AISC

Class: B.E COMP

Experiment 05: Solve a given problem using Wumpus World

Learning Objective:

Basic Experiments

Solve a given problem using Wumpus World

Tools: Python

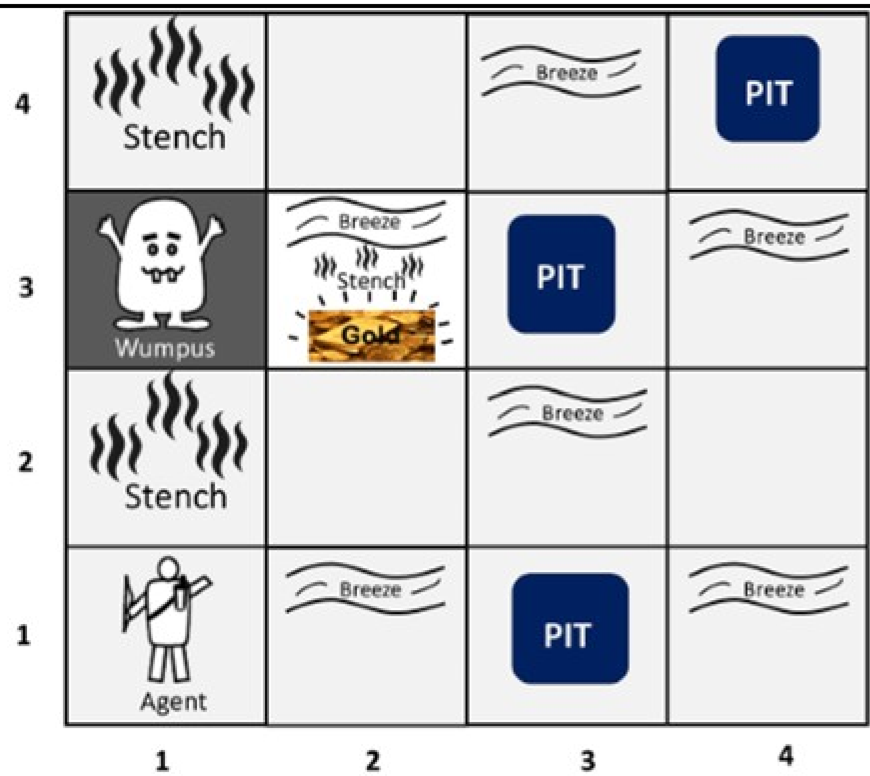
Theory:

lThe Wumpus World in Artificial intelligence

Wumpus world:

The Wumpus world is a simple world example to illustrate the worth of a knowledge-based agent and to represent knowledge representation. It was inspired by a video game Hunt the Wumpus by Gregory Yob in 1973.

The Wumpus world is a cave which has 4/4 rooms connected with passageways. So there are total 16 rooms which are connected with each other. We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room. The Wumpus can be shot by the agent, but the agent has a single arrow. In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever. The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold. So the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus. 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.



There are also some components which can help the agent to navigate the cave. These components are given as follows:

1. The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.
2. The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
3. There will be glitter in the room if and only if the room has gold.
4. The Wumpus can be killed by the agent if the agent is facing to it, and Wumpus will emit a horrible scream which can be heard anywhere in the cave.

PEAS description of Wumpus world:

To explain the Wumpus world we have given PEAS description as below:

Performance measure:

- +1000 reward points if the agent comes out of the cave with the gold.
- -1000 points penalty for being eaten by the Wumpus or falling into the pit.

- -1 for each action, and -10 for using an arrow.
- The game ends if either agent dies or came out of the cave.

Environment:

- A 4*4 grid of rooms.
- The agent initially in room square [1, 1], facing toward the right.
- Location of Wumpus and gold are chosen randomly except the first square [1,1].
- Each square of the cave can be a pit with probability 0.2 except the first square.

Actuators:

- Left turn,
- Right turn
- Move forward
- Grab
- Release
- Shoot.

Sensors:

- The agent will perceive the stench if he is in the room adjacent to the Wumpus. (Not diagonally).
- The agent will perceive breeze if he is in the room directly adjacent to the Pit.
- The agent will perceive the glitter in the room where the gold is present.
- The agent will perceive the bump if he walks into a wall.
- When the Wumpus is shot, it emits a horrible scream which can be perceived anywhere in the cave.
- These percepts can be represented as five element list, in which we will have different indicators for each sensor.
- Example if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as:
  [Stench, Breeze, None, None, None].

