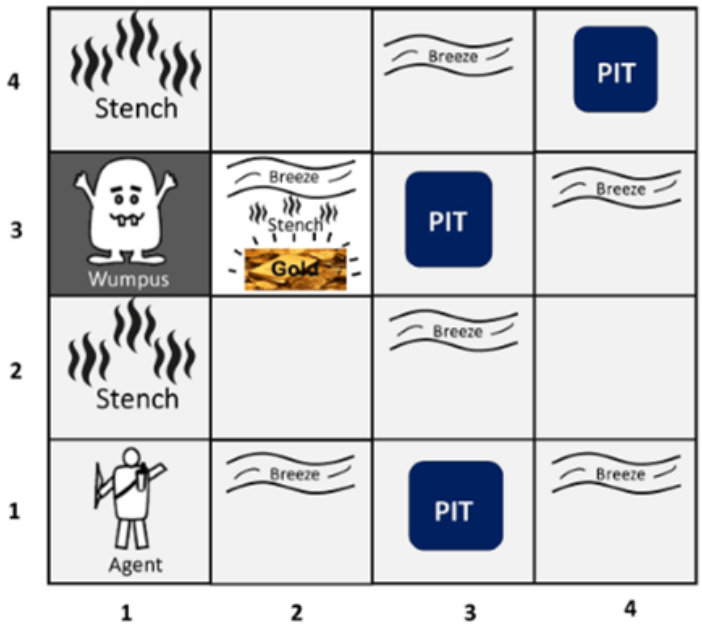
NAME: Meith Navlakha
SAP: 60004190068
AI Practical

EXPERIMENT 8

AIM: Implementation on AI Problem: Wumpus World

THEORY:

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 falling into the 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.



The components of Wumpus World are:

- The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.
- The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
- There will be glitter in the room if and only if the room has gold.
- 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.

NAME: Meith Navlakha
SAP: 60004190068
AI Practical

PEAS description of Wumpus world:

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

CODE:

```
import java.util.Scanner;

class Block {
   public static final int NOT_PRESENT = 0;
   public static final int UNSURE = 1;

   public boolean hasBreeze, hasPit;
   public int pitStatus = UNSURE;

   public boolean hasStench, hasWumpus;
   public int wumpusStatus = UNSURE;

   public boolean hasGold;

   public boolean isVisited;
}
class WumpusWorld {
   static Block[][] maze;
   static int n;

   public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter the order of the maze: ");
    n = sc.nextInt();

    maze = new Block[n][n];
```

```
for(int i=0; i<n; i++) {
   maze[i] = new Block[n];
   for(int j=0; j<n; j++)
      maze[i][j] = new Block();
}

System.out.print("\n Enter the number of pits: ");
int pits = sc.nextInt();

for(int i=0; i<pits; i++) {
   System.out.print("Enter the location of pit " + (i+1) + ": ");
   addPit(n-sc.nextInt(), sc.nextInt()-1);
}

System.out.print("\nEnter the location of wumpus: ");
addWumpus(n-sc.nextInt(), sc.nextInt()-1);

System.out.print("\nEnter the location of gold: ");
addGold(n-sc.nextInt(), sc.nextInt()-1);

System.out.print("\nEnter the starting location: ");
int r = n - sc.nextInt();
int c = sc.nextInt() - 1;
int rPrev = -1, cPrev = -1;

int moves = 0;
System.out.println("\nInitial state:");
printMaze(r, c);

while(!maze[r][c].hasGold) {
   maze[r][c].isVisited = true;
   maze[r][c].pitStatus = Block.NOT_PRESENT;
   maze[r][c].wumpusStatus = Block.NOT_PRESENT;

   if(!maze[r][c].hasBreeze) {
      if(r >= 1 && maze[r-1][c].pitStatus == Block.UNSURE)
         maze[r-1][c].pitStatus = Block.NOT_PRESENT;
      if(r <= (n-2) && maze[r+1][c].pitStatus == Block.UNSURE)
         maze[r+1][c].pitStatus = Block.NOT_PRESENT;
      if(c >= 1 && maze[r][c-1].pitStatus == Block.UNSURE)
         maze[r][c-1].pitStatus = Block.NOT_PRESENT;
      if(c <= (n-2) && maze[r][c+1].pitStatus == Block.UNSURE)
         maze[r][c+1].pitStatus = Block.NOT_PRESENT;
   }

   if(!maze[r][c].hasStench) {
      if(r >= 1 && maze[r-1][c].wumpusStatus == Block.UNSURE)
```

```
            maze[r-1][c].wumpusStatus = Block.NOT_PRESENT;
        if(r <= (n-2) && maze[r+1][c].wumpusStatus == Block.UNSURE)
            maze[r+1][c].wumpusStatus = Block.NOT_PRESENT;
        if(c >= 1 && maze[r][c-1].wumpusStatus == Block.UNSURE)
            maze[r][c-1].wumpusStatus = Block.NOT_PRESENT;
        if(c <= (n-2) && maze[r][c+1].wumpusStatus == Block.UNSURE)
            maze[r][c+1].wumpusStatus = Block.NOT_PRESENT;
    }

    boolean foundNewPath = false;

    if(r >= 1 && !((r-1) == rPrev && c == cPrev) && maze[r-1][c].isVisited == false &&
maze[r-1][c].pitStatus == Block.NOT_PRESENT && maze[r-
1][c].wumpusStatus == Block.NOT_PRESENT) {
        rPrev = r;
        cPrev = c;

        r--;
        foundNewPath = true;
    }
    else if(r <= (n-2) && !((r+1) == rPrev && c == cPrev) && maze[r+1][c].isVisited == false
&& maze[r+1][c].pitStatus == Block.NOT_PRESENT &&
maze[r+1][c].wumpusStatus == Block.NOT_PRESENT) {
        rPrev = r;
        cPrev = c;

        r++;
        foundNewPath = true;
    }
    else if(c >= 1 && !(r == rPrev && (c-1) == cPrev) && maze[r][c-1].isVisited == false &&
maze[r][c-1].pitStatus == Block.NOT_PRESENT && maze[r][c-
1].wumpusStatus == Block.NOT_PRESENT) {
        rPrev = r;
        cPrev = c;

        c--;
        foundNewPath = true;
    }
    else if(c <= (n-2) && !(r == rPrev && (c+1) == cPrev) && maze[r][c+1].isVisited == false
&& maze[r][c+1].pitStatus == Block.NOT_PRESENT &&
maze[r][c+1].wumpusStatus == Block.NOT_PRESENT) {
        rPrev = r;
        cPrev = c;

        c++;
        foundNewPath = true;
    }
```

```
    if(!foundNewPath) {
        int temp1 = rPrev;
        int temp2 = cPrev;

        rPrev = r;
        cPrev = c;

        r = temp1;
        c = temp2;
    }

    moves++;

    System.out.println("\n\nMove " + moves + ":");
    printMaze(r, c);

    if(moves > n*n) {
        System.out.println("\nNo solution found!");
        break;
    }
 }

 if(moves <= n*n)
    System.out.println("\nFound gold in " + moves + " moves.");

 sc.close();
}

static void addPit(int r, int c) {
 maze[r][c].hasPit = true;

 if(r >= 1)
    maze[r-1][c].hasBreeze = true;
 if(r <= (n-2))
    maze[r+1][c].hasBreeze = true;
 if(c >= 1)
    maze[r][c-1].hasBreeze = true;
 if(c <= (n-2))
    maze[r][c+1].hasBreeze = true;
}

static void addWumpus(int r, int c) {
 maze[r][c].hasWumpus = true;

 if(r >= 1)
    maze[r-1][c].hasStench = true;
```

```
    if(r <= (n-2))
       maze[r+1][c].hasStench = true;
    if(c >= 1)
       maze[r][c-1].hasStench = true;
    if(c <= (n-2))
       maze[r][c+1].hasStench = true;
   }

   static void addGold(int r, int c) {
    maze[r][c].hasGold = true;
   }

   static void printMaze(int r, int c) {
    for(int i=0; i<n; i++) {
       for(int j=0; j<n; j++) {
          char charToPrint = '-';
          if(r == i && c == j)
             charToPrint = '*';
          else if(maze[i][j].hasPit)
             charToPrint = 'O';
          else if(maze[i][j].hasWumpus)
             charToPrint = 'X';
          else if(maze[i][j].hasGold)
             charToPrint = '$';
         else if(maze[i][j].hasBreeze)
             charToPrint = 'B';
         else if(maze[i][j].hasStench)
             charToPrint = 'S';

          System.out.print(charToPrint + "\t");
       }
       System.out.println();
    }
   }
}
```

