Date: **Ex No:** 2.1	**Title of the Lab** Wumpus World	**Name:** Yuvraj Singh Chauhan **Registration Number:** RA1911027010058 **Section:** N1 **Lab Batch:** 1 **Day Order:** 3

AIM:

To implement the wumpus world problem in python.

Description of the Concept or Problem given:

The Wumpus world is a cave with 16 rooms (4×4). Each room is connected to others through walkways (no rooms are connected diagonally). The knowledge-based agent starts from Room[1, 1]. The cave has – some pits, a treasure and a beast named Wumpus. The Wumpus can not move but eats the one who enters its room. If the agent enters the pit, it gets stuck there. The goal of the agent is to take the treasure and come out of the cave. The agent is rewarded, when the goal conditions are met. The agent is penalized, when it falls into a pit or being eaten by the Wumpus.

Heuristics used : Early termination, Pure symbols, Unit clauses

Manual Solution:

i. Start from 1,1 and maintain a visited and to_explore sets and a list to mark safe tiles.

ii. Check for stench and breeze and add it to the knowledge base.

iii. Check if there is no wumpus / pit in the adjacent rooms by using dpll algorithm and add corresponding sentence to kb if both are not present and update to_explore set and also update the list with safe tiles.

iv. If we can’t say for sure that the adjacent room is safe from both wumpus and pit, check for the possibility that there is wumpus and pit.

v. Explore new tiles from to_explore set.

vi. Keep exploring the to_explore tiles using bfs algorithm to go from one safe tile to another.

vii. If the tile is 4,4 exit.

viii. If the to_explore set is empty, backtrack to 1,1 and start exploring again.

```
['','','P',''], # Rooms [1,1] to [4,1]
['','','',''], # Rooms [1,2] to [4,2]
['W','','',''], # Rooms [1,3] to [4,3]
['','','',''], # Rooms [1,4] to [4,4]
```

This shows how effective unit clause is in case of our problem which has a lot of unit clauses as we progress deep into the level. This skewed result might also be due to the face that unit clause heuristic in this case also detects failure (p,~p) in this case.

