Experiment-2

Yash Patel, Ojas Patil

2019130047 , 2019130048

TE Comps

Batch C

**Aim**: To implement the fire extinguisher using BFS and DFS

**Theory**:

* Depth-first Search (DFS):
  1. DFS always expands DEPTH-FIRST the deepest node in the current frontier of the search tree.
  2. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors.
  3. DFS uses a LIFO queue.
  4. Visits children before siblings.
* Breadth-first Search (BFS):
  1. BFS is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.
  2. All the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
  3. BFS uses a FIFO queue.
  4. Visits siblings before children.

**Code**:

# tree.py

from tkinter import \*

from tree import Node, Holder

import random

import copy

def create\_grid(grid\_size\_length, grid\_size\_breadth, event=None):

w = grid\_size\_length\*100+1 # Get current width of canvas

h = grid\_size\_breadth\*100+1 # Get current height of canvas

c.delete('grid\_line') # Will only remove the grid\_line

# Creates all vertical lines at intevals of 100

for i in range(0, w, 100):

c.create\_line([(i, 0), (i, h)], tag='grid\_line')

# Creates all horizontal lines at intevals of 100

for i in range(0, h, 100):

c.create\_line([(0, i), (w, i)], tag='grid\_line')

def checkBound(x, y):

if x >= 0 and y >= 0 and x < grid\_size\_length and y < grid\_size\_breadth and (grid[x][y]).fire == False:

return True

else:

return False

root = Tk()

c = Canvas(root, height=1000, width=1000, bg='white')

c.pack(fill=BOTH, expand=True)

dx = [0, 1, 0, -1]

dy = [-1, 0, 1, 0]

grid\_size\_breadth = int(input("Enter number of rows in grid: "))

grid\_size\_length = int(input("Enter number of columns in grid: "))

c.create\_rectangle(grid\_size\_length\*100+50, 50, grid\_size\_length\*100+100, 100, fill='red')

c.create\_text(grid\_size\_length\*100+150, 75, fill='black', text=f"Fire", font=('Helvetica 15 bold'))

c.create\_rectangle(grid\_size\_length\*100+50, 150, grid\_size\_length\*100+100, 200, fill='blue')

c.create\_text(grid\_size\_length\*100+200, 175, fill='black', text=f"Extenguisher On", font=('Helvetica 15 bold'))

c.create\_rectangle(grid\_size\_length\*100+50, 250, grid\_size\_length\*100+100, 300, fill='purple')

c.create\_text(grid\_size\_length\*100+200, 275, fill='black', text=f"Fire + Extinguisher", font=('Helvetica 15 bold'))

c.create\_rectangle(grid\_size\_length\*100+50, 350, grid\_size\_length\*100+100, 400, fill='cyan')

c.create\_text(grid\_size\_length\*100+200, 375, fill='black', text=f"Selected path", font=('Helvetica 15 bold'))

c.create\_rectangle(grid\_size\_length\*100+50, 450, grid\_size\_length\*100+100, 500, fill='yellow')

c.create\_text(grid\_size\_length\*100+200, 475, fill='black', text=f"Stucked person", font=('Helvetica 15 bold'))

c.create\_rectangle(grid\_size\_length\*100+50, 550, grid\_size\_length\*100+100, 600, fill='black')

c.create\_text(grid\_size\_length\*100+200, 575, fill='black', text=f"Exit", font=('Helvetica 15 bold'))

c.bind('<Configure>',create\_grid(grid\_size\_length, grid\_size\_breadth))

grid = [[None for \_ in range(grid\_size\_breadth)] for \_ in range(grid\_size\_length)]

for i in range(grid\_size\_length):

for j in range(grid\_size\_breadth):

grid[i][j] = Node()

no\_of\_points = int(input("Enter number of points where fire has been detected: "))

fire\_point = []

while no\_of\_points > 0:

x = random.randint(1, grid\_size\_length)

y = random.randint(1, grid\_size\_breadth)