NAME: Meith Navlakha
SAP: 60004190068
AI Practical

OUTPUT:

```
C:\Windows\System32\cmd.exe
C:\Users\meith\OneDrive\Desktop\SEM 5\AI>javac WumpusWorld.java

C:\Users\meith\OneDrive\Desktop\SEM 5\AI>java WumpusWorld
Enter the order of the maze: 4

Enter the number of pits: 3
Enter the location of pit 1: 1 3
Enter the location of pit 2: 3 3
Enter the location of pit 3: 4 4

Enter the location of wumpus: 3 1

Enter the location of gold: 3 2

Enter the starting location: 1 1

Initial state:
S       -       B       O
X       $       O       B
S       -       B       -
*       B       O       B


Move 1:
S       -       B       O
X       $       O       B
*       -       B       -
-       B       O       B


Move 2:
S       -       B       O
X       $       O       B
S       -       B       -
*       B       O       B


Move 3:
S       -       B       O
X       $       O       B
S       -       B       -
-       *       O       B
```

NAME: Meith Navlakha
SAP: 60004190068
AI Practical

```
C:\Windows\System32\cmd.exe
X         $         O         B
S         -         B         -
*         B         O         B


Move 1:
S         -         B         O
X         $         O         B
*         -         B         -
-         B         O         B


Move 2:
S         -         B         O
X         $         O         B
S         -         B         -
*         B         O         B


Move 3:
S         -         B         O
X         $         O         B
S         -         B         -
-         *         O         B


Move 4:
S         -         B         O
X         $         O         B
S         *         B         -
-         B         O         B


Move 5:
S         -         B         O
X         *         O         B
S         -         B         -
-         B         O         B

Found gold in 5 moves.

C:\Users\meith\OneDrive\Desktop\SEM 5\AI>
```

CONCLUSION: In this Experiment, I implemented the Wumpus World program which is a knowledge-based agent. Here, the agent can perform 4 movements, forward, backward, left or right. The user can sense the stench or breeze and make appropriate actions accordingly using the knowledge base. Also, the agent is awarded or penalised accordingly based on the actions. The game stops once the user finds the gold. After each attempt it adds information, it learnt from the previous attempt into the knowledge base thereby continually updating the knowledge base.