无限迷宫

发表于 2021-01-04 分类于 Challenge， 2019， UNCTF， Misc
Challenge | 2019 | UNCTF | Misc | 无限迷宫

点击此处获得更好的阅读体验

题目考点

- *opencv处理图片，过滤颜色，查找轮廓，直线检测等知识的运用*
- *graph的构造，寻路方法的算法*
- *使用python处理zip文件*

解题过程

打开下载的图片是一个迷宫，如图1。

图片比较小，但是文件很大，使用010 editor打开下载的图片，发现文件后面有很长的附加数据，如图2. 看文件开头为PK,可能是zip文件。

于是使用7-zip打开图片文件,可以看到是加了密的zip文件，里面有个flag.jpg，如图3。

根据题目的提示:上下左右，1234。猜测迷宫的路径可能就是zip的密码，每一步所走的方向,即上下左右对应1234.
因为迷宫为图片，手工走迷宫太累，使用图像处理的方法解决问题。使用图像处理的方法走迷宫需要下面几个步骤：

1. *识别出开始和目标位置*
2. *识别出迷宫的网格，才能确定走的每一个格子*
3. *根据识别出的网格，转换迷宫图片为graph。*
4. *使用寻路方法，寻找开始位置的格子到目标位置格子的最短路径。*
5. *把找到的路径转换为每一步要走的方向*
6. *转换方向为对应的1234，获得zip文件的密码*