The Wumpus world Properties:

- Partially observable: The Wumpus world is partially observable because the agent can only perceive the close environment such as an adjacent room.
- Deterministic: It is deterministic, as the result and outcome of the world are already known.
- Sequential: The order is important, so it is sequential.
- Static: It is static as Wumpus and Pits are not moving.
- Discrete: The environment is discrete.
- One agent: The environment is a single agent as we have one agent only and Wumpus is not considered as an agent.

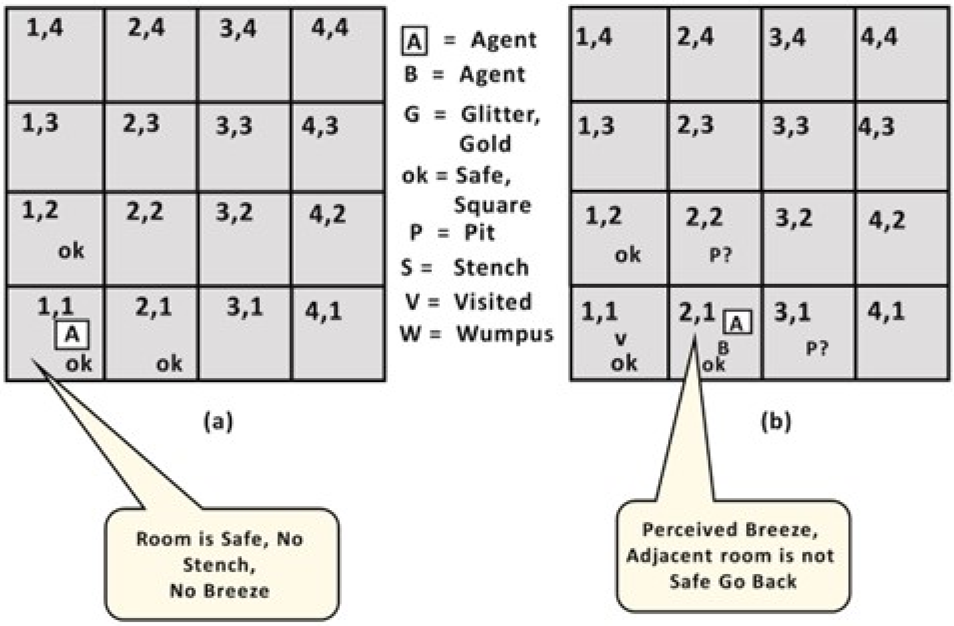
Exploring the Wumpus world:

Now we will explore the Wumpus world and will determine how the agent will find its goal by applying logical reasoning.

Agent's First step:

Initially, the agent is in the first room or on the square [1,1], and we already know that this room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add symbol OK. Symbol A is used to represent agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, W for Wumpus.

At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.



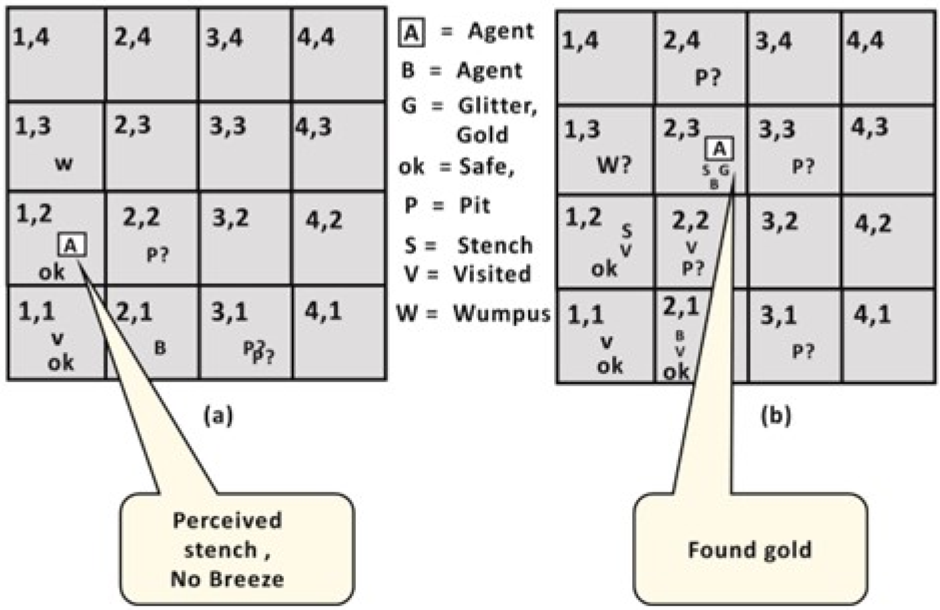
Agent's second Step:

Now agent needs to move forward, so it will either move to [1, 2], or [2,1]. Let's suppose agent moves to the room [2, 1], at this room agent perceives some breeze which means Pit is around this room. The pit can be in [3, 1], or [2,2], so we will add symbol P? to say that, is this Pit room?

