Name : Sudhanshu Narsude, Apurv Sarode

UID : 2019130044 , 2019130054

Subject : AIML

Experiment : 2

Aim : Explore the environment by employing the breadth-first and the depth-first search algorithms.

Theory:

SEARCH STRATEGIES

In order to perform a systematic search, the two fundamental strategies are breadth-first and depth-first. In addition, the agent needs to select among nodes at the same level in a systematic manner. The most frequently used selections here are left-to-right or right-to-left with respect to the way the nodes are listed in a graphical view. Depending on the type of the nodes, there is also often an obvious ordering, such as alphabetical or numerical. In this task, use a clockwise ordering starting from the "North" position (i.e. in the order North, East, South, West).

BREADTH-FIRST SEARCH (BFS) ALGORITHM

In this strategy, all direct successor nodes of a given level are explored first, and then the successor nodes of the first node at this level are examined. In the tree, this means that all nodes at a given level are explored before proceeding to the next level.

DEPTH-FIRST SEARCH (DFS) ALGORITHM

This search strategy picks the first successor node of the current node, then the first successor node of that node, and so on. In the tree, this means that all nodes of a branch are explored until a leaf node is reached. Once a leaf node is reached, the algorithm "backtracks" (goes up one level), and selects the next node at that level as the new starting point of a sub-branch.

Program of BFS :

```
from tkinter import *
from time import *
import random
from collections import defaultdict
from queue import Queue
import sys
import gc

gc.collect()

# function for adding edge to graph
graph = defaultdict(list)
def addEdge(graph,u,v):
```

```
    graph[u].append(v)

# definition of function
def generate_edges(graph):
    edges = []

    # for each node in graph
    for node in graph:

        # for each neighbour node of a single node
        for neighbour in graph[node]:

            # if edge exists then append
            edges.append((node, neighbour))
    return edges


# direction includes right,left,up,down,suck,nop
def dirs(vacpos,act,flag): #flag 0 dont move vac
    global visited_pieces
    if act == 'right':
        if (flag==1):
            vac_pos[1] = vac_pos[1] + 1
        else:
            return vacpos[0]+str(int(vacpos[1]) + 1)
    elif act == 'left':
        if (flag==1):
            vac_pos[1] = vac_pos[1] - 1
        else:
            return vacpos[0]+str(int(vacpos[1]) - 1)
    elif act == 'up':
        if (flag==1):
            vac_pos[0] = vac_pos[0] - 1
        else:
            return str(int(vacpos[0])-1)+vacpos[1]

    elif act == 'down':
        if (flag==1):
            vac_pos[0] = vac_pos[0] + 1
        else:
            return str(int(vacpos[0])+1)+vacpos[1]

    if (flag==1):
        visited_pieces = visited_pieces + 1

def actuator(act):
    global cleaned_pieces
    if act == 'suck':
        maps[vac_pos[0]-1][vac_pos[1]-1] = 0
```

```
        cleaned_pieces = cleaned_pieces + 1

# '0' means clean & '1' means dirty
def initmap(row,col):
    global dirty_pieces
    global maps
    global visited
    for j in range(row):
        tmp = []
        tmpv = []
        for i in range(col):
            rand = random.randint(0,1)
            if rand == 1 :
                dirty_pieces = dirty_pieces + 1
            tmp.append(rand)              #initializing pieces with random cleanliness
            tmpv.append(0)
        visited.append(tmpv)
        maps.append(tmp)

    for i in range(row):
        tmp = []
        for j in range(col):
                if maps[i][j] == 0 :
                    tmp.append(Label(image = tile_clean ))
                else:
                    tmp.append(Label(image = tile_dirty ))
        lab1.append(tmp)

    for i in range(row):
            for j in range(col):
                lab1[i][j].grid(row=i,column=j)

def allowance(pos):
    if rows == 1 and cols != 1:    #horizontal movement
        allow = ['right','left']
    elif rows != 1 and cols == 1:  #vertical movement
        allow = ['up','down']
    elif rows == 1 and cols == 1:  #nop
        allow = []
    else:                          #two dimentional movement
        if pos[0] == '1' and pos[1] == '1':
            allow = ['right','down']
        elif pos[0] == '1' and pos[1] == str(cols):
            allow = ['left','down']
        elif pos[0] == str(rows) and pos[1] == str(cols):
            allow = ['left','up']
        elif pos[0] == str(rows) and pos[1] == '1':
            allow = ['right','up']
```

