**EX NO: 3 IMPLEMENT PROPOSITIONAL MODEL CHECKING ALGORITHMS**

**INTRODUCTION:**

Representing knowledge is the key issue in Artificial Intelligence. We can develop a knowledgeable AI agent that can analyze like humans. In the lab program, we will develop a game engine that will detect a murder based on its knowledge base.

If we want a machine to be intelligent enough to think like a human, then first the machine needs some information about the real-world situation/problem. The knowledge of the real world needs to be represented to the machine in the right manner that is readable to a computer system. Propositional logic is one of the simplest methods of knowledge representation to a machine. By implementing propositional logic to make the game engine knowledgeable, and then making the engine to find the solution.

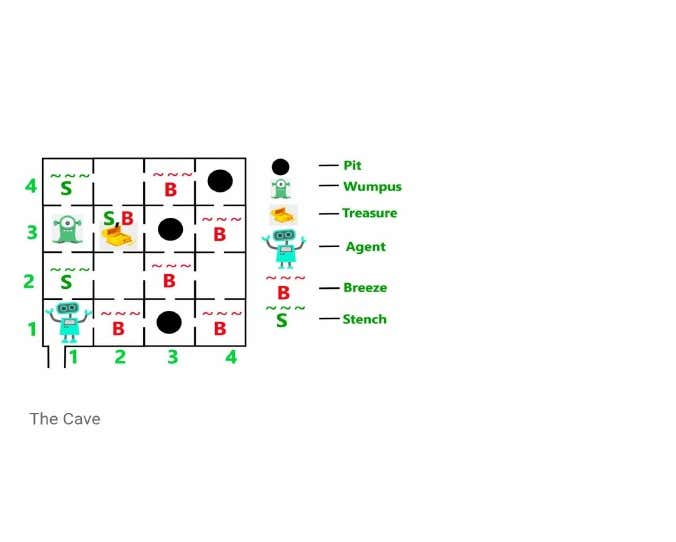
**AIM:**

To implement python code for propositional model checking using Wumpus world problem

**METHODOLOGY**

**1.Goal formation:**

**Input:**



**Output:**

The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.

**2. Problem Statement:**

* A cave with 16(4×4) rooms
* Rooms adjacent (not diagonally) to the Wumpus are stinking Rooms adjacent (not diagonally) to the pit are breezy
* The room with the gold glitters
* Agent’s initial position – Room [1, 1] and facing right side
* Location of Wumpus, gold and 3 pits can be anywhere, except in Room [1, 1].

**3. Search solutions:**

The knowledge-based agent starts from Room[1, 1]. The cave has – some pits, a treasure and a beast named Wumpus. The Wumpus cannot move but eats the one who enters its room. If the agent enters the pit, it gets stuck there.

**4. Execution:**

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

**PROGRAM:**

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()

**OUTPUT:**

curLoc [1, 1]

Percept [breeze, stench]: [False, False]

Action Taken: Right, Current Location [2, 1]

Percept [True, False]

Action Taken: Right, Current Location [3, 1]

P

Agent is DEAD.

Agent cannot perceive. Agent is DEAD. Location: [3, 1]

Percept [None, None]

Action cannot be performed. Agent is DEAD. Location: [3, 1]

Agent cannot perceive. Agent is DEAD. Location: [3, 1]

Percept [None, None]

Action cannot be performed. Agent is DEAD. Location: [3, 1]

Agent cannot perceive. Agent is DEAD. Location: [3, 1]

Percept [None, None]

Action cannot be performed. Agent is DEAD. Location: [3, 1]

Agent cannot perceive. Agent is DEAD. Location: [3, 1]

Percept [None, None]

Action cannot be performed. Agent is DEAD. Location: [3, 1]

Agent cannot perceive. Agent is DEAD. Location: [3, 1]

Percept [None, None]