Now agent will stop and think and will not make any harmful move. The agent will go back to the [1, 1] room. The room [1,1], and [2,1] are visited by the agent, so we will use symbol V to represent the visited squares.

Agent's third step:

At the third step, now agent will move to the room [1,2] which is OK. In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby. But Wumpus cannot be in the room [1,1] as by rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]). Therefore agent infers that Wumpus is in the room [1,3], and in current state, there is no breeze which means in [2,2] there is no Pit and no Wumpus. So it is safe, and we will mark it OK, and the agent moves further in [2,2].



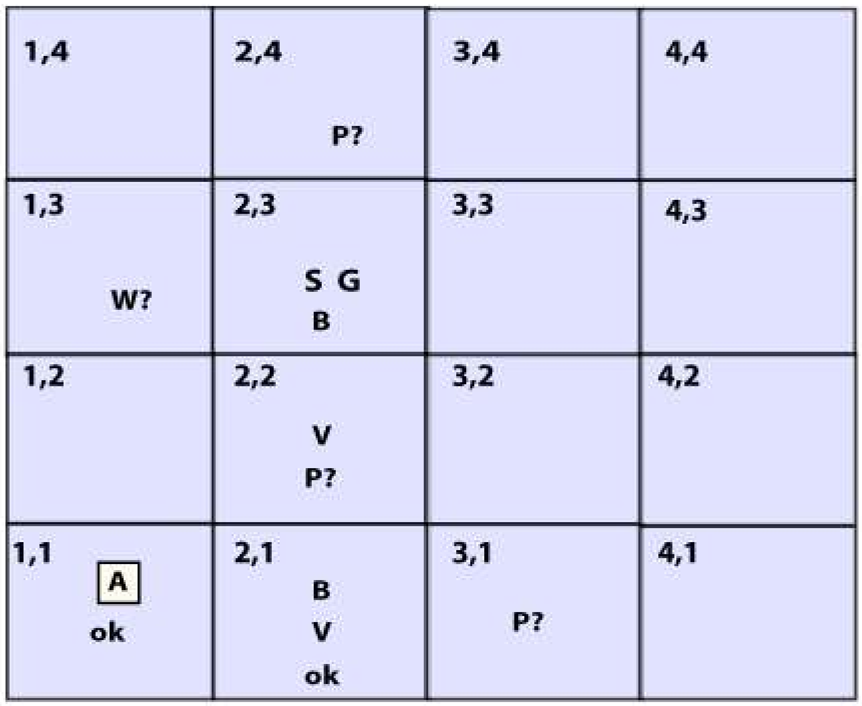
Agent's fourth step:

At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3]. At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

Knowledge-base for Wumpus world

As in the previous topic we have learned about the wumpus world and how a knowledge-based agent evolves the world. Now in this topic, we will create a knowledge base for the wumpus world, and will derive some proves for the Wumpus-world using propositional logic.

The agent starts visiting from first square [1, 1], and we already know that this room is safe for the agent. To build a knowledge base for wumpus world, we will use some rules and atomic propositions. We need symbol [i, j] for each location in the wumpus world, where i is for the location of rows, and j for column location.



Atomic proposition variable for Wumpus world:

- Let Pi,j be true if there is a Pit in the room [i, j].
- Let Bi,j be true if agent perceives breeze in [i, j], (dead or alive).
- Let Wi,j be true if there is wumpus in the square[i, j].
- Let Si,j be true if agent perceives stench in the square [i, j].
- Let Vi,j be true if that square[i, j] is visited.
- Let Gi,j be true if there is gold (and glitter) in the square [i, j].
- Let OKi,j be true if the room is safe.

Note: For a 4 * 4 square board, there will be 7*4*4= 122 propositional variables.

Some Propositional Rules for the wumpus world:

(R1) $\neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$

(R2) $\neg S_{21} \rightarrow \neg W_{11} \wedge \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$

(R3) $\neg S_{12} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{22} \wedge \neg W_{13}$

(R4) $S_{12} \rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$

Note: lack of variables gives us similar rules for each cell.