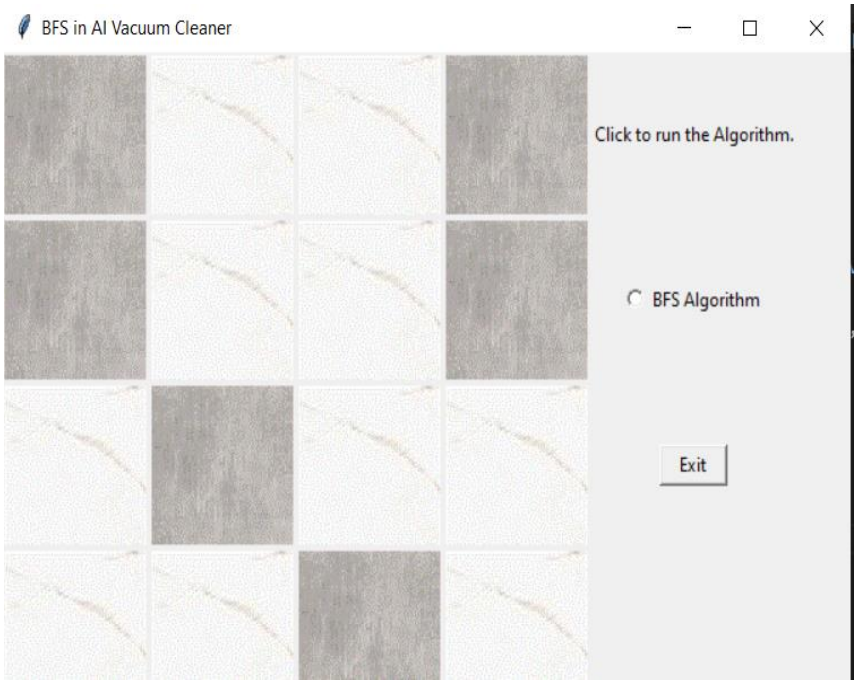
```
        elif pos[0] == '1' and (pos[1]!= '1' or pos[1]!=str(cols)):
            allow = ['right','left','down']
        elif pos[0] == str(rows) and (pos[1]!= '1' or pos[1]!= str(cols)) :
            allow = ['right','left','up']
        elif pos[1] == '1' and (pos[0]!= '1' or pos[1]!= str(rows)):
            allow = ['right','up','down']
        elif pos[1] == str(cols) and (pos[0]!= '1' or pos[1]!= str(rows)):
            allow = ['left','up','down']
        else:
            allow = ['right','left','up','down']
    return allow

#main
#initializing
maps = []
tmp_maps = []
visited = []
visited_pieces = 0
dirty_pieces = 0
cleaned_pieces = 0
rows =  4               #Row number
cols = 4               #Column number
vac_pos = [1,1]          #Current Cursor
vac_str = str(vac_pos[0]) + str(vac_pos[1])
mygui = Tk()
mygui.title("BFS in AI Vacuum Cleaner")
mygui.geometry("600x400")
ll = Label(mygui,text="Click to run the Algorithm.")
ll.grid(row = 0,column=cols)
lab1 = []
tile_clean = PhotoImage(file="Src\\tile2.gif")
tile_dirty = PhotoImage(file="Src\\tile1.gif")
done_icon = PhotoImage(file="Src\done_icon.png")
Vacuum = PhotoImage(file="Src\Vacuum.gif")
initmap(rows,cols)
mygui.update()
tmp_maps = maps[:]

print('-----------------------------------')
print('Current Cursor Location :' , vac_pos)
print('-----------------------------------')
print('env. before cleaning:')
#showing the whole env.
for i in range (rows):
    print (maps[i])

print('-----------------------------------')
#print('Steps and directions:')
```

```
#checking pieces
# Initializing a queue
qs = Queue(maxsize = 10000)
qd = Queue(maxsize = 10000)
sys.setrecursionlimit(10**6)

def rbfs(vacpos,fro_d):
    def find_if_exist(frd):
        for p in allow:
            if frd == p:
                return True
        return False
    global visited_pieces
    global visited
    global cleaned_pieces
    global dirty_pieces

    if (cleaned_pieces==dirty_pieces):
        print('-----------------------------------')
        print('Environment Total Pieces :' , rows*cols)
        print('visited_pieces : ', visited_pieces)
        print('dirty_pieces : ', dirty_pieces)
        print('cleaned_pieces : ', cleaned_pieces)
        #print('The Agent\'s Score : ', cleaned_pieces/visited_pieces)
        print('-----------------------------------')
        #showing the whole env. after cleaning
        print('env. after cleaning:')
        for i in range (rows):
            print (maps[i])
        return

    allow = allowance(vacpos)

    if (fro_d == 'right'):
        frd = 'left'
    elif (fro_d == 'left'):
        frd = 'right'
    elif (fro_d == 'up'):
        frd = 'down'
    elif (fro_d == 'down'):
        frd = 'up'
    if (fro_d!='None'):
        if (find_if_exist(frd) == True):
            allow.remove(frd)

    if (visited[int(vacpos[0])-1][int(vacpos[1])-1]==0):
            m=0
            for m in range(len(allow)):
```

