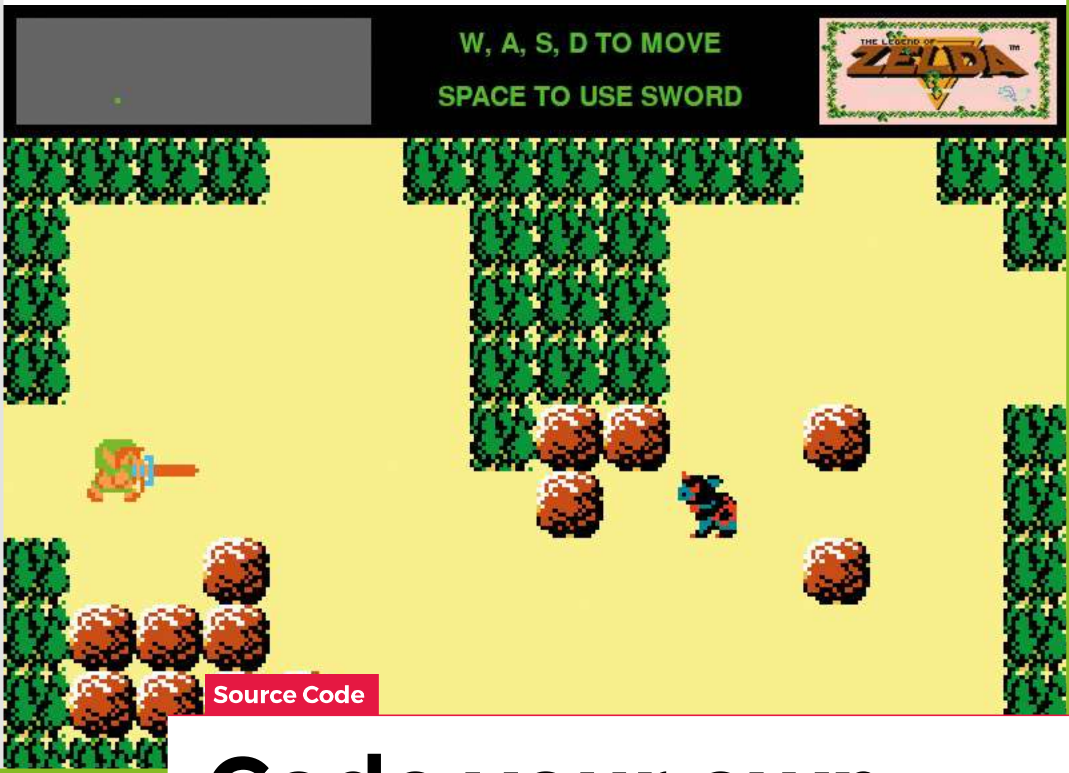


Our homage to the classic *The Legend of Zelda.*

Source Code

Code your own top-down Zelda-like

AUTHOR
MARK VANSTONE

Make your own tribute to the land of Hyrule

With 1986's *The Legend of Zelda*, Nintendo created an adventure with a level of freedom not seen in its earlier games. As plucky hero Link, players were able to head off in any direction they chose, slashing enemies with their sword and uncovering secrets dotted around the fantastical world of Hyrule. The original game's success speaks for itself, with an ongoing string of much-loved sequels and spin-offs, while the series, as a whole, has left a lasting impact on game design.