Representation of Knowledgebase for Wumpus world:

Following is the Simple KB for wumpus world when an agent moves from room [1, 1], to room [2,1]:

$\neg W_{11}$	**$\neg S_{11}$**	**$\neg P_{11}$**	**$\neg B_{11}$**	**$\neg G_{11}$**	**V_{11}**	**OK_{11}**
$\neg W_{12}$	**----**	**$\neg P_{12}$**	**-----**	**----**	**$\neg V_{12}$**	**OK_{12}**
$\neg W_{21}$	**$\neg S_{21}$**	**$\neg P_{21}$**	**B_{21}**	**$\neg G_{21}$**	**V_{21}**	**OK_{21}**

Here in the first row, we have mentioned propositional variables for room[1,1], which is showing that room does not have wumpus(¬ W11), no stench (¬S11), no Pit(¬P11), no breeze(¬B11), no gold (¬G11), visited (V11), and the room is Safe(OK11).

In the second row, we have mentioned propositional variables for room [1,2], which is showing that there is no wumpus, stench and breeze are unknown as an agent has not visited room [1,2], no Pit, not visited yet, and the room is safe.

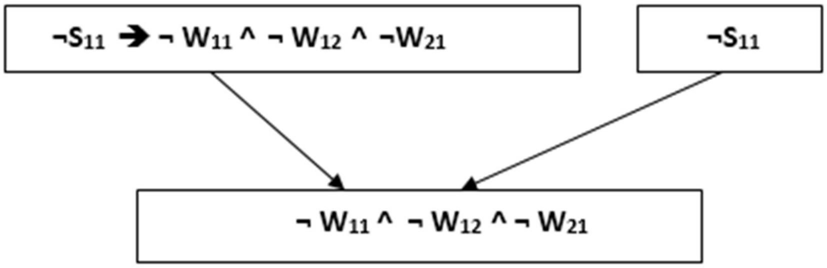
In the third row we have mentioned propositional variable for room[2,1], which is showing that there is no wumpus(¬ W21), no stench (¬S21), no Pit (¬P21), Perceives breeze(B21), no glitter(¬G21), visited (V21), and room is safe (OK21).

Prove that Wumpus is in the room (1, 3)

We can prove that wumpus is in the room (1, 3) using propositional rules which we have derived for the wumpus world and using inference rule.

- Apply Modus Ponens with ¬S11 and R1:

We will firstly apply MP rule with R1 which is ¬S11 → ¬ W11 ^ ¬ W12 ^ ¬ W21, and ¬S11 which will give the output ¬ W11 ^ W12 ^ W12.

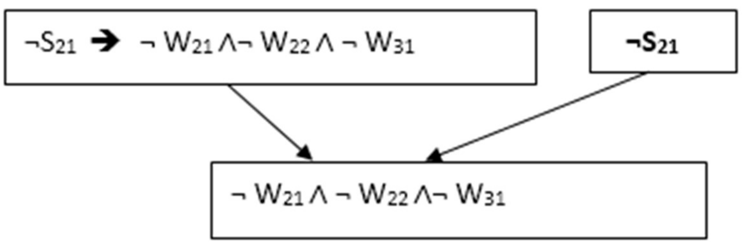


- Apply And-Elimination Rule:

After applying And-elimination rule to ¬ W11 ∧ ¬ W12 ∧ ¬ W21, we will get three statements:
¬ W11, ¬ W12, and ¬W21.

- Apply Modus Ponens to ¬S21, and R2:

Now we will apply Modus Ponens to ¬S21 and R2 which is ¬S21 → ¬ W21 ∧¬ W22 ∧ ¬ W31, which will give the Output as ¬ W21 ∧ ¬ W22 ∧¬ W31

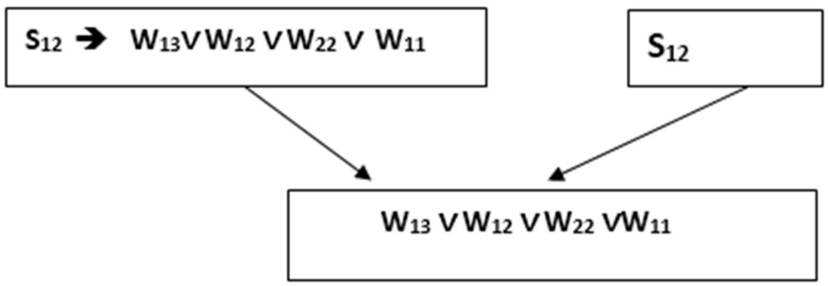


- Apply And -Elimination rule:

Now again apply And-elimination rule to ¬ W21 ∧ ¬ W22 ∧¬ W31, We will get three statements:
¬ W21, ¬ W22, and ¬ W31.