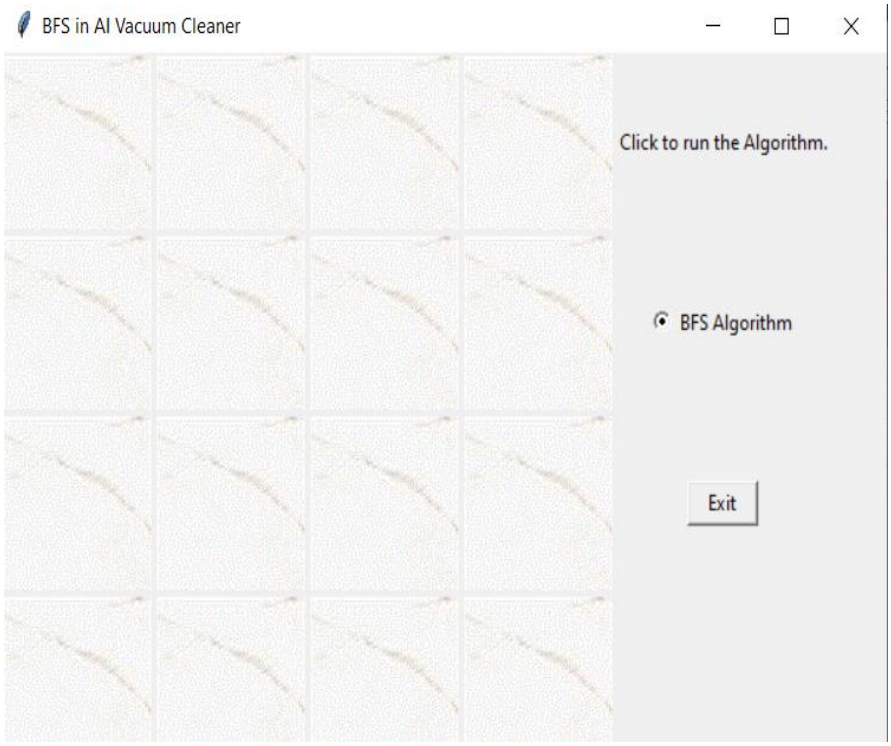
```
                addEdge(graph,vacpos,dirs(vacpos,allow[m],0))
                qs.put(dirs(vacpos,allow[m],0))
                qd.put(allow[m])

    visited_pieces = visited_pieces + 1
    visited[int(vacpos[0])-1][int(vacpos[1])-1] =  1
    lab1[int(vacpos[0])-1][int(vacpos[1])-1].config(image = Vacuum)
    mygui.update()
    sleep(1)
    lab1[int(vacpos[0])-1][int(vacpos[1])-1].config(image = tile_clean)
    sleep(0)

    if maps[int(vacpos[0])-1][int(vacpos[1])-1] == 1:
        maps[int(vacpos[0])-1][int(vacpos[1])-1] = 0
        lab1[int(vacpos[0])-1][int(vacpos[1])-1].config(image = done_icon)
        mygui.update()
        sleep(1)
        cleaned_pieces = cleaned_pieces + 1

    lab1[int(vacpos[0])-1][int(vacpos[1])-1].config(image = tile_clean)
    sleep(0)
    news = qs.get()
    newd = qd.get()

    #print(news,'\n')
    rbfs(news,newd)

def check():
        global cb
        cb = v.get()
        if (cb == 1):
                qs.put(vac_str)
                rbfs(vac_str,'None')
                print("Done")

def Close():
    mygui.destroy()
v = IntVar()
Radiobutton(mygui,text="BFS Algorithm",padx=10,variable =
v,value=1,command=check).grid(row = 1,column=cols)
# Button for closing
exit_button = Button(mygui,padx=10, text="Exit", command=Close)
exit_button.grid(row=2,column=cols)
mygui.mainloop()
```