if (x, y) not in fire\_point and (x,y) != (grid\_size\_length-1, grid\_size\_breadth-1):

fire\_point.append((x, y))

no\_of\_points -= 1

for i in fire\_point:

c.create\_rectangle((i[0]-1)\*100, (i[1]-1)\*100, (i[0]-1)\*100+100, (i[1]-1)\*100+100, fill='red')

for point in fire\_point:

grid[point[0] - 1][point[1] - 1].fire = True

def BFS():

visited = [[False for \_ in range(grid\_size\_length)] for \_ in range(grid\_size\_breadth)]

queue = []

extinguisers\_turned\_on = []

no\_of\_extinguishers = 0

# FIFO -> first in first out

for i in range(grid\_size\_length):

for j in range(grid\_size\_breadth):

if grid[i][j].fire == True:

no\_of\_extinguishers += 1

queue.append((i, j))

dx = [0, 0, -1, 1, 1, -1, 1, -1]

dy = [-1, 1, 0, 0, 1, 1, -1, -1]

while len(queue) > 0:

current = queue[0]

queue.pop(0)

x = current[0]

y = current[1]

visited[x][y] = 1

for i in range(8):

xx = x + dx[i]

yy = y + dy[i]

if xx < 0 or yy < 0 or xx >= grid\_size\_length or yy >= grid\_size\_breadth or grid[xx][yy] == False or visited[xx][yy]:

continue

extinguisers\_turned\_on.append((xx + 1, yy + 1))

# no\_of\_extinguishers += 1

return extinguisers\_turned\_on

\_ = input()

extinguisers\_turned\_on = BFS()

for ele in extinguisers\_turned\_on:

if ele in fire\_point:

extinguisers\_turned\_on.remove(ele)

print(set(extinguisers\_turned\_on))

for i in fire\_point:

c.create\_rectangle((i[0]-1)\*100, (i[1]-1)\*100, (i[0]-1)\*100+100, (i[1]-1)\*100+100, fill='purple')

for i in set(extinguisers\_turned\_on):

c.create\_rectangle((i[0]-1)\*100, (i[1]-1)\*100, (i[0]-1)\*100+100, (i[1]-1)\*100+100, fill='blue')

print("Extinguishers online")

dx = [0, 1, 0, -1]

dy = [-1, 0, 1, 0]

def \_BFS(x, y):

found = False

queue = []

cur = Holder(x, y)

(grid[cur.x][cur.y]).dis = 0

queue.append(cur)

while len(queue) > 0:

cur = queue[0]

queue.pop(0)

if (cur.x == grid\_size\_length - 1) and (cur.y == grid\_size\_breadth - 1):

found = True

for i in range(4):

nt = copy.deepcopy(cur)

nt.x += dx[i]

nt.y += dy[i]

if checkBound(nt.x, nt.y) and grid[nt.x][nt.y].dis == -1:

(grid[nt.x][nt.y]).dis = (grid[cur.x][cur.y]).dis + 1

queue.append(nt)

for i in range(grid\_size\_length):

for j in range(grid\_size\_breadth):

c.create\_text(i\*100+50, j\*100+50, fill='yellow', text=f"{grid[i][j].dis}", font=('Helvetica 15 bold'))

input()

return found

path = []

ans = []

def DFS(stuck\_co\_ordinate, x, y, d):

if x == stuck\_co\_ordinate[0] and y == stuck\_co\_ordinate[1]:

onePath = list()

onePath.append(tuple((grid\_size\_length - 1, grid\_size\_breadth - 1)))

for i in range(d-1, -1, -1):

onePath.append(ans[i])

path.append(onePath)

return

for i in range(4):

nx = x + dx[i]

ny = y + dy[i]

if checkBound(nx, ny) and (grid[x][y]).dis - 1 == (grid[nx][ny]).dis:

ans.append(tuple((nx, ny)))