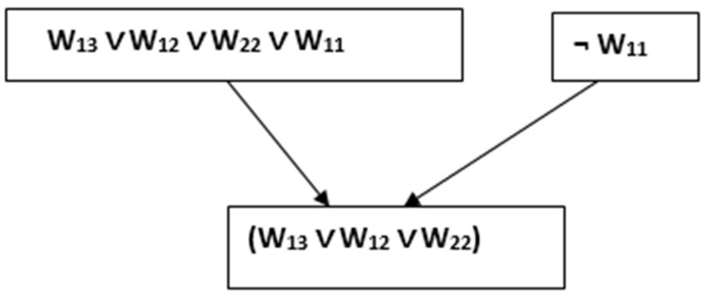
- Apply MP to S12 and R4:

Apply Modus Ponens to S12 and R4 which is S12 → W13 ∨. W12 ∨. W22 ∨.W11, we will get the output as W13∨ W12 ∨ W22 ∨.W11.



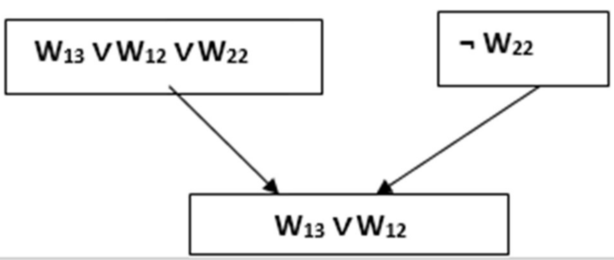
- Apply Unit resolution on W13 ∨ W12 ∨ W22 ∨W11 and ¬ W11 :

After applying Unit resolution formula on W13 ∨ W12 ∨ W22 ∨W11 and ¬ W11 we will get W13 ∨ W12 ∨ W22.



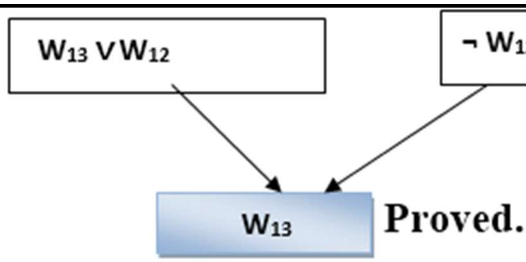
- Apply Unit resolution on W13 ∨ W12 ∨ W22 and ¬ W22 :

After applying Unit resolution on W13 ∨ W12 ∨ W22, and ¬W22, we will get W13 ∨ W12 as output.



- Apply Unit Resolution on W13 ∨ W12 and ¬ W12 :

After Applying Unit resolution on W13 ∨ W12 and ¬ W12, we will get W13 as an output, hence it is proved that the Wumpus is in the room [1, 3].



Implimentation:

Learning Outcomes: Students should have the ability to **implim,ent Wumpus World**

LO1: Understand the problem formulation Example Vacuum Cleaner.

Course Outcomes: Upon completion of the course students will be able to understand problem formulation in ALSC.

Conclusion: Thus, the aim to study and implement **Wumpus World**

Viva Questions:

Ques.1 What do you understand by Wumpus World?

Ques. 2. Explain the basic steps in Wumpus World?

Ques. 3. Write types of problem discussed in detail.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