Output:

Initial Room:



After Cleaning :



Program on DFS:

```
from tkinter import *
from time import *
```

```
import random
from collections import defaultdict
from queue import Queue
import sys
import gc

gc.collect()

# function for adding edge to graph
graph = defaultdict(list)
def addEdge(graph,u,v):
    graph[u].append(v)

# definition of function
def generate_edges(graph):
    edges = []

    # for each node in graph
    for node in graph:

        # for each neighbour node of a single node
        for neighbour in graph[node]:

            # if edge exists then append
            edges.append((node, neighbour))
    return edges


# direction includes right,left,up,down,suck,nop
def dirs(vacpos,act,flag): #flag 0 dont move vac
    global visited_pieces
    if act == 'right':
        if (flag==1):
            vac_pos[1] = vac_pos[1] + 1
        else:
            return vacpos[0]+str(int(vacpos[1]) + 1)
    elif act == 'left':
        if (flag==1):
            vac_pos[1] = vac_pos[1] - 1
        else:
            return vacpos[0]+str(int(vacpos[1]) - 1)
    elif act == 'up':
        if (flag==1):
            vac_pos[0] = vac_pos[0] - 1
        else:
            return str(int(vacpos[0])-1)+vacpos[1]

    elif act == 'down':
        if (flag==1):
```

```
            vac_pos[0] = vac_pos[0] + 1
        else:
            return str(int(vacpos[0])+1)+vacpos[1]

    if (flag==1):
        visited_pieces = visited_pieces + 1

def actuator(act):
    global cleaned_pieces
    if act == 'suck':
        maps[vac_pos[0]-1][vac_pos[1]-1] = 0
        cleaned_pieces = cleaned_pieces + 1

# '0' means clean & '1' means dirty
def initmap(row,col):
    global dirty_pieces
    global maps
    global visited
    for j in range(row):
        tmp = []
        tmpv = []
        for i in range(col):
            rand = random.randint(0,1)
            if rand == 1 :
                dirty_pieces = dirty_pieces + 1
            tmp.append(rand)               #initializing pieces with random
cleanliness
            tmpv.append(0)
        visited.append(tmpv)
        maps.append(tmp)

    for i in range(row):
        tmp = []
        for j in range(col):
                if maps[i][j] == 0 :
                    tmp.append(Label(image = tile_clean ))
                else:
                    tmp.append(Label(image = tile_dirty ))
        lab1.append(tmp)

    for i in range(row):
            for j in range(col):
                lab1[i][j].grid(row=i,column=j)

def allowance(pos):
    if rows == 1 and cols != 1:    #horizontal movement
        allow = ['right','left']
    elif rows != 1 and cols == 1:  #vertical movement
```