Program Implementation [Coding]

```
"""
Agent
"""

class Agent:
    def __init__(self):
        self.__wumpusWorld = [
                 ['','','P',''], # Rooms [1,1] to [4,1]
                 ['','','',''], # Rooms [1,2] to [4,2]
                 ['W','','',''], # Rooms [1,3] to [4,3]
                 ['','','',''], # Rooms [1,4] to [4,4]
                ] # This is the wumpus world shown in the assignment question.
                  # A different instance of the wumpus world will be used for evaluation.
        self._curLoc = [1,1]
        self.__isAlive = True
        self._hasExited = False

    def __FindIndicesForLocation(self,loc):
        x,y = loc
        i,j = y-1, x-1
        return i,j
    def __CheckForPitWumpus(self):
        ww = self._wumpusWorld
        i,j = self.__FindIndicesForLocation(self.__curLoc)
        if 'P' in ww[i][j] or 'W' in ww[i][j]:
            print(ww[i][j])
            self._isAlive = False
            print('Agent is DEAD.')
        return self.__isAlive

    def TakeAction(self,action): # The function takes an action and returns whether the Agent
is alive
                                # after taking the action.
        validActions = ['Up','Down','Left','Right']
        assert action in validActions, 'Invalid Action.'
        if self._isAlive == False:
            print('Action cannot be performed. Agent is DEAD.
Location:{0}'.format(self._curLoc))
            return False
        if self.__hasExited == True:
            print('Action cannot be performed. Agent has exited the Wumpus
world.'.format(self._curLoc))
            return False

        index = validActions.index(action)
        validMoves = [[0,1],[0,-1],[-1,0],[1,0]]
        move = validMoves[index]
        newLoc = []
        for v, inc in zip(self.__curLoc,move):
            z = v + inc #increment location index
            z = 4 if z>4 else 1 if z<1 else z #Ensure that index is between 1 and 4
            newLoc.append(z)
        self._curLoc = newLoc
        print('Action Taken: {0}, Current Location {1}'.format(action,self._curLoc))
        if self._curLoc[0]==4 and self._curLoc[1]==4:
            self._hasExited=True
        return self._CheckForPitWumpus()

    def __FindAdjacentRooms(self):
        cLoc = self._curLoc
        validMoves = [[0,1],[0,-1],[-1,0],[1,0]]
        adjRooms = []
        for vM in validMoves:
            room = []
            valid = True
            for v, inc in zip(cLoc,vM):
                z = v + inc
                if z<1 or z>4:
                    valid = False
                    break
                else:
                    room.append(z)
            if valid==True:
                adjRooms.append(room)
        return adjRooms



    def PerceiveCurrentLocation(self): #This function perceives the current location.
                                    #It tells whether breeze and stench are present in the current
location.
```

```
        breeze, stench = False, False
        ww = self.__wumpusWorld
        if self._isAlive == False:
            print('Agent cannot perceive. Agent is DEAD. Location:{0}'.format(self._curLoc))
            return [None,None]
        if self.__hasExited == True:
            print('Agent cannot perceive. Agent has exited the Wumpus
World.'.format(self._curLoc))
            return [None,None]

        adjRooms = self._FindAdjacentRooms()
        for room in adjRooms:
            i,j = self._FindIndicesForLocation(room)
            if 'P' in ww[i][j]:
                breeze = True
            if 'W' in ww[i][j]:
                stench = True
        return [breeze,stench]

    def FindCurrentLocation(self):
        return self._curLoc

def main():
    ag = Agent()
    print('curLoc',ag.FindCurrentLocation())
    print('Percept [breeze, stench] :',ag.PerceiveCurrentLocation())
    ag.TakeAction('Right')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Right')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Right')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Up')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Up')
    print('Percept',ag.PerceiveCurrentLocation())
    ag.TakeAction('Up')
    print('Percept',ag.PerceiveCurrentLocation())



if __name_=='_main_':
    main()



"""
Main Program
"""
from Agent import *
import copy
numberOfCalls=0

class KnowledgeBase:
    #clause[i]=-1 represents the presence of negative literal represented by i
    #clause[i]=1 represents the presence of positive literal represented by i
    #values 0 to 15 represents W(1,1) to W(4,4)
    #values 16 to 31 represents S(1,1) to S(4,4)
    #values 33 to 47 represents P(1,1) to P(4,4)
    #values 48 to 63 represents B(1,1) to B(4,4)
    def_init_(self):
```

```
self.clauses= []

#clauses for atleast 1 Wumpus and 1 Pit
atleast1Wumpus= {}
atleast1Pit = {}
for i in range (16):
   atleast1Wumpus[i]=1
   atleast1Pit[i+32]=1
self.clauses.append(atleast1Wumpus)
self.clauses.append(atleast1Pit)

#clauses for atmost 1 Wumpus and 1 Pit
for i in range(16):
   for j in range(i+1, 16):
      atmost1Wumpus={}
      atmost1Pit={}
      atmost1Wumpus[i]=-1
      atmost1Wumpus[j]=-1
      atmost1Pit[i+32]=-1
      atmost1Pit[j+32]=-1
      self.clauses.append(atmost1Wumpus)
      self.clauses.append(atmost1Pit)

#Stench-Wumpus bijection clauses
for i in range(16):
   stenchWumpusClause={}
   stenchWumpusClause[i+16]=-1
   if (i+4)//4 < 4:
      stenchWumpusClause[i+4]=1
```

```
            stenchClause={}
            stenchClause[i+16]=1
            stenchClause[i+4]=-1
            self.clauses.append(stenchClause)
        if(i-4)//4 >= 0:
            stenchWumpusClause[i-4]=1
            stenchClause={}
            stenchClause[i+16]=1
            stenchClause[i-4]=-1
            self.clauses.append(stenchClause)
        if i//4 == (i+1)//4:
            stenchWumpusClause[i+1]=1
            stenchClause={}
            stenchClause[i+16]=1
            stenchClause[i+1]=-1
            self.clauses.append(stenchClause)
        if i//4 == (i-1)//4:
            stenchWumpusClause[i-1]=1
            stenchClause={}
            stenchClause[i+16]=1
            stenchClause[i-1]=-1
            self.clauses.append(stenchClause)
        self.clauses.append(stenchWumpusClause)

    #Breeze-Pit Bijection Clauses
    for i in range(16):
        breezePitClause={}
        breezePitClause[i+48]=-1
        if(i+4)//4 < 4:
```