DFS(stuck\_co\_ordinate, nx, ny, d+1)

ans.pop()

stuck\_co\_ordinate = tuple(map(int, input("Are you stuck? tell us your location: ").strip().split()))

stuck\_co\_ordinate = tuple((stuck\_co\_ordinate[0] - 1, stuck\_co\_ordinate[1] - 1))

if grid[grid\_size\_length-1][grid\_size\_breadth-1].fire == True:

c.create\_rectangle((grid\_size\_length-1)\*100, (grid\_size\_breadth-1)\*100, grid\_size\_length\*100, grid\_size\_breadth\*100, fill='black')

else:

c.create\_rectangle((grid\_size\_length-1)\*100, (grid\_size\_breadth-1)\*100, grid\_size\_length\*100, grid\_size\_breadth\*100, fill='maroon')

if not \_BFS(stuck\_co\_ordinate[0], stuck\_co\_ordinate[1]):

print("""Sorry to inform you that there is no path available from your position\n

# Please stay there until our fire extinguishers extinguish the fire and make a way for you\n

# It won't take long and keep yourself away from flames.""")

else:

DFS(stuck\_co\_ordinate, grid\_size\_length - 1, grid\_size\_breadth - 1, 0)

if len(path) > 0:

print("Follow one of these paths")

for i in path:

i.append(i.pop(0))

for j in i:

print(f"({j[0] + 1},{j[1] + 1})", end=" -> ")

print("\n")

selectedPath = random.randint(0, len(path)-1)