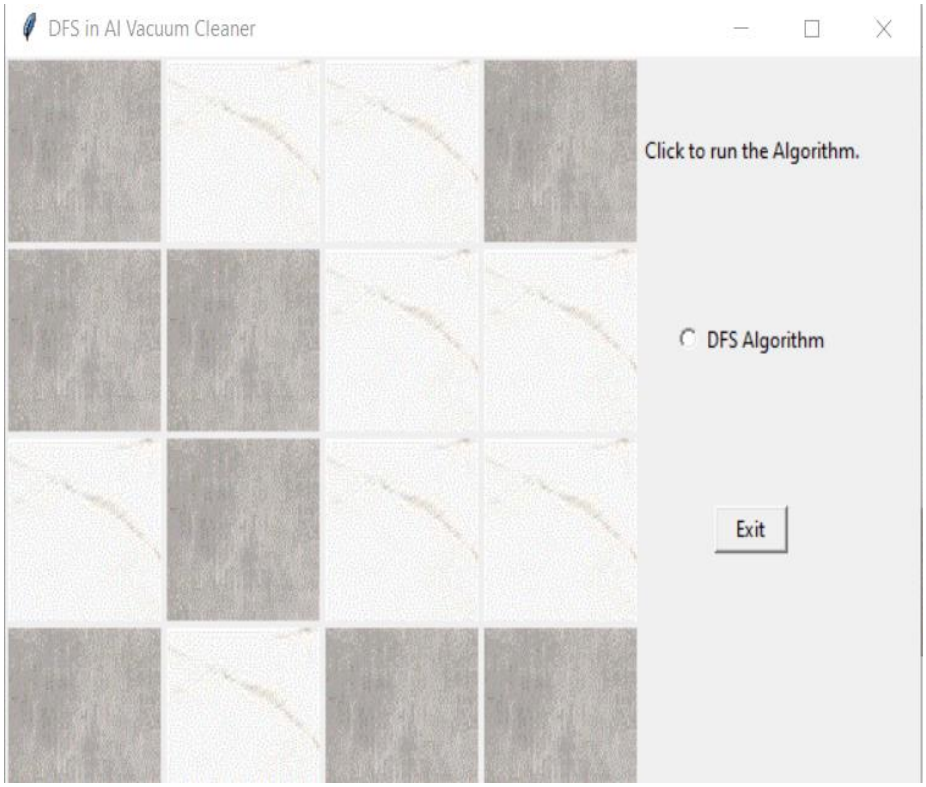
```
        allow = ['up','down']
    elif rows == 1 and cols == 1:  #nop
        allow = []
    else:                          #two dimentional movement
        if pos[0] == '1' and pos[1] == '1':
            allow = ['right','down']
        elif pos[0] == '1' and pos[1] == str(cols):
            allow = ['left','down']
        elif pos[0] == str(rows) and pos[1] == str(cols):
            allow = ['left','up']
        elif pos[0] == str(rows) and pos[1] == '1':
            allow = ['right','up']
        elif pos[0] == '1' and (pos[1]!= '1' or pos[1]!=str(cols)):
            allow = ['right','left','down']
        elif pos[0] == str(rows) and (pos[1]!= '1' or pos[1]!= str(cols)) :
            allow = ['right','left','up']
        elif pos[1] == '1' and (pos[0]!= '1' or pos[1]!= str(rows)):
            allow = ['right','up','down']
        elif pos[1] == str(cols) and (pos[0]!= '1' or pos[1]!= str(rows)):
            allow = ['left','up','down']
        else:
            allow = ['right','left','up','down']
    return allow

#main

#initializing
maps = []
tmp_maps = []
visited = []
visited_pieces = 0
dirty_pieces = 0
cleaned_pieces = 0
rows =  4               #Row number
cols = 4                #Column number
vac_pos = [1,1]           #Current Cursor
vac_str = str(vac_pos[0]) + str(vac_pos[1])
mygui = Tk()
mygui.title("DFS in AI Vacuum Cleaner")
mygui.geometry("600x400")
ll = Label(mygui,text="Click to run the Algorithm.")
ll.grid(row = 0,column=cols)
lab1 = []
tile_clean = PhotoImage(file="Src\\tile2.gif")
tile_dirty = PhotoImage(file="Src\\tile1.gif")
done_icon = PhotoImage(file="Src\done_icon.png")
Vacuum = PhotoImage(file="Src\Vacuum.gif")
initmap(rows,cols)
```

```
mygui.update()
tmp_maps = maps[:]

print('------------------------------------')
print('Current Cursor Location :' , vac_pos)
print('------------------------------------')
print('env. before cleaning:')
#showing the whole env.
for i in range (rows):
    print (maps[i])

print('------------------------------------')



stacks = []
stackd = []
def rdfs(vacpos,fro_d):
    def find_if_exist(frd):
        for p in allow:
            if frd == p:
                return True
        return False
    global visited_pieces
    global cleaned_pieces
    global dirty_pieces
    global visited

    if (cleaned_pieces==dirty_pieces):
        print('------------------------------------')
        print('Environment Total Pieces :' , rows*cols)
        print('visited_pieces : ', visited_pieces)
        print('dirty_pieces : ', dirty_pieces)
        print('cleaned_pieces : ', cleaned_pieces)

        print('------------------------------------')
        #showing the whole env. after cleaning
        print('env. after cleaning:')
        for i in range (rows):
            print (maps[i])

        return

    allow = allowance(vacpos)

    if (fro_d == 'right'):
        frd = 'left'
    elif (fro_d == 'left'):
        frd = 'right'
```