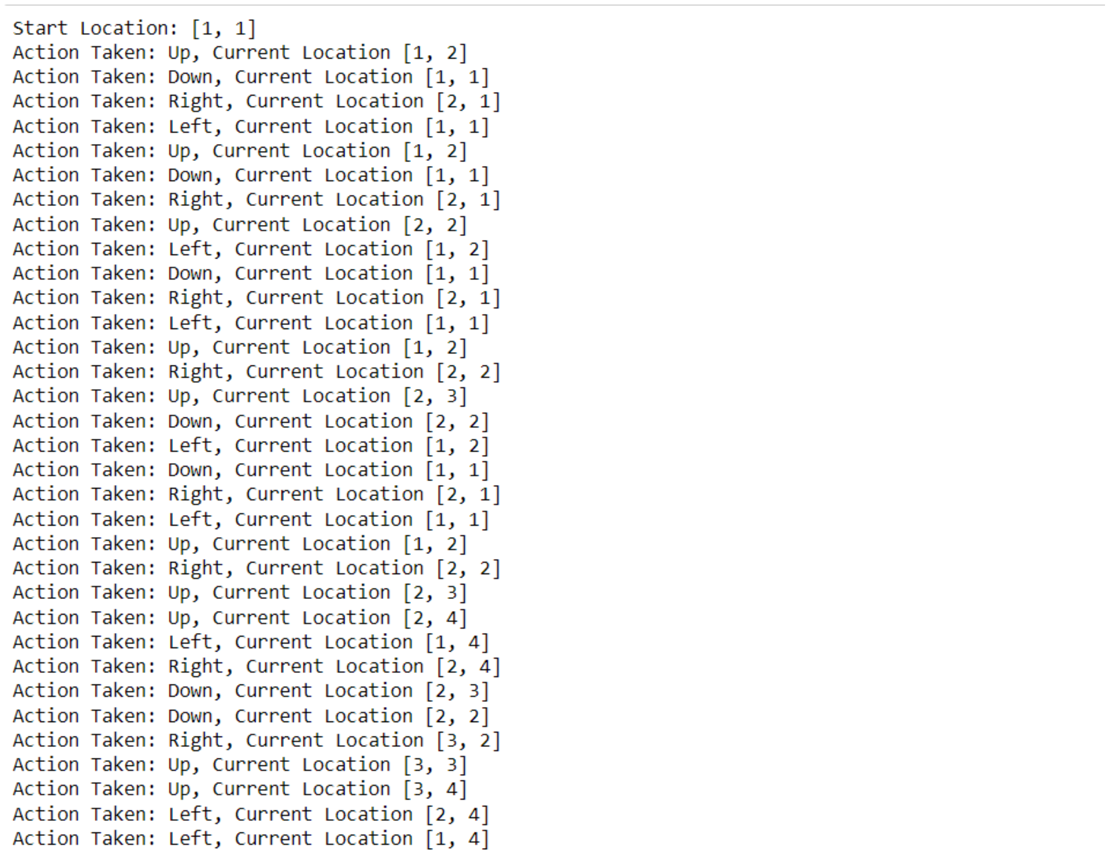
```
        breezePitClause[i+4+32]=1
        pitClause={}
        pitClause[i+48]=1
        pitClause[i+4+32]=-1
        self.clauses.append(pitClause)
    if(i-4)//4 >= 0:
        breezePitClause[i-4+32]=1
        pitClause={}
        pitClause[i+48]=1
        pitClause[i-4+32]=-1
        self.clauses.append(pitClause)
    if i//4 == (i+1)//4:
        breezePitClause[i+1+32]=1
        pitClause={}
        pitClause[i+48]=1
        pitClause[i+1+32]=-1
        self.clauses.append(pitClause)
    if i//4 == (i-1)//4:
        breezePitClause[i-1+32]=1
        pitClause={}
        pitClause[i+48]=1
        pitClause[i-1+32]=-1
        self.clauses.append(pitClause)
    self.clauses.append(breezePitClause)

#No wumpus and pit at [1, 1]
noWumpusStart={0:-1}
noPitStart={32:-1}
self.clauses.append(noWumpusStart)
```

```
        self.clauses.append(noPitStart)

    def AddClause(self, clause): #adding a clause to knowledge base
        self.clauses.append(clause)

    def getclauses(self): #return Wumpus clauses
        return copy.deepcopy(self.clauses)



def FindPureSymbol(clauses, symbols):
    for symbol in symbols:
        positive=0
        negative=0
        for clause in clauses:
            if symbol in clause:
                if clause[symbol]==1:
                    positive= positive+1
                else:
                    negative= negative+1
        if negative==0:
            return symbol, 1
        elif positive==0:
            return symbol, -1
    return -1, 0

def FindUnitClause(clauses):
    for clause in clauses:
        if len(clause)==1:
            for symbol in clause:
```