SOURCE CODE:

```
dirs = ["right","left","up","down"]
gm = 1
gn = 1
pm = 3
pn = 0
h = 0
visited = []
w = [[2,0,0,0],[1,2,0,2],[2,0,2,1],[0,0,0,2]]
print w
p = [[0,0,4,3],[0,4,3,4],[0,0,4,0],[0,4,3,4]]
print p
k =
[["","","",""],["","","",""],["","","",""],["","",
"",""]]
k[pm][pn] = 'S'
print k
tl = []
tl2 = []
present = ""
prev = ""
tmp = ""
tv = 1
mypath = []

def getMatrix(msg,m):
  print msg
  matrix = []
  for i in range(m):
    rl = []
    tmp = raw_input();
    tmp = tmp.replace(' ', '')
    for i in range(len(tmp)):
      fl =  int(tmp[i])
      rl.append(fl)
    matrix.append(rl)
    rl = []
  print matrix
def setdirections_for(i,j):
  si = str(i)
  sj = str(j)
  r = str(j+1)
l = str(j-1)
u = str(i-1)
d = str(i+1)
if i is 0 and j is 0:
  dirs[0] = si + r
  dirs[1] = ""
  dirs[2] = ""
  dirs[3] = d  + sj
  #print i," ",j,"down right"
elif i is 0 and j is 3:
  dirs[0] = ""
  dirs[1] = si + l
  dirs[2] = ""
  dirs[3] = d  + sj
  #print i," ",j,"left down"
elif i is 3 and j is 0:
  dirs[0] = si + r
  dirs[1] = ""
  dirs[2] = u  + sj
  dirs[3] = ""
  #print i," ",j,"up right"
elif i is 3 and j is 3:
  dirs[0] = ""
  dirs[1] = si + l
  dirs[2] = u  + sj
  dirs[3] = ""
  #print i," ",j,"left up"
elif i is 0:
  dirs[0] = si + r
  dirs[1] = si + l
  dirs[2] = ""
  dirs[3] = d  + sj
  #print i," ",j,"left down right"
elif i is 3:
  dirs[0] = si + r
  dirs[1] = si + l
  dirs[2] = u  + sj
  dirs[3] = ""
  #print i," ",j,"left up right"
elif j is 0:
  dirs[0] = si + r
```

```
    dirs[1] = ""
    dirs[2] = u  + sj
    dirs[3] = d  + sj
    #print i," ",j,"up right down"
  elif j is 3:
    dirs[0] = ""
    dirs[1] = si + l
    dirs[2] = u  + sj
    dirs[3] = d  + sj
    #print i," ",j,"left up down"
  else:
    dirs[0] = si + r
    dirs[1] = si + l
    dirs[2] = u  + sj
    dirs[3] = d  + sj
    #print i," ",j,"left up right down"
def getdiagonals_for(i,j):
  iup   = str(i-1)
  idown = str(i+1)
  jright  = str(j+1)
  jleft = str(j-1)
  if i is 0 and j is 0:
    dirs[0] = ""
    dirs[1] = ""
    dirs[2] = ""
    dirs[3] = idown + jright
    #print i," ",j,"down right"
  elif i is 0 and j is 3:
    dirs[0] = ""
    dirs[1] = ""
    dirs[2] = idown + jleft
    dirs[3] = ""
    #print i," ",j,"left down"
  elif i is 3 and j is 0:
    dirs[0] = ""
    dirs[1] = iup + jright
    dirs[2] = ""
    dirs[3] = ""
    #print i," ",j,"up right"
  elif i is 3 and j is 3:
    dirs[0] = iup + jleft
    dirs[1] = ""
    dirs[2] = ""
    dirs[3] = ""
    #print i," ",j,"left up"
  elif i is 0:
    dirs[0] = ""
    dirs[1] = ""
    dirs[2] = idown + jleft
    dirs[3] = idown + jright
    #print i," ",j,"left down right"
  elif i is 3:
    dirs[0] = iup + jleft
    dirs[1] = iup + jright
    dirs[2] = ""
    dirs[3] = ""
    #print i," ",j,"left up right"
  elif j is 0:
    dirs[0] = ""
    dirs[1] = iup + jright
    dirs[2] = ""
    dirs[3] = idown + jright
    #print i," ",j,"up right down"
  elif j is 3:
    dirs[0] = iup + jleft
    dirs[1] = ""
    dirs[2] = idown + jleft
    dirs[3] = ""
    #print i," ",j,"left up down"
  else:
    dirs[0] = iup + jleft
    dirs[1] = iup + jright
    dirs[2] = idown + jleft
    dirs[3] = idown + jright
    #print i," ",j,"left up right down"
def allowedsteps(ls):
  for objs in dirs:
    if objs is not "":
      ls.append(objs)
  return ls
def checkforwumpus(steps):
  if w[int(steps[0])][int(steps[1])] is 1:
    print "Player got killed"
    print visited
    print k
    exit(0)
```

```
  elif w[int(steps[0])][int(steps[1])] is 2:
    return '2'
  else:
    return '0'
def checkforpit(steps):
  if p[int(steps[0])][int(steps[1])] is 3:
    print "Player got killed"
    print k
    exit(0)
  elif p[int(steps[0])][int(steps[1])] is 4:
    return '4'
  else:
    return '0'
def getdiagonals(step):
  ls = []
  getdiagonals_for(int(step[0]),int(step[1]))
  ls = allowedsteps(ls)
  return ls

def stepintocell(step):
  ws = checkforwumpus(step)
  pb = checkforpit(step)
  return ws+pb

def applylogic_from_knowledge():
  global tl2,present,tv
  print "\n\nApplying logic"
  for step in tl:
    print "_______________thinking for
%s__________________"%(step)
    if step in visited:
      print step,"is visited so avoid thinking
about that\n"
      if int(step[0]) is pm and int(step[0]) is
pn:
        print "1"
      elif '2' in k[int(step[0])][int(step[1])]:
        print "2"
    else:
      setdirections_for(int(step[0]),int(step[1])
)
      tl2 = tl2[0:0]
      tl2 = allowedsteps(tl2)
      print "Connections of",step,"checking
for :",tl2
      #print present
      #print prev
      for st in tl2:
        if st  == present:
          print st,"is it is present"
        else:
          print
"***********************checking :",st
          #print "Visited :",visited
          #print present,step,start
          dg = getdiagonals(st)
          print "Diagonals",dg,
          if st in start:
            print st,"is Start state"
          elif st == present:
            print st,"is Present state"
          elif st in visited:
            print st,"yes it is in visited"
            if (k[int(st[0])][int(st[1])] in
['04','40','02','20']) and
(k[int(present[0])][int(present[1])] in
['02','20','04','40']):
              print "Applying first condition and
is true"
              k[int(step[0])][int(step[1])] = 'S'
              print "Putting Safe State at
",int(step[0]),int(step[1])
              print "Visited :",visited
              return step
              print "\n"
          else:
            for objs in dg:
              if objs in start:
                print "Diagonal",objs,"is Start
state"
              elif objs == present:
                print "Diagonal",objs,"is Present
state"
              elif objs in visited:
                tv = 1
```

```
            print "%s is diagonal of %s which
is visited"%(objs,st)

          else:
            print objs,"not visited"
        else:
          if tv is not 1:
            print "here Nothing could be done
for",st
  print "________________\n"
  return 0
def whats_nextstep(ws,wp):
  global tl,present,prev,tmp,h
  print "\nThinking whats_nextstep
from",ws,wp,
  print "\nNow present is :",present,
  print "Now previous is :",prev
  setdirections_for(ws,wp)
  tl = tl[0:0]
  tl = allowedsteps(tl)
  if ws == gm and wp == gn:
    h = 1
    if h is 1:
      print "\nHurray!!Got the Gold",
    print "\nPresently in %s returning back to
%s"%(present,start)
    #print mypath
    for steps in range(len(mypath),0,-1):
      #print "Stepping into ",mypath[steps-1]
      prev = present
      present = mypath[steps-1]
      #print "Present : ",present,"Previous
:",prev
    print "Now reached to :",present
    exit(0)
    #return_to_initial_postition()
  elif k[ws][wp] is "S":
    print "\nSafe Cell"
    print "Allowed Steps",tl,
    for step in tl:
      tmp = step
      if step not in visited:
        print step,"Not visited"
        prev = present
        print "previous cell :",prev
        present = step[0]+step[1]
        print "present cell",present
        #print "Visited :",visited
        #Theres something wrong here
        k[int(step[0])][int(step[1])] =
stepintocell(step)
        print "\nStepping into ",step[0],step[1]
        mypath.append(step)
        visited.append(step)
        print visited
        print k
        whats_nextstep(int(step[0]),int(step[1])
)

        print
      else:
        print step,"already visited",
  else:
    print "\nNot a Safe Cell",
    print "\nConnections of ",present,":",tl,
    if ws is 2 and wp is 2:
      print tl
      #print "exiting"
      #exit(0)
    l = applylogic_from_knowledge()
    print "i got here",l
    if l is 0:
      print "Stepping Back to",prev,"\n"
      mypath.pop()
      print prev
      present = prev
      print present,prev
      whats_nextstep(int(prev[0]),int(prev[1]))
    else:
      print "here I
reached********************"
      visited.append(l)
      print visited
      prev = tmp
      present = l
      print present,prev
      print "\nfrom here Stepping into ",l,
```

```
    mypath.append(l)
    whats_nextstep(int(l[0]),int(l[1]))
if __name__ == "__main__":
  #getMatrix("Enter Matrix",4);
  '''
  for i in range(1,5):
    for j in range(1,5):
      setdirections_for(i,j)
      print "from",i,j,"it can move to",
      allowedcells()
  '''
  prm = pm
  prn = pn
  start = str(pm)+str(pn)
  visited.append(start)
  print visited
  print "Starting from ",pm,pn
  prev = str(pm)+str(pn)
  present = str(pm)+str(pn)
  mypath.append(start)
  whats_nextstep(pm,pn)
  print present
  print k
```