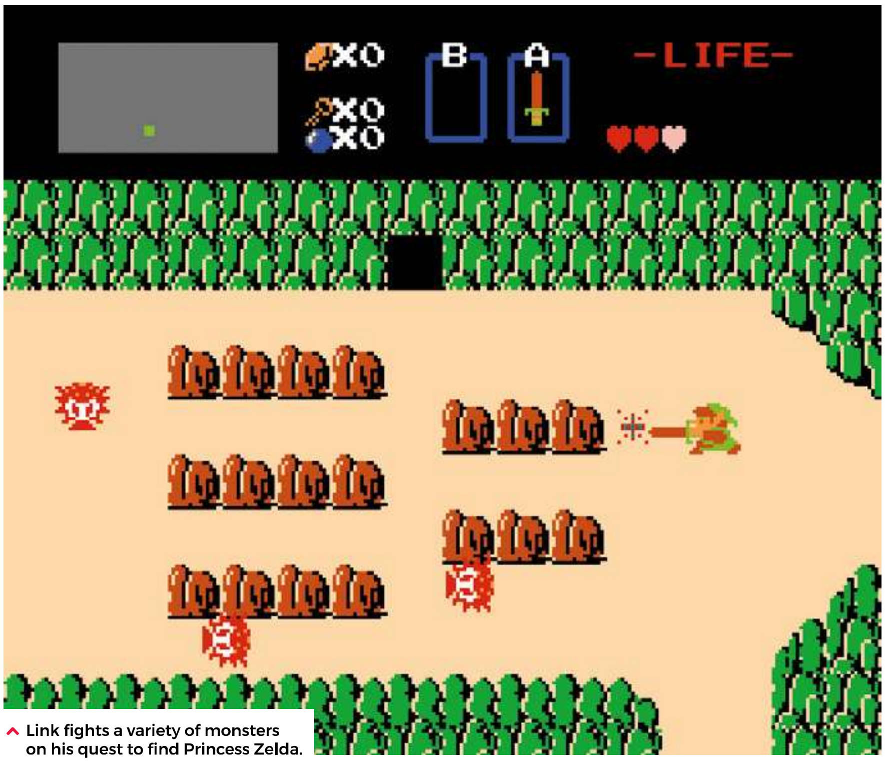
For our Pygame Zero remake, we're going to focus on the original NES version's top-down gameplay. We'll create a map made of blocks that will scroll around as Link explores, and we'll also add some enemies for him to dispatch with his trusty sword.

First, let's generate a *Zelda*-style map. The screen elements are made up of square blocks arranged in a grid. There are squares for trees, rocks, and other types of terrain, including water. For this example, we'll stick to a small selection of block types, but you can add extra ones yourself. If you need to find graphics for these sorts of retro projects, **spriters-resource.com** is worth checking out – the sprites used in our project were created by MisterMike. When designing a map system to represent an area much larger than the screen, we need to store it in a way that isn't just a huge image, so we break it down into blocks. In this case, we'll use blocks that are 50 pixels square, and have our visible map area comprising 16 blocks wide and 10 blocks high.

We could hold all our map data in a large two-dimensional list with each block being represented by a number, but that would mean that even for a fairly small map we'd end up with lines and lines of map data which would be boring to type in. A smarter way of storing this data is to represent the whole map in an image 50 times smaller than the actual map, with each pixel representing one block. Our ground is represented by black pixels, our trees by green pixels, while rocks are yellow and boulders are red. Then we can read through the area of the minimap image currently shown on screen and translate that to our map blocks.

When we display our map blocks, we'll start drawing from 100 pixels down the screen to leave room for a minimap at the top. So all we need to do for our `drawMap()` function is run through an embedded loop of x and y values to translate the pixel colours (which we read using the `image.get_at()` function from the Pygame module) into our larger map blocks and blit them to the screen in the correct positions.

The part of the map we're displaying is controlled by the variables `mapx` and `mapy`. If we want to see a different part of the map then we just need to change those values and that area will be displayed on the next draw cycle. To make the map scroll rather than just switch, we have the



Link fights a variety of monsters on his quest to find Princess Zelda.

The western NES edition of *The Legend of Zelda* came in a gold cartridge.

variables **mapScrollx** and **mapScrolly**. If we set the **mapScrollx** to 16, we can count down with that variable each update cycle and increase the **mapx** value by one each time. This means that in 16 cycles, the viewing area of the map will have moved 16 blocks to the right. We can use the same process for negative scroll values and for the y axis, but in that case, only scrolling 10 blocks at a time.

HERO TIME

Now we're ready to add our hero. Link will be represented by an Actor object which we start with in the first frame of animation and place him somewhere in the middle of the play area. We draw Link each frame in the **draw()** function, but we'll want to add some extra code in a function called **drawChars()**. To get Link to move around the map, we'll define some keys to move in four directions. We could limit Link so he can only move from one block to the next, but this can make the movement look rather jerky, so we're going to allow Link to move a couple of pixels each time we detect that a key is held down. This means we need to work out which block he's moving towards, and if it's not a ground block (represented on our minimap by black pixels), we cancel the move. If Link does move, we need to cycle his animation frames using a frame counter, and then allocate an image based on that frame counter and the direction he's facing.