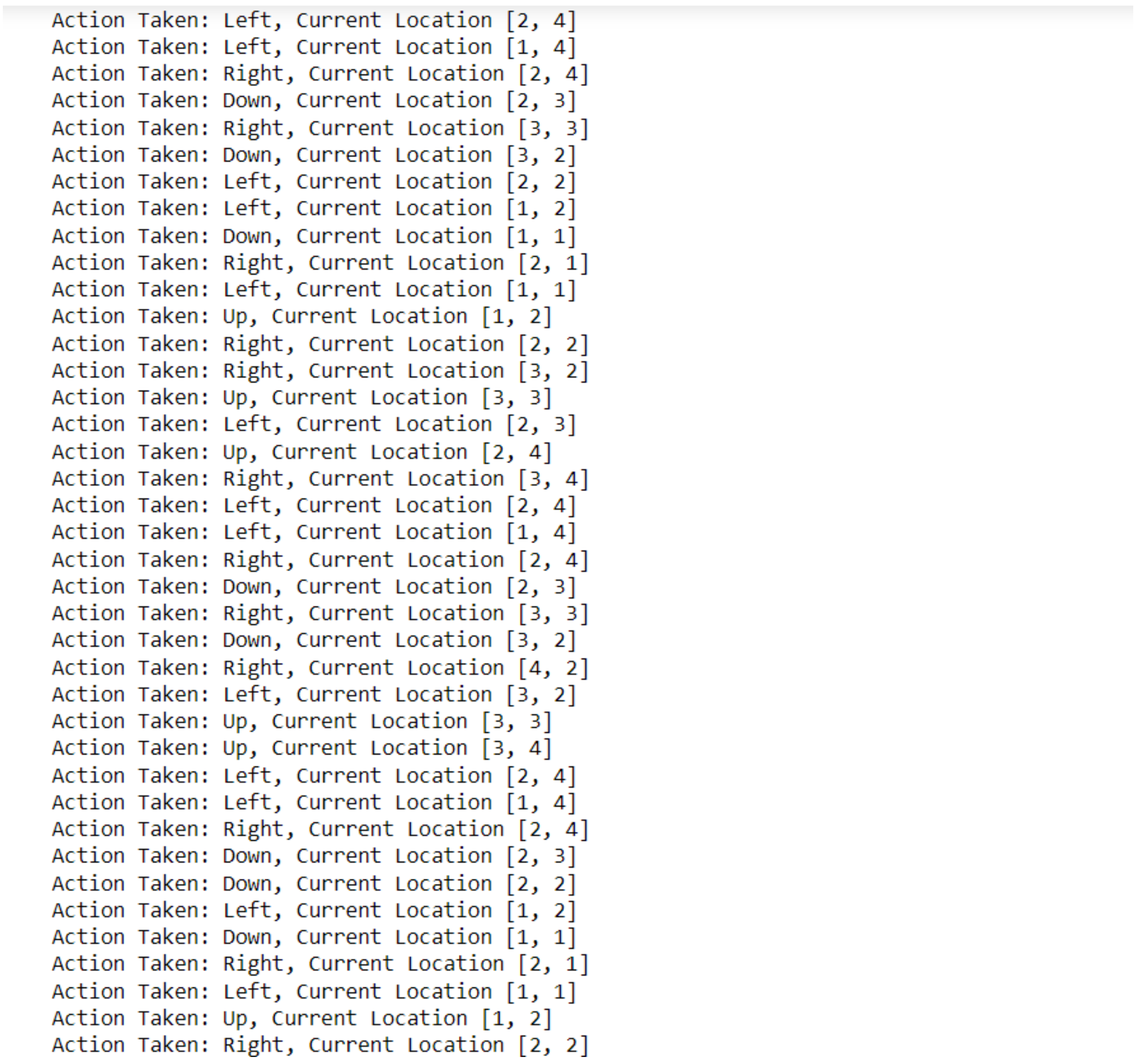
```
            return symbol, clause[symbol]
    return -1, 0

def selectSymbol(clauses, symbols):
    count={}
    positive={}
    negative={}
    for clause in clauses:
        for literal in clause:
            if literal not in count:
                count[literal]=0
                positive[literal]=0
                negative[literal]=0

            count[literal]= count[literal]+1
            if clause[literal]==1:
                positive[literal]=positive[literal]+1
            else:
                negative[literal]=negative[literal]+1

    maxLiteral= list(symbols.keys())[0]
    maxCount=0
    for literal in count:
        if count[literal]>maxCount:
            maxLiteral= literal
            maxCount= count[literal]

    if positive[maxLiteral]>negative[maxLiteral]:
        return maxLiteral, 1
    return maxLiteral, -1

def DPLL(clauses, symbols, model):
    global numberOfCalls
    numberOfCalls= numberOfCalls+1
    removeClauses=[]
    for clause in clauses:
        valueUnknown=True
        deleteLiterals=[]
        for literal in clause.keys():
            if literal in model.keys():
                if model[literal]==clause[literal]: #clause is true
                    removeClauses.append(clause)
                    valueUnknown=False
                    break
                else:
                    deleteLiterals.append(literal)

        for literal in deleteLiterals:
            del clause[literal]
        if valueUnknown==True and not bool(clause): #clause is false
            return False

    clauses= [ x for x in clauses if x not in removeClauses]

    if len(clauses)==0: #all clauses are true
        return True

    pureSymbol, value = FindPureSymbol(clauses, symbols)
    if value!=0:
        del symbols[pureSymbol]
        model[pureSymbol]=value
        return DPLL(clauses, symbols, model)

    unitSymbol, value = FindUnitClause(clauses)
    if value!=0:
        del symbols[unitSymbol]
        model[unitSymbol]=value
        return DPLL(clauses, symbols, model)

    symbol, value= selectSymbol(clauses, symbols)
    del symbols[symbol]
    model[symbol]= value

    if DPLL(copy.deepcopy(clauses), copy.deepcopy(symbols), copy.deepcopy(model)):
        return True

    model[symbol]= -value
    return DPLL(clauses, symbols, model)


def DPLLSatisfiable(clauses):
    symbols={}
    for clause in clauses:
        for literal in clause:
            symbols[literal]=True

    model={}
    return DPLL(clauses, symbols, model)


def MoveToUnvisited(ag, visited, goalLoc, dfsVisited): #dfs to new safe room
    curPos=ag.FindCurrentLocation()
    curLoc= 4*(curPos[0]-1)+curPos[1]-1
    if(curLoc==goalLoc):
        return True
    dfsVisited[curLoc]=True


    if curPos[1]+1 <=4 and (visited[curLoc+1]==True or (curLoc+1)==goalLoc) and 