转换为代码如下:

```
#!/usr/bin/env python3
# coding=utf-8
# 安装必备工具和库
# apt-get install unzip
# pip3 install numpy
# pip3 install opencv-python
from os.path import isfile, join
from os import listdir
import os
import shutil
import subprocess
from collections import Counter
import math
import cv2 as cv
import numpy as np
import logging
def find_color_max_rect(img, lower, upper):
    ''' 查找lower-upper指定的颜色区域最大的轮廓,
    lower, upper为hsv颜色空间'''
    hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
    # 过滤出红色，(指示起点的图片)
    binary = cv.inRange(hsv, lower, upper)
    # 闭运算，消除起始图片中的空洞
    kernel = np.ones((20, 20), np.uint8)
    closing = cv.morphologyEx(binary, cv.MORPH_CLOSE, kernel)
    # 查找起始图片的轮廓
    contours, _ = cv.findContours(
        closing, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    logging.info("find start contours:%d" % len(contours))
    # 返回面积最大的轮廓
    max_area = 0
    for c in contours:
        c_area = cv.contourArea(c)
        if c_area > max_area:
            max_area = c_area
```

```
            max_c = c
    return cv.boundingRect(max_c)
def find_start(img):
    ''' 查找开始位置--迷宫开始图片的矩形'''
    lower_red = np.array([0, 0, 100])
    upper_red = np.array([15, 255, 200])
    return find_color_max_rect(img, lower_red, upper_red)
def find_end(img):
    ''' 查找结束位置--迷宫目标图片的矩形'''
    lower_yellow = np.array([20, 0, 100])
    upper_yellow = np.array([30, 250, 250])
    return find_color_max_rect(img, lower_yellow, upper_yellow)
def show_rects(img, rects):
    "显示矩形区域"
    ret = img.copy()
    for [x, y, w, h] in rects:
        cv.rectangle(ret, (x, y), (x+w, y+h), (0, 0, 255), 2)
    cv.imshow('rects', ret)
    cv.imwrite('show.jpg', ret)
    cv.waitKey(0)
def uniq_lines(lines, precision=5):
    '''按照precision指定的误差统一直线'''
    sort_lines = lines.copy()
    sort_lines.sort()
    uniq_sort_lines = list(set(sort_lines))
    uniq_sort_lines.sort()
    prev = uniq_sort_lines[0]
    result = [prev]
    for p in uniq_sort_lines[1:]:
        diff = abs(p - prev)
        if diff > precision:
            result.append(p)
        else:
            # 在误差范围内，纠正上一个值，保存为两条线的中间值
            mp = min(p, prev)
            result[-1] = (mp + int(diff/2))
        prev = p
    return result
def find_lines(img, min_length=50):
    "查找线条，返回[horz_lines, vert_lines]"
    src = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    src = cv.GaussianBlur(src, (5, 5), 0)
    edges = cv.Canny(src, 50, 150, None, 3)
    # 霍夫变换检测直线
    lines = cv.HoughLinesP(edges, 1, np.pi / 180, 50, None, min_length, 10)
    # 把误差较小的直线合并
    horz_lines = []
    vert_lines = []
    for ls in lines:
        x1, y1, x2, y2 = ls[0]
        if y1 == y2:
            horz_lines.append(y1)
        elif x1 == x2:
            vert_lines.append(x1)
    horz_lines = uniq_lines(horz_lines)
    vert_lines = uniq_lines(vert_lines)
    return [horz_lines, vert_lines]
def clear_rect(img, rect):
    "清除img中rect指定的区域图像"
    x, y, w, h = rect
    img[y:y+h, x:x+w] = 255
    return img
def best_grid_size(grids):
    "返回最合适的grid大小"
    items = grids[0]
    diffs = [x-y for x, y in zip(items[1:], items[:-1])]
    items2 = grids[1]
    diffs2 = [x-y for x, y in zip(items2[1:], items2[:-1])]
    c = Counter(diffs+diffs2)
    return c.most_common(1)[0][0]
def make_grid_pos(length, grid_size):
    '''根据网格大小生成网格线位置'''
    return [i*grid_size for i in range(int(length/grid_size)+1)]
def find_grid_lines(img, start_rect, end_rect, min_length=50):
    "查找图片的网格线"
    img2 = img.copy()
    # 清理掉开始和结束的图片,提高精确度
    img2 = clear_rect(img2, start_rect)
    img2 = clear_rect(img2, end_rect)