"We're going to focus on the original NES version's top-down gameplay"

Now we have Link moving around the screen, we need to bring in the scrolling screen function when he reaches the edge. We do this by detecting when he gets to x<0 or x>800 or y>600 or y<100. We can then set our **mapScrollx** or **mapScrolly** variables; this will trigger the scrolling mechanism which will take us to the next section of the map. While we're scrolling, however, we need to make sure other objects on the map also scroll so that they remain in the correct position relative to the map.

Once we have Link moving around from screen to screen, we'll want some enemies for him to fight. We'll make some monsters as Actors and dot them around the map. We want them to move around a bit, so we can add some logic in the update cycle to move them towards Link if they're on the screen at the same time. We can test for block collision the same way we do for Link.

Next, we'll need to add Link's sword-fighting ability. We'll have the sword appear when the **SPACE** bar is pressed by making the sword as a separate Actor. There are four different images for the four directions, although the same effect could be done by changing the angle of the sword Actor. When this animation's running, we can test to see if there's a collision with any of the monsters, and if so, trigger a change of state so they're dispatched. We change the state of the monster from 10 to 9, which we then use to count down during the update while turning the Actor around, and when that state reaches zero, the enemy's no longer displayed.

The last part we'll cover is the minimap in the top left of the screen. This is an indicator of where Link is on the larger map, so all we need to do is translate his coordinates on the large map down to the size of the minimap and plot a square.

And that's it – we have the basics of a top-down *Zelda*-like! There are all kinds of elements you could add – potions and other useful items, more enemies, secret paths, and so on – but as always, we'll leave those for you to dream up.

A mini Hyrule in Python

Here’s Mark’s code for a top-down *Zelda*-like. To get it working on your system, you’ll first need to install Pygame Zero – full instructions can be found at **wfmag.cc/pgzero**.

Download the code from GitHub: **wfmag.cc/wfmag61**

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

link = Actor(“link_1”,center=(400,400))
link.frame = link.movex = link.movey = link.dir = link.testx =
link.testy = 0
sword = Actor(“sword_1”,center=(400,400))
sword.frame = sword.dir = 0
myDirs = [(0,1),(-1,0),(0,-1),(1,0)]
monstersXY = [(1325,375),(1025,-225),(300,-225),(1925,-
225),(1925,375)]
monsters = []
for m in monstersXY:
    monsters.append(Actor(‘monster_1’, center=(m[0], m[1])))
    l = len(monsters)-1
    monsters[l].state = 10
    monsters[l].frame = monsters[l].movex = monsters[l].movey =
monsters[l].dir = monsters[l].testx = monsters[l].testy = 0

mymap = image.load(‘images/map.png’)
mapx = 0
mapy = 10
mapScrollx = 0
mapScrolly = 0

def draw():
    screen.clear()
    screen.blit(“logo”,(612,10))
    screen.draw.text(“W, A, S, D TO MOVE”, center= (440, 30),
color=(0,255,0) , fontsize=30)
    screen.draw.text(“SPACE TO USE SWORD”, center= (440, 70),
color=(0,255,0) , fontsize=30)
    drawMap()
    drawChars()

def drawMap():
    xtest = math.floor(link.x/50 + (link.movex))
    ytest = math.floor((link.y-100)/50 + (link.movey))
    for x in range(16):
        for y in range(10):
            col = mymap.get_at((x+mapx,y+mapy))
            if col == (0,255,0): screen.
blit(“tree”,(x*50,(y*50)+100))
            if col == (0,0,0): screen.
blit(“ground”,(x*50,(y*50)+100))
            if col == (255,0,0): screen.
blit(“boulder”,(x*50,(y*50)+100))
            if col == (255,255,0): screen.
blit(“rock”,(x*50,(y*50)+100))
    maprect = Rect((10, 10), (266, 80))
    screen.draw.filled_rect(maprect, (100, 100, 100))
    mx = (mapx*50)+link.x
    my = (mapy*50)+link.y
    linkrect = Rect(((mx/12)+10, (my/12)), (4, 4))
    screen.draw.filled_rect(linkrect, (0, 255, 0))