OUTPUT:

```
[[2, 0, 0, 0], [1, 2, 0, 2], [2, 0, 2, 1], [0, 0, 0, 2]]
[[0, 0, 4, 3], [0, 4, 3, 4], [0, 0, 4, 0], [0, 4, 3, 4]]
[['', '', '', ''], ['', '', '', ''], ['', '', '', ''], ['S', '', '', '']]
['30']
Starting from  3 0

Thinking whats_nextstep from 3 0
Now present is : 30 Now previous is : 30

Safe Cell
Allowed Steps ['31', '20'] 31 Not visited
previous cell : 30
present cell 31

Stepping into  3 1
['30', '31']
[['', '', '', ''], ['', '', '', ''], ['', '', '', ''], ['S', '04', '', '']]

Thinking whats_nextstep from 3 1
Now present is : 31 Now previous is : 30

Not a Safe Cell
Connections of  31 : ['32', '30', '21']

Applying logic
_______________thinking for 32_________________
Connections of 32 checking for : ['33', '31', '22']
***********************checking : 33
Diagonals ['22'] 22 not visited
31 is it is present
***********************checking : 22
Diagonals ['11', '13', '31', '33'] 11 not visited
13 not visited
Diagonal 31 is Present state
33 not visited
_______________thinking for 30_________________
30 is visited so avoid thinking about that

_______________thinking for 21_________________
Connections of 21 checking for : ['22', '20', '11', '31']
***********************checking : 22
Diagonals ['11', '13', '31', '33'] 11 not visited
13 not visited
Diagonal 31 is Present state
33 not visited
***********************checking : 20
Diagonals ['11', '31'] 11 not visited
Diagonal 31 is Present state
```