    grids = find_lines(img2, min_length)
    # 使用查找到的线条重新生成网格线，防止漏掉某些线
    grid_size = best_grid_size(grids)
    y, x, _ = img.shape
    hls = make_grid_pos(y, grid_size)
    vls = make_grid_pos(x, grid_size)
    return [hls, vls]
def show_grid(img, horz_lines, vert_lines):
    '''显示网格线'''
```

```
    ret = img.copy()
    for y in horz_lines:
        cv.line(ret, (0, y), (10000, y), (255, 0, 0), 2)
    for x in vert_lines:
        cv.line(ret, (x, 0), (x, 10000), (255, 0, 0), 2)
    cv.imwrite("show_grid.jpg", ret)
    cv.imshow("grid", ret)
    cv.waitKey(0)
def in_thresh(source, target, thresh):
    '''是否在阈值范围内'''
    return target-thresh <= source <= target+thresh
def count_range_color(img, x, y, width, height, color, color_thresh=40):
    '''统计矩形范围内指定颜色像素的个数'''
    count = 0
    for i in range(width):
        for j in range(height):
            sb, sg, sr = img[y+j][x+i]
            tb, tg, tr = color
            if in_thresh(sb, tb, color_thresh) and in_thresh(sg, tg, color_thresh) and in_thresh(sr, tr, color_thresh):
                count += 1
    return count
# 墙的颜色
wall = (0, 0, 0)
def fix_v(x, max_v):
    "修正x,使0 <= x <= max_v"
    x = min(x, max_v)
    x = max(0, x)
    return x
def fix_x(img, x):
    return fix_v(x, img.shape[1])
def fix_y(img, y):
    return fix_v(y, img.shape[0])
def is_horz_wall(img, x, y, grid_size, precision=3):
    "是否是水平方向的墙 x,y为图片坐标, precision为选取测试的矩形范围,增强容错"
    w = int(grid_size / 2)  # 取中间的一半长度进行测试
    h = precision*2
    x = x + int(w/2)
    y = y - precision
    w = fix_x(img, x+w)-x
    h = fix_y(img, y+h)-y
    x = fix_x(img, x)
    y = fix_y(img, y)
    count = count_range_color(img, x, y, w, h, wall)
    logging.info(f"x:{x}, y:{y}, w:{w}, h:{h} count:{count}")
    if count >= w*0.8:
        return True
    return False
def is_vert_wall(img, x, y, grid_size, precision=3):
    "是否是垂直方向的墙 x,y为图片坐标"
    w = precision*2
    h = int(grid_size / 2)  # 取中间的一半长度进行测试
    x = x - precision
    y = y + int(h/2)
    w = fix_x(img, x+w)-x
    h = fix_y(img, y+h)-y
    x = fix_x(img, x)
    y = fix_y(img, y)
    count = count_range_color(img, x, y, w, h, wall)
    logging.info(f"x:{x}, y:{y}, w:{w}, h:{h} count:{count}")
    if count >= h*0.8:
        return True
    return False
def check_wall(img, grid_lines, x, y):
    "检测x,y指定格子四周是否有墙, 返回[上, 下, 左, 右]是否有墙的bool值"
    logging.info(f"check wall x:{x}, y:{y}")
    hls, vls = grid_lines
    grid_size = min(hls[1]-hls[0], vls[1]-vls[0])
    # left = x * grid_size + vls[0]
    # top = y * grid_size + hls[0]
    # right = left + grid_size
    # bottom = top + grid_size
    left = vls[x]
    right = vls[fix_v(x+1, len(vls)-1)]
    top = hls[y]
    bottom = hls[fix_v(y+1, len(hls)-1)]
    logging.info(f"left:{left}, right:{right}, top:{top}, bottom:{bottom}")
    top_wall = is_horz_wall(img, left, top, grid_size)
    bottom_wall = is_horz_wall(img, left, bottom, grid_size)
    left_wall = is_vert_wall(img, left, top, grid_size)
    right_wall = is_vert_wall(img, right, top, grid_size)
    return [top_wall, bottom_wall, left_wall, right_wall]
def find_in_range_pos(ranges, v):
    '''ranges必须为升序列表,
    查找v在ranges中的第一个位置索引'''
    for idx, v2 in enumerate(ranges):
        if v2 >= v:
            return idx
    return None
```

```
def find_grid_pos(img, grid_lines, x, y):
    "查找图像坐标x,y所在的格子"
    hls, vls = grid_lines
    x_pos = find_in_range_pos(vls, x) - 1
    y_pos = find_in_range_pos(hls, y) - 1
    return [x_pos, y_pos]
def rect_center(rect):
    '''计算矩形中心点'''
    x, y, w, h = rect
    return [x+int(w/2), y+int(h/2)]
```

