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 to 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.

Problem Statement:
An intelligent fire extinguisher system that detects fire in a cell of a given grid and turns on the extinguisher system not only to extinguish fire but also to contain it. It also provides all the paths available to a trapped person so that he/she can rescue himself/herself safely.

Working:
In current implementation, we are using a function to randomly ignite fire at random places in the given grid.
1) Once the fire has started, the system uses a BFS function to find all the cells that are in the first frontier and surround the cell on fire. It starts the fire extinguisher system of these surrounding cells as well as cells on fire so as to contain fire and extinguish it.
2) Once the extinguisher systems are online, we now ask the user to provide his location within the grid. Once the position is provided, the system again runs a BFS on grid from the trapped position and calculates the distance of each cell which is not on fire and is accessible from the trapped position. We assume that only one exit door exists and it is at the lower right corner of the grid. After calculating the distance, we get the minimum distance required to reach the door. After that, we run a DFS on the same grid from the exit door location till we find out each path that leads to the trapped position. In this way, we get all the paths that are available for the trapped person to rescue himself.
(We have selected one random available path and showed it in the output GUI)

Code:

fireExtinguisher.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 intervals of 100
   for i in range(0, w, 100):
       c.create_line([(i, 0), (i, h)], tag='grid_line')

   # Creates all horizontal lines at intervals 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"Extinguisher 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')
```

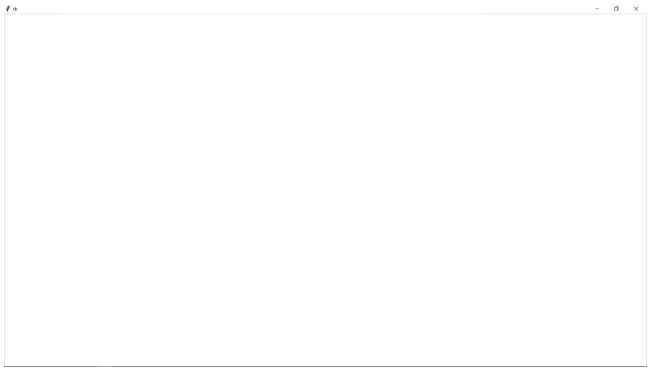
```
c.create_rectangle(stuck_co_ordinate[0]*100, stuck_co_ordinate[1]*100,
stuck_co_ordinate[0]*100+100, stuck_co_ordinate[1]*100+100, fill='yellow')
root.mainloop()
```

tree.py

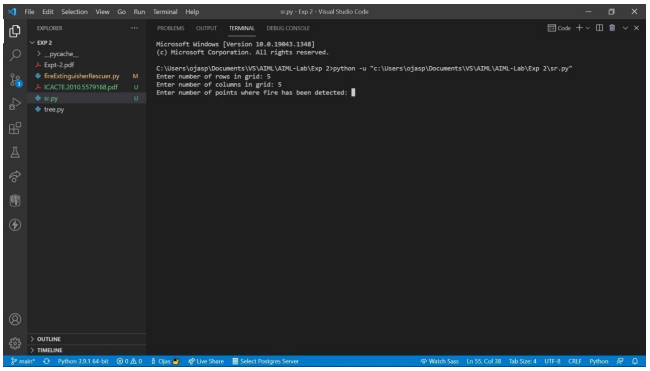
```
class Node:
 def __init__(self):
   self.fire = False # If fire is there in this node, turn self.fire = True
   self.DFS = False # If you explore this node using BFS, turn self.BFS =
True
   self.BFS = False # If you explore this node using DFS, turn self.DFS =
True
   self.dis = -1