def drawChars():
    link.image = “link_”+str(((link.dir*2)+1)+math.floor(link.
frame/10))
    if sword.frame > 0 and sword.dir == 2:
        sword.draw()
    link.draw()
    if sword.frame > 0 and sword.dir != 2:
        sword.draw()
    for m in monsters:
        if onScreen(m.x,m.y) and m.state > 0:
            if m.state < 10:
                m.angle += 10
                m.state -= 1
            if m.state == 10: m.image = “monster_”+str(((m.
dir*2)+1)+math.floor(m.frame/10))
            m.draw()

def update():
    global mapScrollx, mapScrolly,mapx,mapy
    checkInput()
    moveChars()
    if(mapScrollx > 0): mapScroll(1,0)
    if(mapScrollx < 0): mapScroll(-1,0)
    if(mapScrolly > 0): mapScroll(0,1)
    if(mapScrolly < 0): mapScroll(0,-1)
    if sword.frame > 0:
        if(sword.frame > 5):
            sword.x += myDirs[sword.dir][0]*2
            sword.y += myDirs[sword.dir][1]*2
        else:
            sword.x -= myDirs[sword.dir][0]*2
            sword.y -= myDirs[sword.dir][1]*2
        sword.frame -= 1
        for m in monsters:
            if m.collidepoint((sword.x, sword.y)):
                m.state = 9
```

```
def mapScroll(x,y):
    global mapScrollx, mapScrolly,mapx,mapy
    mapx += x
    mapScrollx -= x
    link.x -= x*50
    mapy += y
    mapScrolly -= y
    link.y -= y*50
    for m in monsters:
        m.x -= x*50
        m.y -= y*50

def checkInput():
    if keyboard.a: link.movex = -1
    if keyboard.d: link.movex = 1
    if keyboard.w: link.movey = -1
    if keyboard.s: link.movey = 1

def on_key_down(key):
    if key.name == “SPACE”:
        sword.frame = 10
        sword.dir = link.dir
        sword.image = “sword_”+str(sword.dir)
        sword.x = link.x + (myDirs[sword.dir][0]*30)
        sword.y = link.y + (myDirs[sword.dir][1]*30)

def moveChars():
    global mapScrollx,mapScrolly,mapx,mapy
    getCharDir(link)
    if link.movex or link.movey:
        link.frame += 1
        if link.frame >= 20: link.frame = 0
        if link.movex == 1:
            link.testx = round((link.x-48)/50 + (link.movex))
        else:
            link.testx = round((link.x)/50 + (link.movex))
        if link.movey == 1:
            link.testy = round((link.y-148)/50 + (link.movey))
        else:
            link.testy = round((link.y-100)/50 + (link.movey))
        testmove = (link.testx+mapx,link.testy+mapy)
        if mymap.get_at(testmove) == Color(‘black’):
            link.x += link.movex*2
            link.y += link.movey*2
        link.movex = 0
        link.movey = 0
        if link.x > 800 and mapScrollx == 0:
            mapScrollx = 16
        if link.x < 0 and mapScrollx == 0:
            mapScrollx = -16
        if link.y > 600 and mapScrolly == 0:
            mapScrolly = 10
        if link.y < 100 and mapScrolly == 0:
            mapScrolly = -10
    for m in monsters:
        if onScreen(m.x,m.y) and m.state == 10:
            if (m.x > link.x+50):
                m.movex = -1
                m.testx = round((m.x)/50 + (m.movex))
            else:
                if (m.x < link.x-50):
                    m.movex = 1
                    m.testx = round((m.x-48)/50 + (m.movex))
            if (m.y > link.y+50):
                m.movey = -1
                m.testy = round((m.y-100)/50 + (m.movey))
            else:
                if (m.y < link.y-50):
                    m.movey = 1
                    m.testy = round((m.y-148)/50 + (m.movey))
            getCharDir(m)
            if m.movex or m.movey:
                m.frame += 1
                if m.frame >= 20: m.frame = 0
                testmove = (m.testx+mapx,m.testy+mapy)
                if mymap.get_at(testmove) == Color(‘black’):
                    m.x += m.movex*2
                    m.y += m.movey*2
                m.movex = 0
                m.movey = 0

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

def onScreen(x,y):
    if(x>0 and x<800 and y>100 and y<800): return True
    return False

pgzrun.go()
```