dfsVisited[curLoc+1]==False:
        ag.TakeAction('Up')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
            return True
        ag.TakeAction('Down')


    if curPos[0]+1 <=4 and (visited[curLoc+4]==True or (curLoc+4)==goalLoc) and 
dfsVisited[curLoc+4]==False:
        ag.TakeAction('Right')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
            return True
        ag.TakeAction('Left')


    if curPos[0]-1 >0 and (visited[curLoc-4]==True or (curLoc-4)==goalLoc) and 
dfsVisited[curLoc-4]==False:
        ag.TakeAction('Left')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
            return True
        ag.TakeAction('Right')

    if curPos[1]-1 >0 and (visited[curLoc-1]==True or (curLoc-1)==goalLoc) and
dfsVisited[curLoc-1]==False:
        ag.TakeAction('Down')
        roomReachable= MoveToUnvisited(ag, visited, goalLoc, dfsVisited)
        if roomReachable:
            return True
        ag.TakeAction('Up')

    return False

def ExitWumpusWorld(ag, kb):
    visited = [False for i in range(16)] #Rooms Visited till now
    while(ag.FindCurrentLocation()!=[4, 4]):
        percept= ag.PerceiveCurrentLocation()
        curPos = ag.FindCurrentLocation()
        curLocIndex= 4*(curPos[0]-1)+ curPos[1]-1
        visited[curLocIndex]=True

        breezeClause={}
        stenchClause={}

        if percept[0]==True: #breeze
            breezeClause[curLocIndex+48]=1
        else:
            breezeClause[curLocIndex+48]=-1
        kb.AddClause(breezeClause) #presence/absence of breeze
        if percept[1]==True: #stench
            stenchClause[curLocIndex+16]=1
        else:
            stenchClause[curLocIndex+16]=-1
        kb.AddClause(stenchClause) #presence/absence of stench

        for newLoc in range(16):
            if visited[newLoc]==False:
                tempclauses= kb.getclauses()
                checkClause={newLoc:1, newLoc+32:1}
                tempclauses.append(checkClause)
                if DPLLSatisfiable(tempclauses)==False:
                    #Room is safe
                    noWumpus={newLoc:-1}
                    noPit={newLoc+32:-1}
                    kb.AddClause(noWumpus)
                    kb.AddClause(noPit)
                    dfsVisited = [False for i in range(16)]
                    roomReachable=MoveToUnvisited(ag, visited, newLoc, dfsVisited) #dfs to new
safe Room
                    if roomReachable:
                        break

def main():
    ag = Agent()
    kb= KnowledgeBase()
    print('Start Location: {0}'.format(ag.FindCurrentLocation()))
    ExitWumpusWorld(ag, kb)
    print('{0} reached. Exiting the Wumpus World.'.format(ag.FindCurrentLocation()))
    print('Total number of times DPLL function is called: {0}'.format(numberOfCalls))
```

```
if __name_=='_main_':
    main()
```