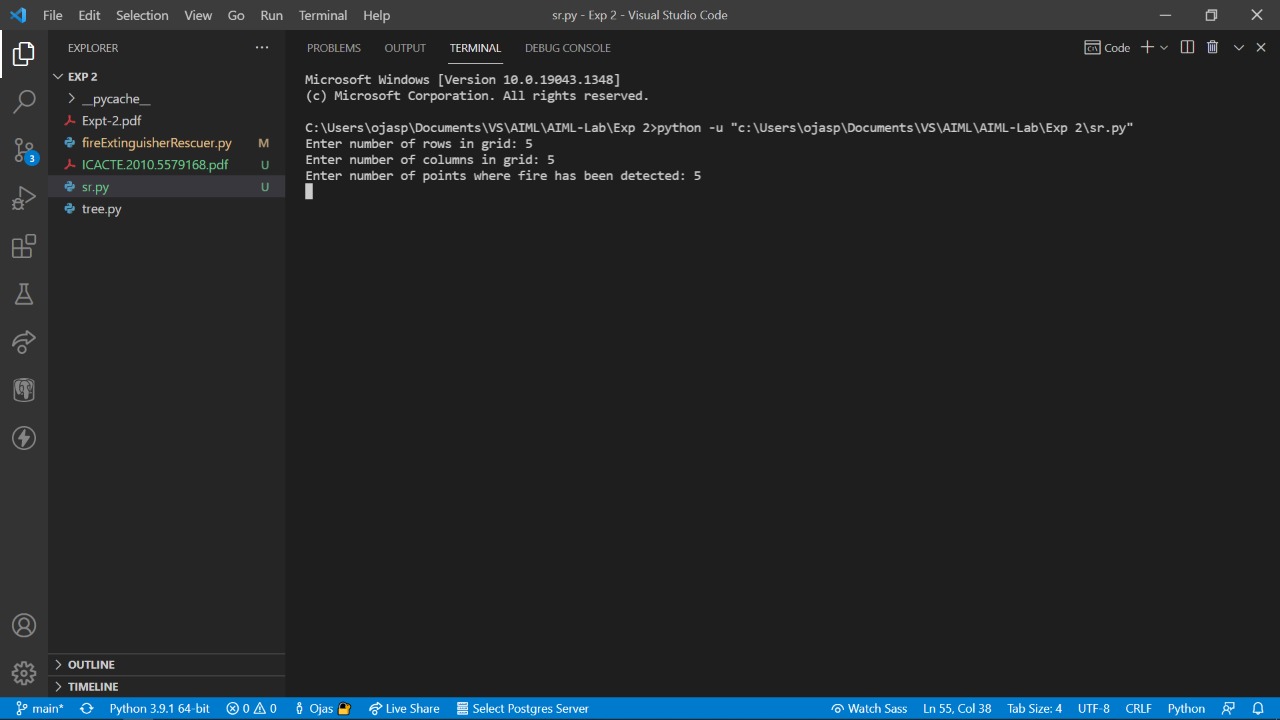
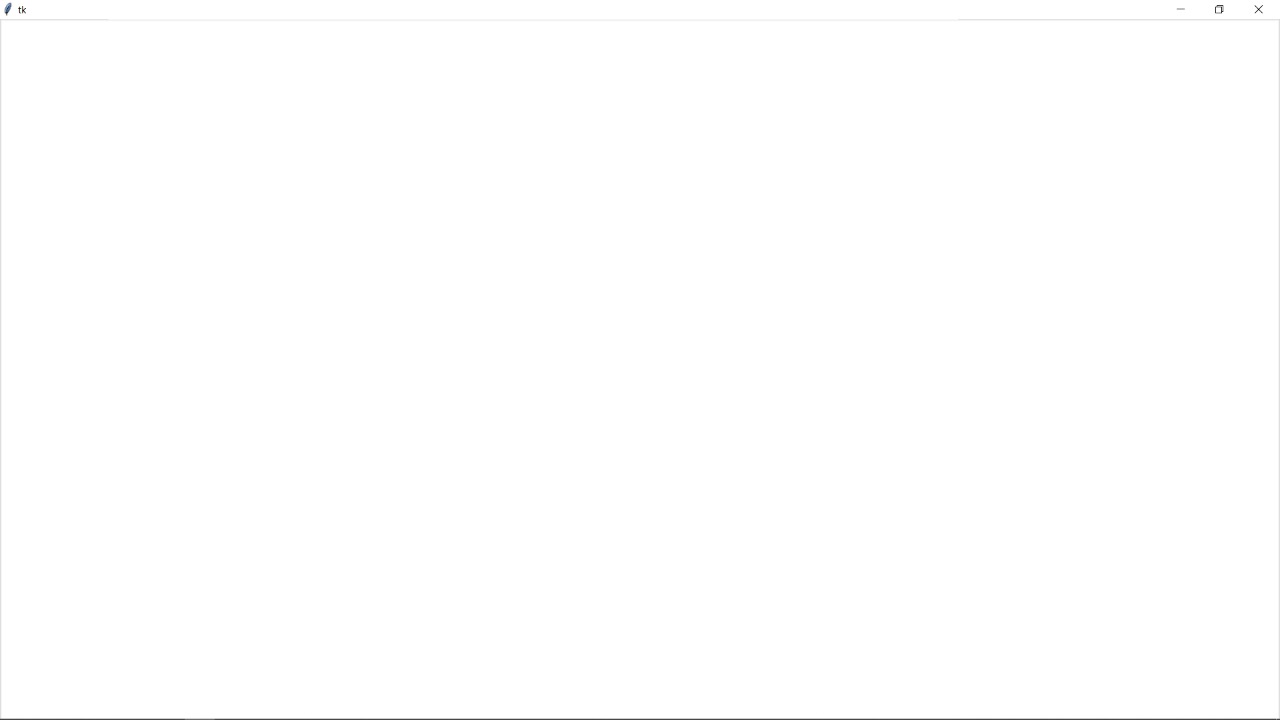
