```python
## ==========================================
# ENPM661 Spring 2023: Robotic Path Planning
# Project #2
# Maze Search with Obstacles with Dijkstra's Algorithm
#
# Author: Doug Summerlin (dsumm1001@gmail.com, dsummerl@umd.edu)
# UID: 114760753
# Directory ID: dsummerl
# ==========================================
# Run as 'python3 Dij_maze_search.py'
# Press CTRL+C for exit

import numpy as np
import matplotlib.pyplot as plt
import cv2
import math
from queue import PriorityQueue
import time

def getValidInput(type, maze):
    thresh = 5
    while True:
        try:
            coordInput = input("Enter " + type + " node coordinates in x, y format, separated by a comma: ")
            coords = tuple(int(item) for item in coordInput.split(','))
        except ValueError:
            print("Sorry, results invalid. Please try again, entering two integer inputs between 5-595 and 5-245, respectively. ")
            continue
        if coords[0] < 0 + thresh or coords[0] > 600 - thresh or coords[1] < 0 + thresh or coords[1] > 250 - thresh:
            print("Sorry, results invalid. Please try again, entering two integer inputs between 5-595 and 5-245, respectively. ")
            continue
        if checkObstacle(coords, maze) == True:
            print("Sorry, results invalid. Please try again, making sure to not place the start or goal in an obstacle space.")
            continue
        else:
            break
    return coords

def drawMaze():
    mazeSize = (250,600)

    # Create blank maze
    maze = np.zeros((mazeSize[0], mazeSize[1], 3), dtype = np.uint8)
    maze[:] = (0, 255, 0)
    cv2.rectangle(maze, pt1=(5,5), pt2=(595,245), color=(255,255,255), thickness= -1)

    # draw rectangle obstacles
    cv2.rectangle(maze, pt1=(95,0), pt2=(155,105), color=(0,255,0), thickness= -1)
    cv2.rectangle(maze, pt1=(95,145), pt2=(155,mazeSize[1]), color=(0,255,0), thickness= -1)

    cv2.rectangle(maze, pt1=(100,0), pt2=(150,100), color=(0,0,255), thickness= -1)
    cv2.rectangle(maze, pt1=(100,150), pt2=(150,mazeSize[1]), color=(0,0,255), thickness= -1)

    # draw hexagonal obstacle and boundary
    hexBoundPts = np.array([[300, 44], [370, 84],
                            [370, 166], [300, 206],
                            [230, 166], [230,84]])
    cv2.fillConvexPoly(maze, hexBoundPts, color=(0, 255, 0))

    hexPts = np.array([[300, 125-75], [365, math.ceil(125-37.5)],
                            [365, math.ceil(125+37.5)], [300, 125+75],
                            [235, math.ceil(125+37.5)], [235, math.ceil(125-37.5)]])
    cv2.fillConvexPoly(maze, hexPts, color=(0, 0, 255))

    # draw triangular obstacle
    cv2.circle(maze, [460, 25], 5, color=(0, 255, 0), thickness=-1)
    cv2.circle(maze, [460, 225], 5, color=(0, 255, 0), thickness=-1)
    cv2.circle(maze, [510, 125], 5, color=(0, 255, 0), thickness=-1)

    cv2.rectangle(maze, pt1=(455,25), pt2=(460,225), color=(0,255,0), thickness= -1)

    triUpperBoundPts = np.array([[460, 25], [465, 22], [516, 125], [510, 125]])
    cv2.fillConvexPoly(maze, triUpperBoundPts, color=(0, 255, 0))

    triLowerBoundPts = np.array([[510, 125], [516, 125], [465, 228], [460, 225]])
    cv2.fillConvexPoly(maze, triLowerBoundPts, color=(0, 255, 0))

    triPts = np.array([[460, 25], [460, 225], [510, 125]])
    cv2.fillConvexPoly(maze, triPts, color=(0, 0, 255))
    return maze

def checkObstacle(xyCoords, maze):
    if all(maze[xyCoords[1], xyCoords[0]] == [255,255,255]):
        return False
    else:
        return True

# [c2c, index, coords, cost, actioncost]
def actMoveRight(curr, maze):
    if (checkObstacle((curr[0]+1,curr[1]), maze) == False):
        newRight = [None, None, (curr[0]+1, curr[1]), None, 1]