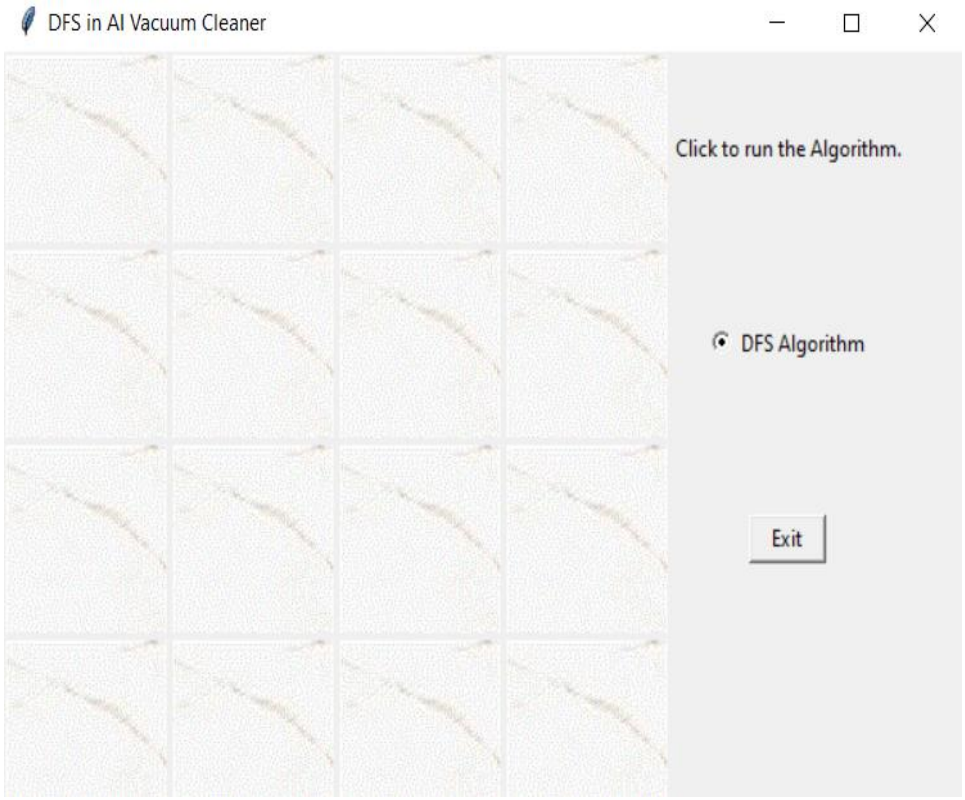
```
    elif (fro_d == 'up'):
        frd = 'down'
    elif (fro_d == 'down'):
        frd = 'up'
    if (fro_d!='None'):
        if (find_if_exist(frd) == True):
            allow.remove(frd)

    if (visited[int(vacpos[0])-1][int(vacpos[1])-1]==0):
            m=0
            for m in range(len(allow)):
                addEdge(graph,vacpos,dirs(vacpos,allow[m],0))
                stacks.append(dirs(vacpos,allow[m],0))
                stackd.append(allow[m])

    lab1[int(vacpos[0])-1][int(vacpos[1])-1].config(image = Vacuum)
    mygui.update()
    sleep(1)
    lab1[int(vacpos[0])-1][int(vacpos[1])-1].config(image = tile_clean)
    sleep(0)

    visited_pieces = visited_pieces + 1
    visited[int(vacpos[0])-1][int(vacpos[1])-1] = 1
    if maps[int(vacpos[0])-1][int(vacpos[1])-1] == 1:
        maps[int(vacpos[0])-1][int(vacpos[1])-1] = 0
        lab1[int(vacpos[0])-1][int(vacpos[1])-1].config(image = done_icon)
        mygui.update()
        sleep(1)
        cleaned_pieces = cleaned_pieces + 1

    lab1[int(vacpos[0])-1][int(vacpos[1])-1].config(image = tile_clean)
    sleep(0)
    news = stacks.pop()
    newd = stackd.pop()
    #print(news,'\n')
    rdfs(news,newd)

def check():
        global cb
        cb = v.get()
        if (cb == 1):
                stacks.append(vac_str)
                rdfs(vac_str,'None')
                print("Done!!")


def Close():