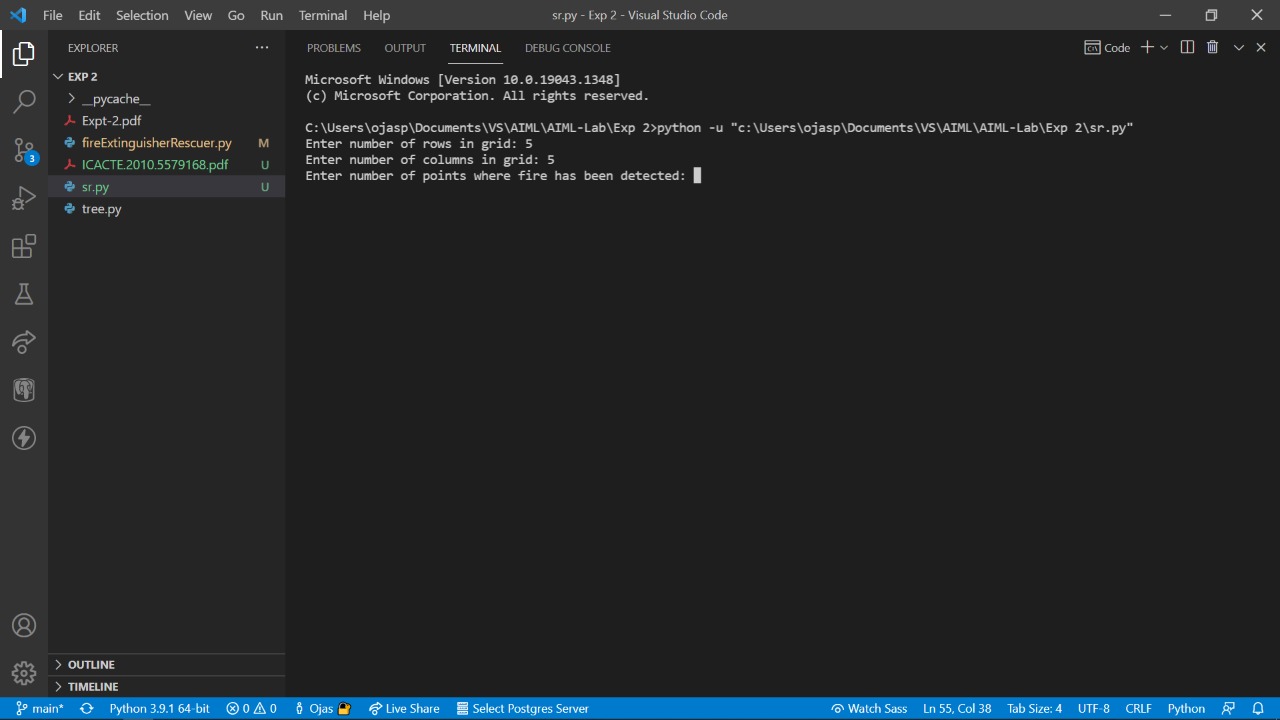
for j in path[selectedPath]:

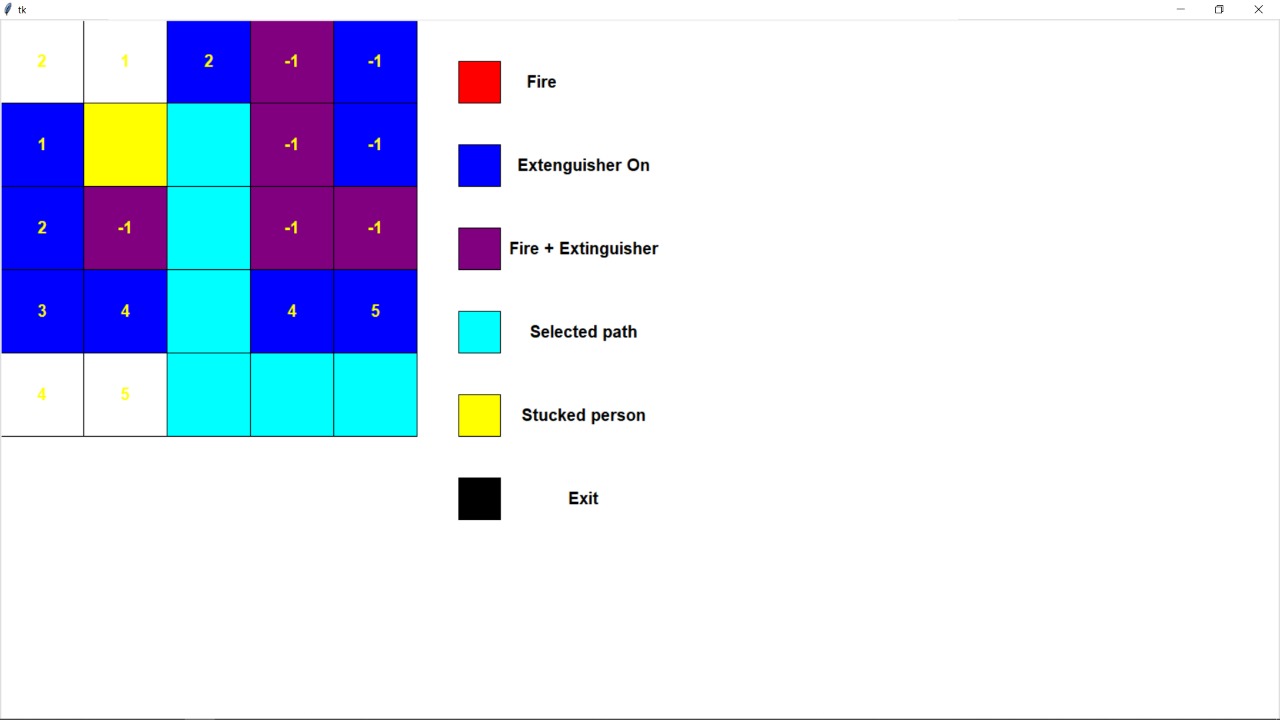
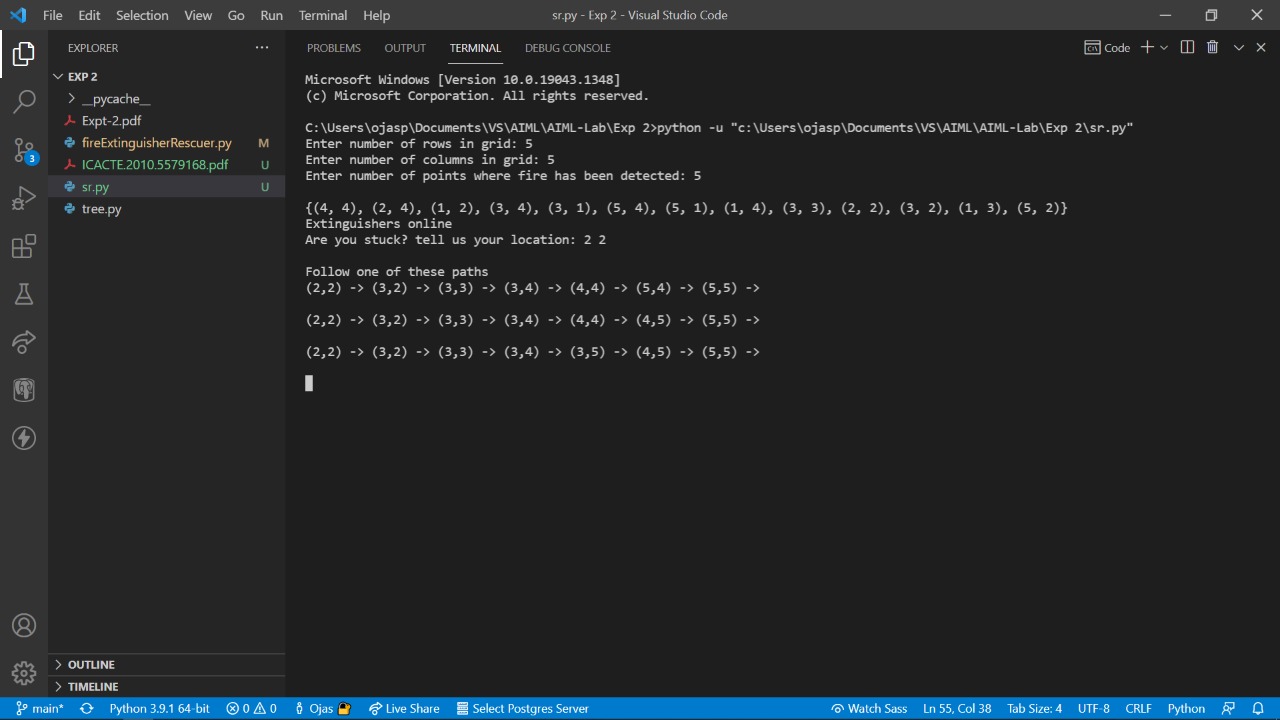
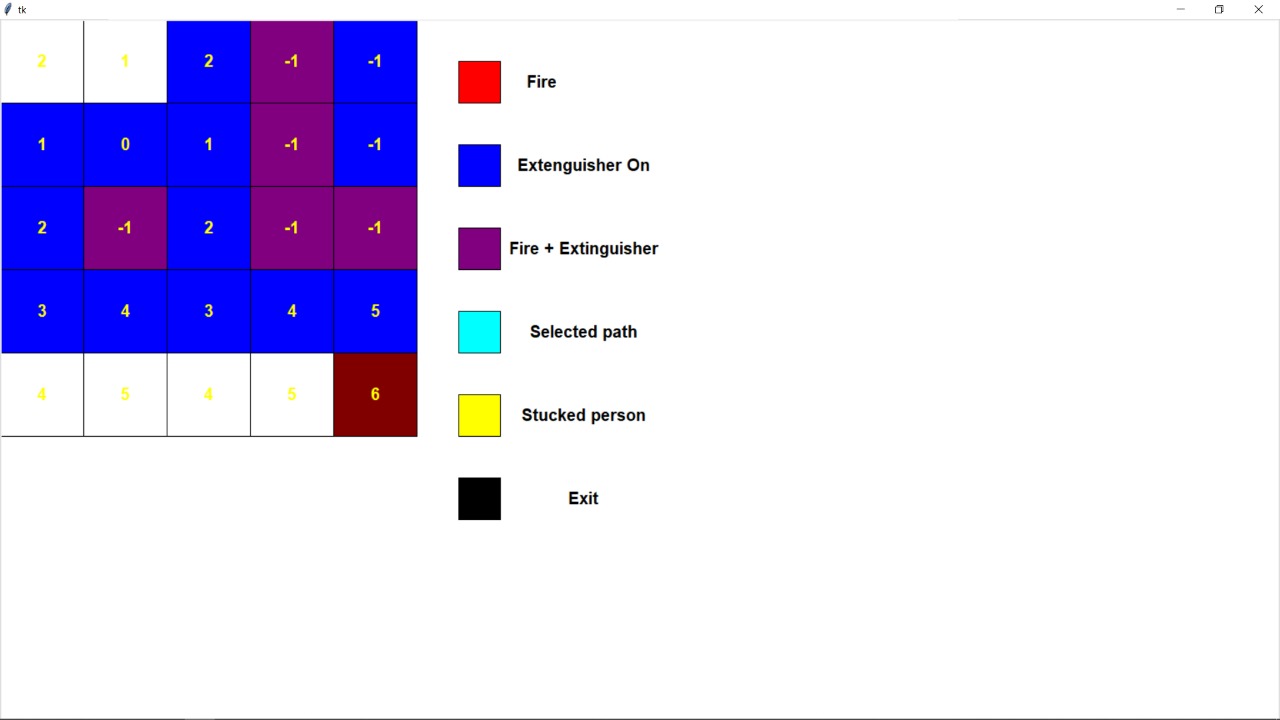
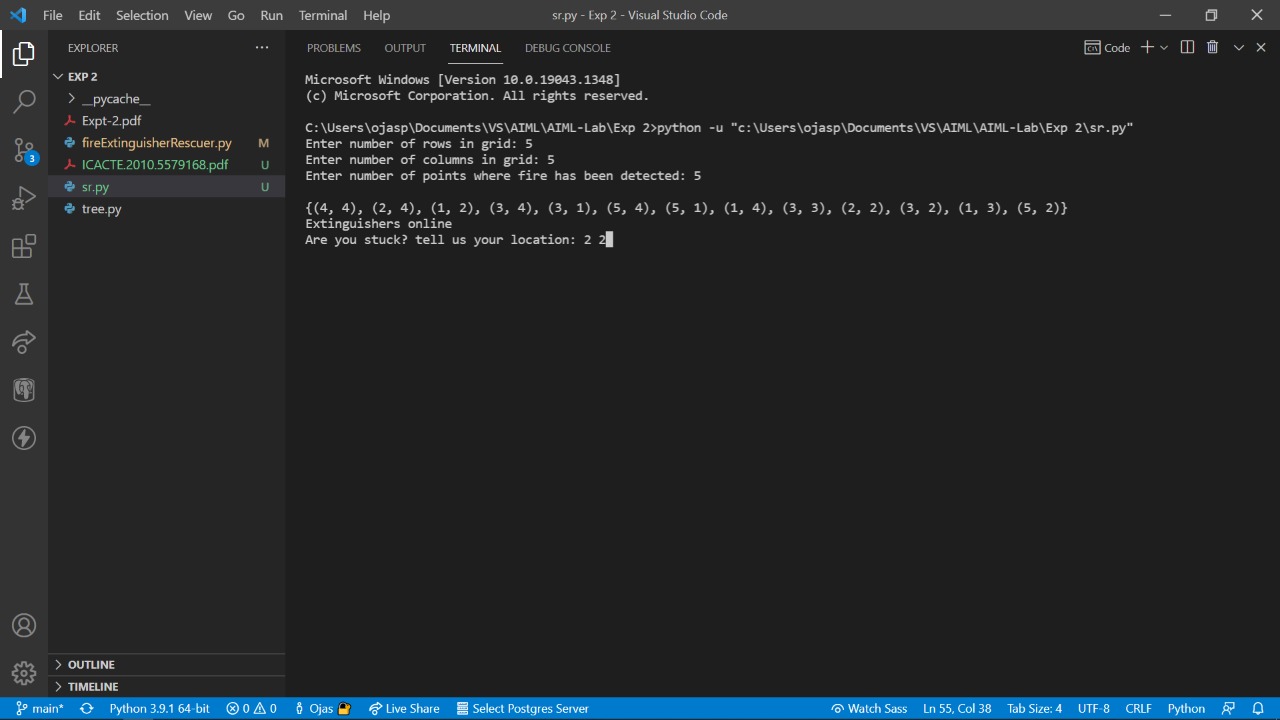
c.create\_rectangle((j[0])\*100, (j[1])\*100, j[0]\*100+100, j[1]\*100+100, fill='cyan')

c.create\_rectangle(stuck\_co\_ordinate[0]\*100, stuck\_co\_ordinate[1]\*100, stuck\_co\_ordinate[0]\*100+100, stuck\_co\_ordinate[1]\*100+100, fill='yellow')

root.mainloop()

**Input/Output**:





**Conclusion**:

This experiment takes place on a floor that is divided into a grid of size NxM. Each node is equipped with two extinguishers as well as a sensor. If any of the nodes catch fire, the BFS search is launched to locate all of the nodes in the next frontier, and the fire extinguishers are activated to put out the fire. If the location of a person who has become stuck in one of the nodes is known, a BFS search is run from that place to discover the shortest path to the exit door in the room's lower right corner. Because there may be several shortest paths, we use DFS to offer the user with all available paths so that he or she can escape the room. As a result, we learned how to use DFS and BFS algorithms.

**Links**:

* Yash Patel: <https://github.com/yash19pro/AI-ML-Lab>
* Ojas Patil: <https://github.com/PatilOjas/AIML-Lab>