        return newRight
    else:
        return None

def actMoveLeft(curr, maze):
    if (checkObstacle((curr[0]-1,curr[1]), maze) == False):
        newLeft = [None, None, (curr[0]-1, curr[1]), None, 1]
        return newLeft
    else:
        return None

def actMoveUp(curr, maze):
    if (checkObstacle((curr[0],curr[1]+1), maze) == False):
        newUp = [None, None, (curr[0], curr[1]+1), None, 1]
        return newUp
    else:
        return None

def actMoveDown(curr, maze):
    if (checkObstacle((curr[0],curr[1]-1), maze) == False):
        newDown = [None, None, (curr[0], curr[1]-1), None, 1]
        return newDown
    else:
        return None

def actMoveUpRight(curr, maze):
    if (checkObstacle((curr[0]+1,curr[1]+1), maze) == False):
        newUpRight = [None, None, (curr[0]+1, curr[1]+1), None, 1.4]
        return newUpRight
    else:
        return None

def actMoveUpLeft(curr, maze):
    if (checkObstacle((curr[0]-1,curr[1]+1), maze) == False):
        newUpLeft = [None, None, (curr[0]-1, curr[1]+1), None, 1.4]
        return newUpLeft
    else:
        return None

def actMoveDownRight(curr, maze):
    if (checkObstacle((curr[0]+1,curr[1]-1), maze) == False):
        newDownRight = [None, None, (curr[0]+1, curr[1]-1), None, 1.4]
        return newDownRight
    else:
        return None

def actMoveDownLeft(curr, maze):
    if (checkObstacle((curr[0]-1,curr[1]-1), maze) == False):
        newDownLeft = [None, None, (curr[0]-1, curr[1]-1), None, 1.4]
        return newDownLeft
    else:
        return None

def searchNode(nodeCoords, maze):
    right = actMoveRight(nodeCoords, maze)
    left = actMoveLeft(nodeCoords, maze)
    up = actMoveUp(nodeCoords, maze)
    down = actMoveDown(nodeCoords, maze)
    upRight = actMoveUpRight(nodeCoords, maze)
    upLeft = actMoveUpLeft(nodeCoords, maze)
    downRight = actMoveDownRight(nodeCoords, maze)
    downLeft = actMoveDownLeft(nodeCoords, maze)

    results = []

    if right is not None:
        results.append(right)
    if left is not None:
        results.append(left)
    if up is not None:
        results.append(up)
    if down is not None:
        results.append(down)
    if upRight is not None:
        results.append(upRight)
    if upLeft is not None:
        results.append(upLeft)
    if downRight is not None:
        results.append(downRight)
    if downLeft is not None:
        results.append(downLeft)

    return results

def generatePath(nodeIndex, nodeCoords, maze):
    pathIndices = []
    pathCoords = []

    while nodeIndex is not None:
        pathIndices.append(nodeIndex)
        pathCoords.append(nodeCoords)
        cv2.circle(maze, (int(nodeCoords[0]),int(nodeCoords[1])), 1, color=(0,255,255), thickness=-1)
        nodeCoords = coordDict[nodeIndex]
        nodeIndex = parentDict[nodeIndex]

    return pathIndices, pathCoords

def simulateBot(pathCoords, searchedNodeCoords, emptyMaze):
    index = 90
    while index >=0:
        index -= 1
        outVid.write(cv2.flip(emptyMaze,0))

    for i in searchedNodeCoords:
        emptyMaze[i[1], i[0]] = (255,0,0)
        index += 1
        if index >= 50:
            outVid.write(cv2.flip(emptyMaze,0))
            index = 0

    for i in pathCoords:
        cv2.circle(emptyMaze, (int(i[0]),int(i[1])), 1, color=(0,255,255), thickness=-1)
        outVid.write(cv2.flip(emptyMaze,0))

    pathCoords.reverse()

    for i in pathCoords:
       emptyMazeCopy = emptyMaze.copy()
       currCirc = cv2.circle(emptyMazeCopy, (int(i[0]),int(i[1])), 5, color=(255,0,255), thickness=-1)
       outVid.write(cv2.flip(currCirc,0))

    index = 60
    while index >=0:
        index -= 1
        outVid.write(cv2.flip(currCirc,0))

