JUNAID GIRKAR
60004190057
TE COMPS A4

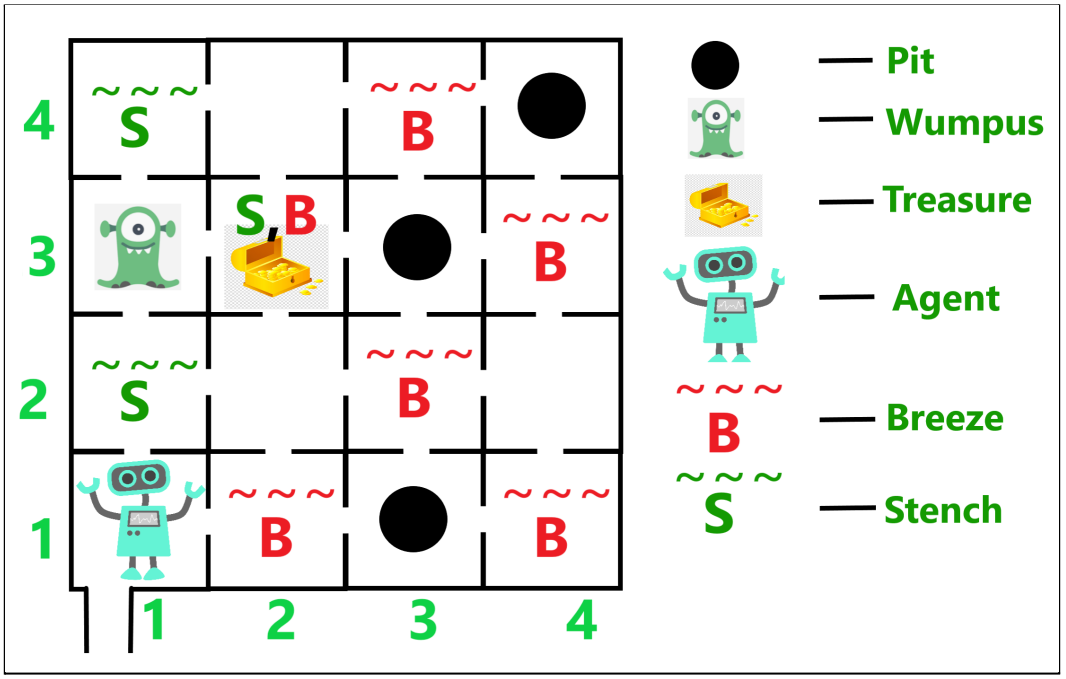
EXPERIMENT 8

ARTIFICIAL INTELLIGENCE

Aim: Implementation on any AI Problem : Wumpus world, Tic-tac-toe

Theory:

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 is eaten by the Wumpus.
Some elements support the agent to explore the cave, like -The wumpus's adjacent rooms are stenchy. -The agent is given one arrow which it can use to kill the wumpus when facing it (Wumpus screams when it is killed). – The adjacent rooms of the room with pits are filled with breeze. -The treasure room is always glittery.



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 grids 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 a 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 precepts can be represented as a 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:

```
public 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;
}



import java.util.Scanner;

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("\nEnter 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;

       System.out.print("\nYour Position : *\nWumpus : X\nGold : $\nPit : O");

        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 = '$';

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

Output:

```
Enter the order of the maze: 4

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

Enter the location of wumpus: 2 4

Enter the location of gold: 2 3

Enter the starting location: 1 1

Your Position : *
Wumpus : X
Gold : $
Pit : O
```

```
Initial state:
-   -   -   -
-   -   -   O
-   -   $   X
*   -   -   O

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

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

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

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

Move 5:
-   -   -   -
-   *   -   O
-   -   $   X
-   -   -   O

Move 6:
-   -   -   -
-   -   -   O
```

```
-    *    $    X
-    -    -    O


Move 7:
-    -    -    -
-    -    -    O
-    -    $    X
-    *    -    O


Move 8:
-    -    -    -
-    -    -    O
-    -    $    X
-    -    *    O


Move 9:
-    -    -    -
-    -    -    O
-    -    *    X
-    -    -    O

Found gold in 9 moves.
```

Conclusion: We learnt about Knowledge based agents and about the Wumpus World game. We then explored the PEAS of an agent in the Wumpus World game and wrote a code in java to simulate the game.