Vivek Poojari
2019130050
TE COMPS
AIML Lab

Experiment 1

Aim: To implement an intelligent agent in python.

Code:

```
global N
N = 8

def printSolution(board):
    for i in range(N):
        for j in range(N):
            print (board[i][j], end = " ")
        print()

# A utility function to check if a queen can
# be placed on board[row][col]. Note that this
# function is called when "col" queens are
# already placed in columns from 0 to col -1.
# So we need to check only left side for
# attacking queens
def isSafe(board, row, col):

    # Check this row on left side
    for i in range(col):
        if board[row][i] == 1:
            return False

    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check lower diagonal on left side
    for i, j in zip(range(row, N, 1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
```

```
            return False

    return True

def solveNQUtil(board, col):
    # base case: If all queens are placed
    # then return true
    if col >= N:
        return True
    # Consider this column and try placing
    # this queen in all rows one by one
    for i in range(N):

        if isSafe(board, i, col):

            # Place this queen in board[i][col]
            board[i][col] = 1

            # recur to place rest of the queens
            if solveNQUtil(board, col + 1) == True:
                return True

            # If placing queen in board[i][col
            # doesn't lead to a solution, then
            # queen from board[i][col]
            board[i][col] = 0

    # if the queen can not be placed in any row in
    # this colum col then return false
    return False

# This function solves the N Queen problem using
# Backtracking. It mainly uses solveNQUtil() to
# solve the problem. It returns false if queens
# cannot be placed, otherwise return true and
# placement of queens in the form of 1s.
# note that there may be more than one
# solutions, this function prints one of the
# feasible solutions.
def solveNQ():
    board = [ [0, 0, 0, 0, 0, 0, 0, 0],
              [0, 0, 0, 0, 0, 0, 0, 0],
```

```
            [0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0]]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False

    printSolution(board)
    return True

# Driver Code
solveNQ()
```

Output:

```
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
```

Conclusion:

The environment for this experiment consisted of a chessboard represented by a n*n grid. As each queen enter row by row, a checker function will be used to approve for the validity of the position of the queen. The placed queens must follow the principle of safe squares. A square is considered to be safe if no other queen sees that square vertically, horizontally as well as diagonally. So in this manner, backtracking algorithm is used to place the queens on the chessboard. This agent is unable to place queens and find solution for 2 specific cases.