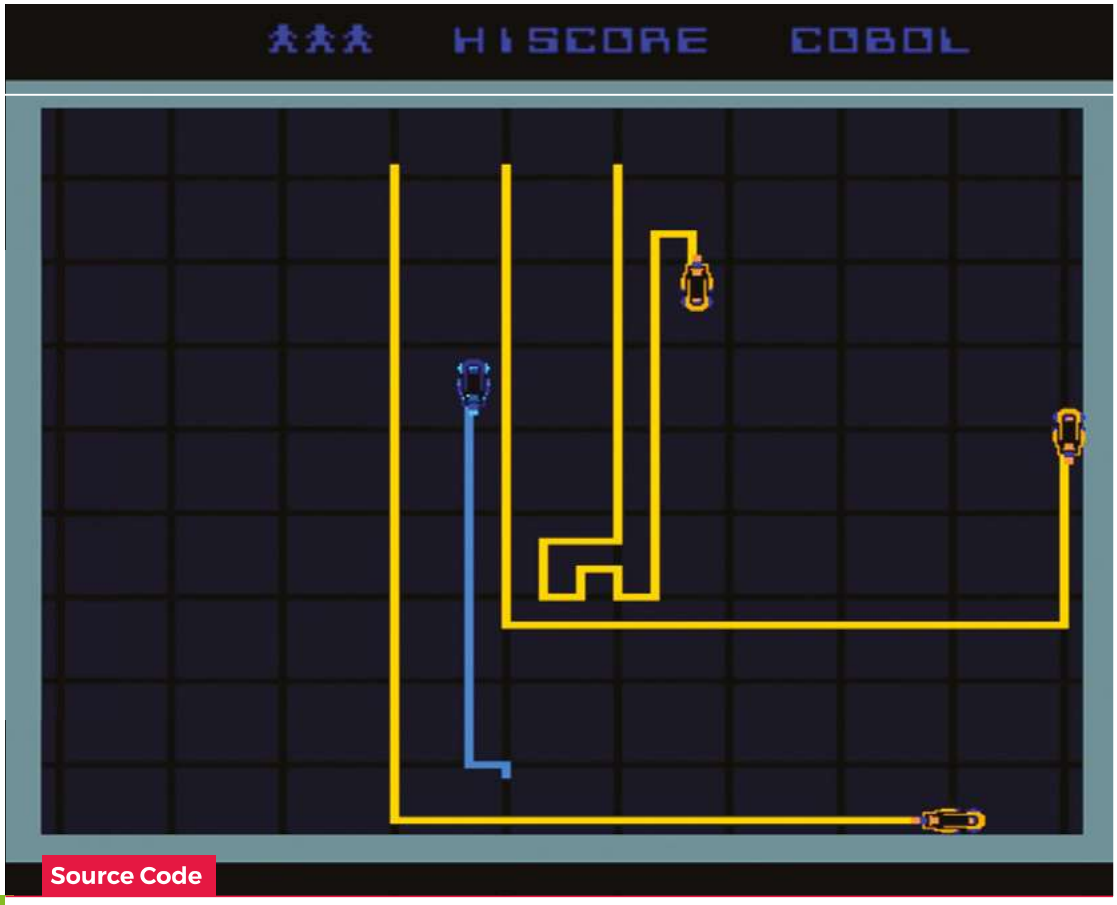
The *TRON* cab had two game controllers: a rotary wheel and a joystick.

Battle against AI enemies in the original arcade classic.



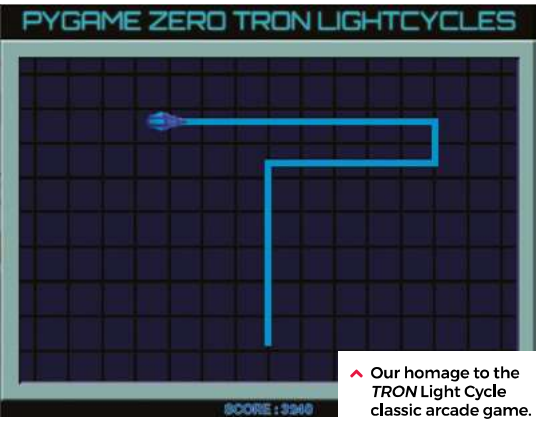
Source Code

Code a Light Cycle arcade minigame

Speed around an arena, avoiding walls and deadly trails

AUTHOR
MARK VANSTONE

At the beginning of the 1980s, Disney made plans for an entirely new kind of animated movie that used cutting-edge computer graphics. The resulting film was 1982's *TRON*, and it inevitably sparked one of the earliest tie-in arcade machines. The game featured several minigames, including one based on the Light Cycle section of the movie, where players speed around an arena on high-tech motorbikes, which leave a deadly trail of light in their wake. If competitors hit any walls or cross the path of any trails, then it's game over. Players progress through the twelve levels which were all named after programming languages. In the Light Cycle game, the players compete against AI players who drive yellow Light Cycles around the arena. As the levels progress, more AI Players are added.