class Holder:
 def __init__(self, x, y) -> None:
   self.x = x
   self.y = y
```

Input/Output:

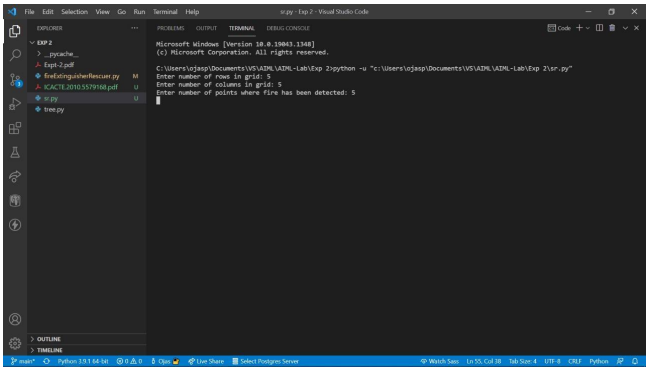


Started



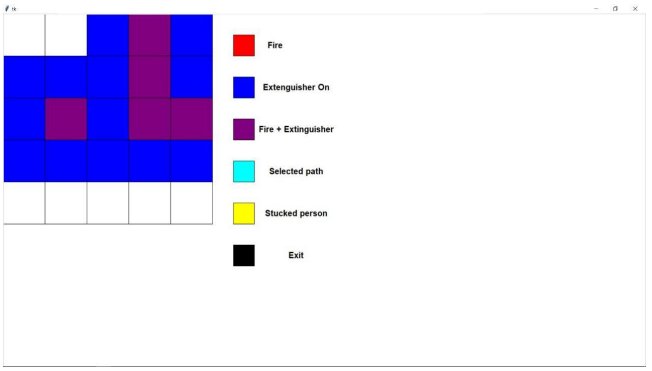
Grid Size defined

Grid with legends

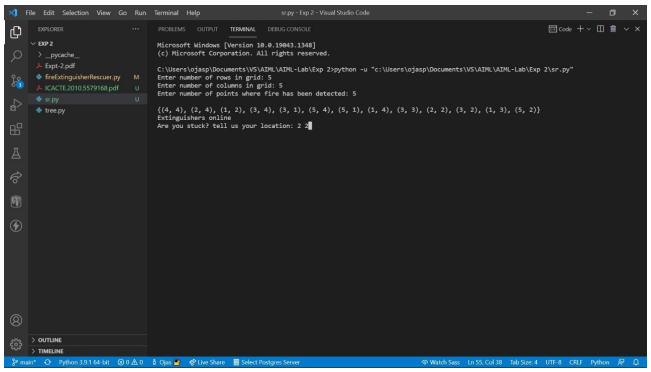


Number of fire places mentioned

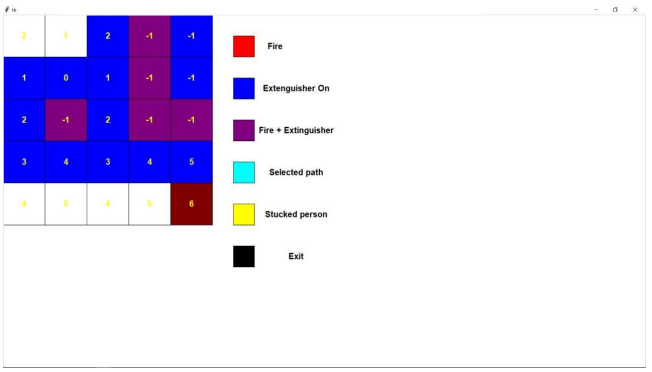
Fire starts at random places



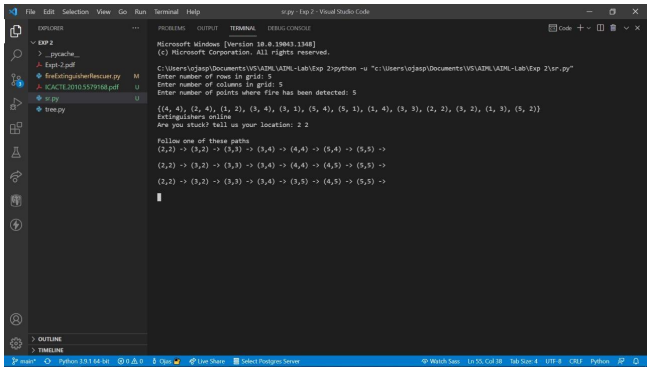
Extinguisher system ONLINE!



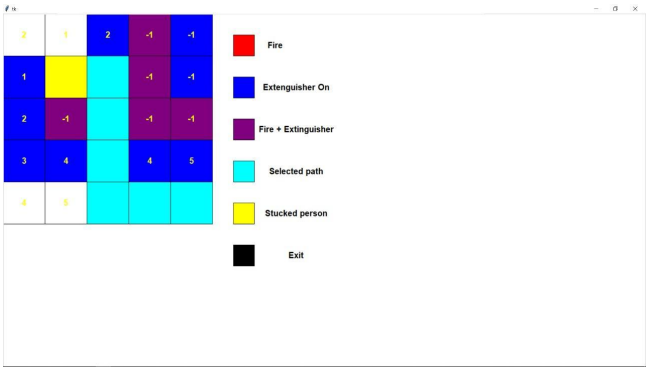
Trapped position provided



Working of BFS is shown by mentioning distances in the cell itself



All the available paths are printed



One path is selected randomly

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

  Video is available on both of the above Github repositories.