    mygui.destroy()
v = IntVar()
```

```
Radiobutton(mygui,text="DFS Algorithm",padx=10,variable =
v,value=1,command=check).grid(row = 1,column=cols)
# Button for closing
exit_button = Button(mygui,padx=10, text="Exit", command=Close)
exit_button.grid(row=2,column=cols)
mygui.mainloop()
```

Initial Room:



After Cleaning:




```
------------------------------------
Current Cursor Location : [1, 1]
------------------------------------
env. before cleaning:
[1, 0, 0, 1]
[1, 1, 0, 0]
[0, 1, 0, 0]
[1, 0, 1, 1]
------------------------------------
------------------------------------
Environment Total Pieces : 16
visited_pieces :  18
dirty_pieces :  8
cleaned_pieces :  8
------------------------------------
env. after cleaning:
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
Done!!
```

Conclusion :

We were able to implement BFS and DFS in Intelligent agent. In that the environment for this is a floor which is divided into n*n tiles . The floor is randomly generated and some tiles are clean while some of them are dirty . So the bot will sense the status of the tiles using sensors and determine whether to suck or leave it as it is. In BFS it will use queue data structure and search breadth -wise . Similarly In DFS it will use stack data structure and search using depth-wise. Thus from this experiment we have learnt the efficiency of BFS and DFS algorithm and implemented it successfully using a vacuum cleaner bot .