print("\nWelcome to the Dijkstra Maze Finder Program! \n")

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
outVid = cv2.VideoWriter('output.mp4', fourcc, 60, (600,250))
#outVid = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc(*'MJPG'), 60, (600,250))

maze = drawMaze()
blankMaze = maze.copy()

# get start and goal nodes
start = getValidInput("start", maze)
goal = getValidInput("goal", maze)
print()
print("Pathfinding... \n")

startTime = time.time()

solved = False
openList = PriorityQueue()

# intialize data containers for backtracking
parentDict = {1:None}
coordDict = {1:start}
```

```python
closedSet = set()
closedList = []
openSet = set()

# [c2c, index, coords, cost, actioncost]
startNode = [0, 1, start, 0, 0]
index = startNode[1]

openList.put(startNode)
openSet.add(start)
maze[startNode[2][1], startNode[2][0]] = (255,0,255)

while not openList.empty() and solved == False:
    first = openList.get()
    openSet.remove(first[2])
    closedSet.add(first[2])
    closedList.append(first[2])

    if ((first[2] == goal)):
        elapsedTime = time.time() - startTime
        print ("Yay! Goal node located... Operation took ", elapsedTime, " seconds.")
        print("Current node index: ", first[1], " and cost: ", round(first[3],2), "\n")
        solved = True

        pathIndices, pathCoords = generatePath(first[1], first[2], maze)
        print("Displaying generated path... close window to continue \n")
        # # display the path image using opencv
        dispMaze = maze.copy()
        dispMaze = cv2.flip(dispMaze, 0)
        dispMaze = cv2.resize(dispMaze, (1200,500), interpolation = cv2.INTER_AREA)
        cv2.imshow('Maze', dispMaze)
        cv2.waitKey(0)

        print("Generating simulation...")
        simulateBot(pathCoords, closedList, blankMaze)
        print("Simulation complete! \n")
        break

    results = searchNode(first[2], maze)

    for i in results:
        if not i[2] in closedSet:
            maze[i[2][1], i[2][0]] = (255,0,0)
            if not i[2] in openSet:
                index += 1
                i[1] = index
                i[0] = first[0] + i[4]
                i[3] = i[0]

                parentDict[i[1]] = first[1]
                coordDict[i[1]] = i[2]

                openList.put(i)
                openSet.add(i[2])
        else:
            if i[3] > first[0] + i[4]:
                parentDict[i[1]] = first[1]
                i[0] = first[0] + i[4]
                i[3] = i[0]

if solved == False:
    print ("Failure! Goal node not found")

print("Saving video... ")
outVid.release()
print("Video saved successfully! Displaying video... \n")

cap = cv2.VideoCapture('output.mp4')

if cap.isOpened() == False:
    print("Error File Not Found")

while cap.isOpened():
    ret,frame= cap.read()
    if ret == True:
        cv2.imshow('frame', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break

cap.release()
print("Video displayed successfully! Program termination  \n")
cv2.destroyAllWindows()

# Resources
# https://www.programiz.com/dsa/priority-queue
# https://bobbyhadz.com/blog/python-input-tuple
# https://stackoverflow.com/questions/23294658/asking-the-user-for-input-until-they-give-a-valid-response
# https://www.w3schools.com/python/python_sets.asp
# https://www.freecodecamp.org/news/python-set-how-to-create-sets-in-python/#:~:text=How%20to%20Add%20Items%20to%20a%20Set%20in%20Python,passed%20in%20as%20a%20parameter.&text=We%20add
# https://stackoverflow.com/questions/30103077/what-is-the-codec-for-mp4-videos-in-python-opencv
# https://docs.opencv.org/3.4/dd/d43/tutorial_py_video_display.html
# https://www.geeksforgeeks.org/python-play-a-video-using-opencv/
```