***********************checking : 11
Diagonals ['00', '02', '20', '22'] 00 not visited
02 not visited
20 not visited
22 not visited
31 is it is present
____

i got here 0
Stepping Back to 30

30
30 30

Thinking whats_nextstep from 3 0
Now present is : 30 Now previous is : 30

Safe Cell
Allowed Steps ['31', '20'] 31 already visited
20 Not visited
previous cell : 30
present cell 20

Stepping into 2 0
['30', '31', '20']
[['', '', '', ''], ['', '', '', ''], ['20', '', '', ''], ['S', '04', '', '']]

Thinking whats_nextstep from 2 0
Now present is : 20 Now previous is : 30

Not a Safe Cell
Connections of 20 : ['21', '10', '30']

Applying logic
_______________thinking for 21_________________
Connections of 21 checking for : ['22', '20', '11', '31']
***********************checking : 22
Diagonals ['11', '13', '31', '33'] 11 not visited
13 not visited
31 is diagonal of 22 which is visited
33 not visited
20 is it is present
***********************checking : 11
Diagonals ['00', '02', '20', '22'] 00 not visited
02 not visited
Diagonal 20 is Present state
22 not visited
***********************checking : 31
Diagonals ['20', '22'] 31 yes it is in visited
Applying first condition and is true
Putting Safe State at 2 1
Visited : ['30', '31', '20']
i got here 21
here I reached*******************
['30', '31', '20', '21']
21 20

from here Stepping into 21
Thinking whats_nextstep from 2 1
Now present is : 21 Now previous is : 20

Safe Cell
Allowed Steps ['22', '20', '11', '31'] 22 Not visited
previous cell : 21
present cell 22

Stepping into 2 2
['30', '31', '20', '21', '22']
[['', '', '', ''], ['', '', '', ''], ['20', 'S', '24', ''], ['S', '04', '', '']]

Thinking whats_nextstep from 2 2
Now present is : 22 Now previous is : 21

Not a Safe Cell
Connections of 22 : ['23', '21', '12', '32']
['23', '21', '12', '32']

Applying logic
_______________thinking for 23_________________
Connections of 23 checking for : ['22', '13', '33']

22 is it is present
***********************checking : 13
Diagonals ['02', '22'] 02 not visited
Diagonal 22 is Present state
***********************checking : 33
Diagonals ['22'] Diagonal 22 is Present state
_______________thinking for
21_________________
21 is visited so avoid thinking about that

_______________thinking for
12_________________
Connections of 12 checking for : ['13', '11', '02', '22']
***********************checking : 13
Diagonals ['02', '22'] 02 not visited
Diagonal 22 is Present state
***********************checking : 11
Diagonals ['00', '02', '20', '22'] 00 not visited
02 not visited
20 is diagonal of 11 which is visited
Diagonal 22 is Present state
***********************checking : 02
Diagonals ['11', '13'] 11 not visited
13 not visited
22 is it is present
_______________thinking for
32_________________
Connections of 32 checking for : ['33', '31', '22']
***********************checking : 33
Diagonals ['22'] Diagonal 22 is Present state
***********************checking : 31
Diagonals ['20', '22'] 31 yes it is in visited
22 is it is present

____

i got here 0
Stepping Back to 21

21
21 21

Thinking whats_nextstep from 2 1
Now present is : 21 Now previous is : 21

Safe Cell
Allowed Steps ['22', '20', '11', '31'] 22 already visited 20 already visited 11 Not visited
previous cell : 21
present cell 11

Stepping into 1 1
['30', '31', '20', '21', '22', '11']
[['', '', '', ''], ['', '24', '', ''], ['20', 'S', '24', ''], ['S', '04', '', '']]

Thinking whats_nextstep from 1 1
Now present is : 11 Now previous is : 21

Hurray!!Got the Gold
Presently in 11 returning back to 30
Now reached to : 30