```
import tkinter as tk

import time

from pyswip import Prolog

prolog = Prolog()

from pyswip import Functor, Variable, Query, call


prolog.consult("wumpusLogic.pl")


frameGap=0.5 #seconds


openNodes=[]

knownBreeze=[]

knownStench=[]


rows=4

cols=4


pitLocations=[2,10,15]

wumpusLocations=[8]

goldLocations=[9]


def adjecent(pos):

        result=[]

        if ((pos-cols)>=0):

                result.append(pos-cols)

        if ((pos+cols)<rows*cols):

                result.append(pos+cols)

        if (pos%cols!=0):

                result.append(pos-1)
```

```
    if ((pos+1)%cols!=0):
        result.append(pos+1)
    return result

def adjVect(locs):
    result=[]
    for L in locs:
        adj=adjecent(L)
        for p in adj:
            if not(p in result):
                result.append(p)
    return result

labels=[]

# To move without game over for test purposes
def move2(curr, end):
    adjPit=adjVect(pitLocations)
    adjWumpus=adjVect(wumpusLocations)
    time.sleep(frameGap)
    if (curr in wumpusLocations):
        labels[curr]["image"]=wumpus
    elif (curr in pitLocations):
        labels[curr]["image"]=pit
    elif (curr in goldLocations):
        if ((curr in adjPit) and (curr in adjWumpus)):
            labels[curr]["image"]=gdanger
        elif (curr in adjWumpus):
            labels[curr]["image"]=gstench
        elif (curr in adjPit):
            labels[curr]["image"]=gbreeze
        else:
            labels[curr]["image"]=gold
    elif ((curr in adjPit) and (curr in adjWumpus)):
        labels[curr]["image"]=danger
    elif (curr in adjWumpus):
        labels[curr]["image"]=stench
    elif (curr in adjPit):
        labels[curr]["image"]=breeze
    else:
        labels[curr]["image"]=visited

    if (end in wumpusLocations):
        labels[end]["image"]=wumpus
    elif (end in pitLocations):
        labels[end]["image"]=pit
    elif (end in goldLocations):
        labels[end]["image"]=gold
    elif ((end in adjPit) and (curr in adjWumpus)):
        labels[end]["image"]=adanger
        knownStench.append(end)
        knownBreeze.append(end)
    elif (end in adjWumpus):
        labels[end]["image"]=astench
        knownStench.append(end)
    elif (end in adjPit):
        labels[end]["image"]=abreeze
        knownBreeze.append(end)
    else:
        labels[end]["image"]=agent
    if not(end in openNodes):
        openNodes.append(end)
    root.update()
    return end



# To move to adjecent squres of the agent
def move(curr, end):
    if ((not((curr in pitLocations) or (curr in wumpusLocations) or (curr in goldLocations))) and (end in adjVect([curr]))):
        adjPit=adjVect(pitLocations)
        adjWumpus=adjVect(wumpusLocations)
        time.sleep(frameGap)
        if ((curr in adjPit) and (curr in adjWumpus)):
            labels[curr]["image"]=danger
        elif (curr in adjWumpus):
            labels[curr]["image"]=stench
        elif (curr in adjPit):
            labels[curr]["image"]=breeze
        else:
            labels[curr]["image"]=visited

        if (end in wumpusLocations):
            labels[end]["image"]=wumpus
        elif (end in pitLocations):
            labels[end]["image"]=pit
        elif (end in goldLocations):
            labels[end]["image"]=gold
        elif ((end in adjPit) and (curr in adjWumpus)):
            labels[end]["image"]=adanger
            knownStench.append(end)
            knownBreeze.append(end)
        elif (end in adjWumpus):
            labels[end]["image"]=astench
            knownStench.append(end)
        elif (end in adjPit):
            labels[end]["image"]=abreeze
            knownBreeze.append(end)
        else:
            labels[end]["image"]=agent
        if not(end in openNodes):
            openNodes.append(end)
        root.update()
        return end
    else:
        return curr

# For backtracking search, history is considered to avoind revisiting sqares
def succAndCost(pos,dest,hist):