下面继续是代码

```
# ------------------------------- maze 算法
def format_node(x, y):
    "格式化节点的表示"
    return f"{x}-{y}"
def generate_graph(img, grids):
    "从图片中生成graph"
    hls, vls = grids
    width = len(vls)-1
    height = len(hls)-1
    verticies = 0
    edges = 0
    graph = {}
    logging.info(f"width:{width}, height:{height}")
    for x in range(width):
        for y in range(height):
            verticies += 1
            node = format_node(x, y)
            graph[node] = set()
            top, down, left, right = check_wall(img, grids, x, y)
            if x >= 1:
                if not left:
                    graph[node].add(format_node(x-1, y))
                    edges += 1
            if x+1 < width:
                if not right:
                    graph[node].add(format_node(x+1, y))
                    edges += 1
            if y >= 1:
                if not top:
                    graph[node].add(format_node(x, y-1))
                    edges += 1
            if y+1 < height:
                if not down:
                    graph[node].add(format_node(x, y+1))
                    edges += 1
    print(verticies, "verticies")
    print(edges, "edges")
    return graph
def bfs_paths(graph, start, goal):
    queue = [(start, [start])]
    while queue:
        (vertex, path) = queue.pop(0)
        for next in graph[vertex] - set(path):
            if next == goal:
                yield path + [next]
            else:
                queue.append((next, path + [next]))
def shortest_path(graph, start, goal):
    '''查找最短路径'''
    try:
        return next(bfs_paths(graph, start, goal))
    except StopIteration:
        return None
def parse_node(node):
    "解析node为x,y坐标"
    return [int(i) for i in node.split('-')]
def get_direction(route):
    "获取路由每一步的方向，上下左右对应为1234"
    prev = parse_node(route[0])
    directs = []
    for curr in route[1:]:
        curr = parse_node(curr)
        x1, y1 = prev
        x2, y2 = curr
        if y2 < y1:
            directs.append('1')
        elif y2 > y1:
            directs.append('2')
        elif x2 < x1:
            directs.append('3')
        elif x2 > x1:
            directs.append('4')
        else:
```

```
            logging.error(f"error direction prev:{prev} current:{curr}")
        prev = curr
    return ''.join(directs)
def solve_maze(filename):
    '''解一个迷宫图片，返回每一步的路径'''
    img = cv.imread(filename)
    start = find_start(img)
    end = find_end(img)
    logging.info(f"image {filename} start pos: {start}, end pos: {end}.")
    # cv.imwrite("out.jpg", img)
    # show_rects(img, [start, end])
    # 格子的最小长度
    min_len = min(start[2], start[3], end[2], end[3])
    # 获取网格线
    grids = find_grid_lines(img, start, end, min_len)
    # show_grid(img, grids[0], grids[1])
    start_center = rect_center(start)
    start_pos = find_grid_pos(img, grids, start_center[0], start_center[1])
    end_center = rect_center(end)
    end_pos = find_grid_pos(img, grids, end_center[0], end_center[1])
    logging.info(f"start grid pos:{start_pos}, end grid pos:{end_pos}.")
    # check_wall(img, grids, x, y)
    g = generate_graph(img, grids)
    start_node = format_node(start_pos[0], start_pos[1])
    end_node = format_node(end_pos[0], end_pos[1])
    return [g, shortest_path(g, start_node, end_node)]
# -------------------------------- zip操作
zip_tmp = 'ziptmp/'
def unzip_file(filename, password):
    "解压zip文件，返回解压的文件列表"
    # 先解压到临时目录中
    if os.path.exists(zip_tmp):
        shutil.rmtree(zip_tmp)
    os.mkdir(zip_tmp)
    subprocess.run(['unzip', '-o', '-P', password, filename, '-d', zip_tmp])
    files = [f for f in listdir(zip_tmp) if isfile(join(zip_tmp, f))]
    print(f"unzip files:{files}.")
    # 然后把文件移动出来
    for f in files:
        if os.path.exists(f):
            os.unlink(f)
        shutil.move(join(zip_tmp, f), "./")
    return files
logging.getLogger().setLevel(logging.WARN)
count = 0
fname = "infinity_maze.jpg"
while True:
    g, route = solve_maze(fname)
    answer = get_direction(route)
    files = unzip_file(fname, answer)
    count += 1
    print(f"count: {count}")
    fname = "flag.jpg"
    if not fname in files:
        break
print("over!")
```

不断地解决迷宫，解压文件，经过128次之后，最终获得flag.txt文件，如图4。

注意这里解压zip文件使用了linux下的unzip工具，可以自动识别解压jpg文件末尾的zip文件。如果用python实现需要先提取出zip文件，再进行解压。

Fag

```
flag{af810046166d7b8a9c87227fcf341290}
```

- ***本文作者：*** *CTFHub*
- ***本文链接：*** *https://writeup.ctfhub.com/Challenge/2019/UNCTF/Misc/ade4VgXw1GPKUEkkZM2mZB.html*
- ***版权声明：*** *本博客所有文章除特别声明外，均采用 BY-NC-SA 许可协议。转载请注明出处！*

 # Challenge # Misc # 2019 # UNCTF
平淡生活下的秘密
贝斯的图