P18-0126 Ahmad Hasan Syed

initially state:A
Goal State:all locations are clean (Back)
Operators :diff functions have (0,1) used as operator in dirt cal fun in location check (0-9)

(Bfs and DFS along with vacuum cleaner agent are implemented in same code!)
Code :

```
class Binary_Search_Tree:


    def __init__(self, data):
        self.data = data
        self.Left_child = None
        self.Right_child = None


    def Add_Node(self, data):
        if data == self.data:
            return # node already exist


        if data < self.data:
            if self.Left_child:
                self.Left_child.Add_Node(data)
            else:
                self.Left_child = Binary_Search_Tree(data)


        else:
            if self.Right_child:
                self.Right_child.Add_Node(data)
            else:
                self.Right_child = Binary_Search_Tree(data)

    def bfs(self):
        elements = []
        path_cost=0
        if self.Left_child:
            elements += self.Left_child.bfs()
            path_cost=path_cost+1

        elements.append(self.data)
        path_cost=path_cost+1

        if self.Right_child:
            elements += self.Right_child.bfs()
            path_cost=path_cost+1

        print(path_cost) #each element traversed and added to queue will be counted as 1 cost of path

        return elements
    def dfs(self):
        elements = [self.data]
        if self.Left_child:
            elements += self.Left_child.dfs()
        if self.Right_child:
            elements += self.Right_child.dfs()

        return elements
```

```
#vacum cleaner code
env_table={"Location A 0":"vacum at location A and location is dirty",
           "Location A 1":"vacum at location A and location is clean",
           "Location B 0":"vacum at location B and location is dirty",
           "Location B 1":"vacum at location B and location is clean",
           "Location C 0":"vacum at location C and location is dirty",
           "Location C 1":"vacum at location C and location is clean",
           "Location D 0":"vacum at location D and location is dirty",
           "Location D 1":"vacum at location D and location is clean",
           "Location E 0":"vacum at location E and location is dirty",
           "Location E 1":"vacum at location E and location is clean",
           "Location F 0":"vacum at location F and location is dirty",
           "Location F 1":"vacum at location F and location is clean",
           "Location G 0":"vacum at location G and location is dirty",
           "Location G 1":"vacum at location G and location is clean",
           "Location H 0":"vacum at location H and location is dirty",
           "Location H 1":"vacum at location H and location is clean",
           "Location I 0":"vacum at location I and location is dirty",
           "Location I 1":"vacum at location I and location is clean",
           "Back":"All locations are clean"
           }
def location_check():
        n=0 #here we mean 0 is A
        b=Binary_Search_Tree(n)
        b.Add_Node(n)#setting initial state at A
        q=9
        for i in range(q):
            n = np.random.uniform(low=1.0, high=9.0) #starting from B
            b.insert(n)
        c=b.bfs() #by this we will search the location using bfs in tree
        return c

def dirt_clean_cal():
    num=1
    for i in range(num):  #for loop for flexibility so that if new state to be added
        num = np.random.uniform(low=0.0, high=1.0)
        if num > 0.2:
            return 0           # prob of 0.2 of dirt if prob>0.2 return 0(location is dirty!)
        else:
            return 1  #else location is clean
```

P18-0126 Ahmad Hasan Syed

```
def main_fun():
    x=location_check()
    s=dirt_clean_cal()
    if x==0:
        if s==0:
            print(env_table["Location A 0"])
        else:
            print(env_table["Location A 1"])

    if x==1:
        if s==0:
            print(env_table["Location B 0"])
        else:
            print(env_table["Location B 1"])

    if x==2:
        if s==0:
            print(env_table["Location C 0"])
        else:
            print(env_table["Location C 1"])
    if x==3:
        if s==0:
            print(env_table["Location D 0"])
        else:
            print(env_table["Location D 1"])
    if x==4:
        if s==0:
            print(env_table["Location E 0"])
        else:
            print(env_table["Location E 1"])
    if x==5:
        if s==0:
            print(env_table["Location F 0"])
        else:
            print(env_table["Location F 1"])
    if x==6:
        if s==0:
            print(env_table["Location G 0"])
        else:
            print(env_table["Location G 1"])
    if x==7:
        if s==0:
            print(env_table["Location H 0"])
        else:
            print(env_table["Location H 1"])
    if x==8:
        if s==0:
            print(env_table["Location I 0"])
        else:
            print(env_table["Location I 1"])
    else:
        print(env_table["Back"])

    return
```

P18-0126 Ahmad Hasan Syed

If the environment is increased to nxn the performance measure won’t be affected as much as it would be affected in simple reflex agent ! (The function is using bfs so it won’t take long enough to sort nxn environment)