    result=[]
    if ((((pos-cols) in openNodes) or ((pos-cols) == dest)) and (not((pos-cols) in hist))):
        result.append((pos-cols,1))
    if ((((pos+cols) in openNodes) or ((pos+cols) == dest)) and (not((pos+cols) in hist))):
        result.append((pos+cols,1))
    if ((((pos-1) in openNodes) or ((pos-1) == dest)) and (not((pos-1) in hist))):
        result.append((pos-1,1))
    if ((((pos+1) in openNodes) or ((pos+1) == dest)) and (not((pos+1) in hist))):
        result.append((pos+1,1))
    return result

# To find optimum path from agent to a propoed safe point
def backtrackingSearch(position,dest):
    # dictionary to hold cost
    best={
        'cost': float('inf'),
        'history': None
    }
    # recursively search down the tree
    def recurse (state, destination, history, totalCost):
        # to be an end state it has to pass goal test and have the least cost
        if (state==destination):
            if totalCost<best['cost']:
                best['cost']=totalCost
                best['history']=history
            return
        # if not end state recurse down lower branches
        for newState, cost in succAndCost(state,dest,history):
            recurse(newState, destination, history+[newState], totalCost+cost)
    recurse (position, dest, history=[], totalCost=0)
    return best['history']

# To move agent  from current location to a propoed safe point with the help of backtracking search
def goTo(curr, dest):
    histr=backtrackingSearch(curr,dest)
    for k in range(len(histr)):
```

```
                    curr=move(curr, histr[k])
          return curr


# GUI and images for it
root=tk.Tk()
agent = tk.PhotoImage(file="agent.png")
gold = tk.PhotoImage(file="gold.png")
visited = tk.PhotoImage(file="visited.png")
unvisited = tk.PhotoImage(file="unvisited.png")
wumpus = tk.PhotoImage(file="wumpus.png")
stench = tk.PhotoImage(file="stench.png")
breeze = tk.PhotoImage(file="breeze.png")
danger = tk.PhotoImage(file="danger.png")
pit = tk.PhotoImage(file="pit.png")
astench = tk.PhotoImage(file="astench.png")
abreeze = tk.PhotoImage(file="abreeze.png")
adanger = tk.PhotoImage(file="adanger.png")
gstench = tk.PhotoImage(file="gstench.png")
gbreeze = tk.PhotoImage(file="gbreeze.png")
gdanger = tk.PhotoImage(file="gdanger.png")


# Start state in GUI
for r in range(rows):
          for c in range(cols):
                    box=tk.Label(root, image=unvisited)
                    box.grid(row=r, column=c)
                    labels.append(box)


root.update()
```

```
labels[0]["image"]=agent
openNodes.append(0)
current=0

# A-B
def setSubstact(A, B):
    result=[]
    for a in A:
        if not(a in B):
            result.append(a)
    return result


# for direct motion without finding safe path test is used
# test=[1,5,4,6,7,11,7,3,7,6,5,9,13,12,13,14,13]

# for k in range(len(test)):
#   current=move2(current, test[k])

#To see board

# for k in range(rows*cols):
#   current=move2(current, k)
# current=move2(current, 0)


# until end of game
while not((current in pitLocations) or (current in wumpusLocations) or (current in goldLocations)):
    # The following lists can be given to prologe as input
    #knownBreeze
    #knownStench
    adjKnownBreeze=adjVect(knownBreeze)
    adjKnownStench=adjVect(knownStench)
    knownNotBreeze=setSubstact(openNodes,knownBreeze)
    knownNotStench=setSubstact(openNodes,knownStench)
    adjKnownNotBreeze=adjVect(knownNotBreeze)
    adjKnownNotStench=adjVect(knownNotStench)
    nextNodes=setSubstact(adjVect(openNodes),openNodes)

    Array1=[0]
    Array2=[0]
    #prolog logic to choose a next node, choose first solution and break
    for soln in prolog.query("nextNodeToGo("+str(Array1)+","+str(Array2)+",X)"):
        nextNode=soln["X"]
        # print(nextNode)
        break
    current=goTo(current,nextNode)

root.mainloop()
```