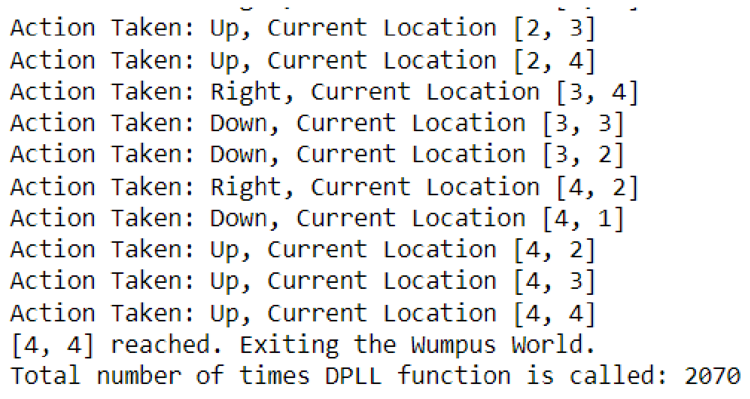
Screenshots of the Outputs:



```
Start Location: [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Left, Current Location [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Up, Current Location [2, 2]
Action Taken: Left, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Left, Current Location [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Right, Current Location [2, 2]
Action Taken: Up, Current Location [2, 3]
Action Taken: Down, Current Location [2, 2]
Action Taken: Left, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Left, Current Location [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Right, Current Location [2, 2]
Action Taken: Up, Current Location [2, 3]
Action Taken: Up, Current Location [2, 4]
Action Taken: Left, Current Location [1, 4]
Action Taken: Right, Current Location [2, 4]
Action Taken: Down, Current Location [2, 3]
Action Taken: Down, Current Location [2, 2]
Action Taken: Right, Current Location [3, 2]
Action Taken: Up, Current Location [3, 3]
Action Taken: Up, Current Location [3, 4]
Action Taken: Left, Current Location [2, 4]
Action Taken: Left, Current Location [1, 4]
```



```
Action Taken: Left, Current Location [2, 4]
Action Taken: Left, Current Location [1, 4]
Action Taken: Right, Current Location [2, 4]
Action Taken: Down, Current Location [2, 3]
Action Taken: Right, Current Location [3, 3]
Action Taken: Down, Current Location [3, 2]
Action Taken: Left, Current Location [2, 2]
Action Taken: Left, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Left, Current Location [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Right, Current Location [2, 2]
Action Taken: Right, Current Location [3, 2]
Action Taken: Up, Current Location [3, 3]
Action Taken: Left, Current Location [2, 3]
Action Taken: Up, Current Location [2, 4]
Action Taken: Right, Current Location [3, 4]
Action Taken: Left, Current Location [2, 4]
Action Taken: Left, Current Location [1, 4]
Action Taken: Right, Current Location [2, 4]
Action Taken: Down, Current Location [2, 3]
Action Taken: Right, Current Location [3, 3]
Action Taken: Down, Current Location [3, 2]
Action Taken: Right, Current Location [4, 2]
Action Taken: Left, Current Location [3, 2]
Action Taken: Up, Current Location [3, 3]
Action Taken: Up, Current Location [3, 4]
Action Taken: Left, Current Location [2, 4]
Action Taken: Left, Current Location [1, 4]
Action Taken: Right, Current Location [2, 4]
Action Taken: Down, Current Location [2, 3]
Action Taken: Down, Current Location [2, 2]
Action Taken: Left, Current Location [1, 2]
Action Taken: Down, Current Location [1, 1]
Action Taken: Right, Current Location [2, 1]
Action Taken: Left, Current Location [1, 1]
Action Taken: Up, Current Location [1, 2]
Action Taken: Right, Current Location [2, 2]
```



```
Action Taken: Up, Current Location [2, 3]
Action Taken: Up, Current Location [2, 4]
Action Taken: Right, Current Location [3, 4]
Action Taken: Down, Current Location [3, 3]
Action Taken: Down, Current Location [3, 2]
Action Taken: Right, Current Location [4, 2]
Action Taken: Down, Current Location [4, 1]
Action Taken: Up, Current Location [4, 2]
Action Taken: Up, Current Location [4, 3]
Action Taken: Up, Current Location [4, 4]
[4, 4] reached. Exiting the Wumpus World.
Total number of times DPLL function is called: 2070
```

Signature of the Student

[YUVRAJ SINGH CHAUHAN]