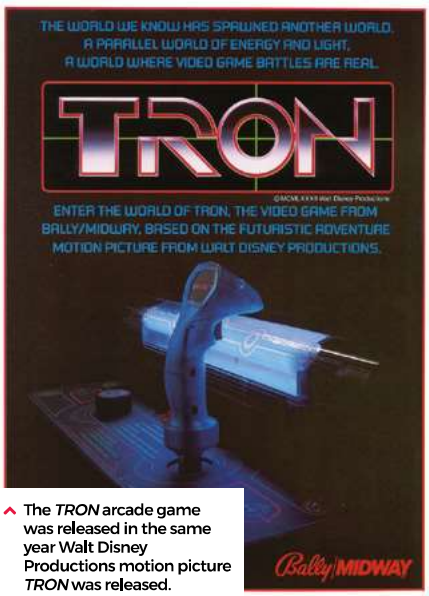


Our homage to the *TRON* Light Cycle classic arcade game.

The *TRON* game, distributed by Bally Midway, was well-received in arcades, and even won Electronic Games Magazine's (presumably) coveted Coin-operated Game of the Year gong. Although the arcade game wasn't ported to home computers at the time, several similar games – and outright clones – emerged, such as the unsubtly named *Light Cycle* for the BBC Micro, Oric, and ZX Spectrum.

The *Light Cycle* minigame is essentially a variation on *Snake*, with the player leaving a trail behind them as they move around the screen. There are various ways to code this with Pygame Zero. In this sample, we'll focus on the movement of the player Light Cycle and creating the trails that are left behind as it moves around the screen. We could use line drawing functions for the trail behind the bike, or go for a system like *Snake*, where blocks are added to the trail as the player moves. In this example, though, we're going to use a two-dimensional list as a matrix of positions on the screen. This means that wherever the player moves on the screen, we can set the position as visited or check to see if it's been visited before and, if so, trigger an end-game event.

For the main `draw()` function, we first blit our background image which is the cross-hatched arena, then we iterate through our two-dimensional list of screen positions (each 10 pixels square) displaying a square anywhere the Cycle has been. The Cycle is then drawn and we can add a display of the score. The `update()` function contains code to move the Cycle and check for collisions. We use a list of directions in degrees to control



The *TRON* arcade game was released in the same year Walt Disney Productions motion picture *TRON* was released.

the angle the player is pointing, and another list of x and y increments for each direction. Each update we add x and y coordinates to the Cycle actor to move it in the direction that it's pointing multiplied by our speed variable. We have an **on_key_down()** function defined to handle changing the direction of the Cycle actor with the arrow keys.

We need to wait a while before checking for collisions on the current position, as the Cycle won't have moved away for several updates, so each screen position in the matrix is actually a counter of how many updates it's been there for. We can then test to see if 15 updates have happened before testing the square for collisions, which gives our Cycle enough time to clear the area. If we do detect a collision, then we can start the game-end sequence. We set the **gamestate** variable to 1, which then means the **update()** function uses that variable as a counter to run through the frames of animation for the Cycle's explosion. Once it reaches the end of the sequence, the game stops. We have a key press defined (the **SPACE** bar) in the **on_key_down()** function to call our **init()** function, which will not only set up variables when the game starts but sets things back to their starting state.

So that's the fundamentals of the player Light Cycle movement and collision checking. To make it more like the original arcade game, why not try experimenting with the code and adding a few computer-controlled rivals?

Light Cycles in Python

Here's Mark's code for a Light Cycle minigame straight out of *TRON*. To get it working 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/wfmag47**

```
# TRON

speed = 3
dirs = [0,90,180,270]
moves = [(0,-1),(-1,0),(0,1),(1,0)]

def draw():
    screen.blit("background", (0, 0))
    for x in range(0, 79):
        for y in range(0, 59):
            if matrix[x][y] > 0:
                matrix[x][y] += 1
                screen.blit("dot",((x*10)-5,(y*10)-5))
    bike.draw()
    screen.draw.text("SCORE : "+ str(score), center=(400, 588), owidth=0.5,
ocolor=(0,255,255), color=(0,0,255) , fontsize=28)

def update():
    global matrix,gamestate,score
    if gamestate == 0:
        bike.angle = dirs[bike.direction]
        bike.x += moves[bike.direction][0]*speed
        bike.y += moves[bike.direction][1]*speed
        score += 10
        if matrix[int(bike.x/10)][int(bike.y/10)] < 15 :
            matrix[int(bike.x/10)][int(bike.y/10)] += 1
        else:
            gamestate = 1
        if bike.x < 60 or bike.x > 750 or bike.y < 110 or bike.y > 525:
            gamestate = 1
    else:
        if gamestate < 18:
            bike.image = "bike"+str(int(gamestate/2))
            bike.angle = dirs[bike.direction]
            gamestate += 1

def on_key_down(key):
    if key == keys.LEFT:
        bike.direction += 1
        snapBike()
        if bike.direction == 4 : bike.direction = 0
    if key == keys.RIGHT:
        bike.direction -= 1
        snapBike()
        if bike.direction == -1 : bike.direction = 3
    if key == keys.SPACE and gamestate == 18:
        init()

def snapBike():
    bike.x = int(bike.x/10)*10
    bike.y = int(bike.y/10)*10

def init():
    global bike,matrix,gamestate,score
    bike = Actor('bike1', center=(400, 500))
    bike.direction = 0
    matrix = [[0 for y in range(60)] for x in range(80)]
    gamestate = score = 0

init()
```