Shri Vile Parle Kelavani Mandal's

INSTITUTE OF TECHNOLOGY

DHULE (M.S.)

DEPARMENT OF COMPUTER ENGINEERING

Name: Pagariya Sakshi	**Roll No**.:17
Subject : Artificial Intelligence Lab	**Subject Code :** BTCOL707
Class: Final Year Comp. Engg.	**Expt. No. :** 02

Title : Write a program to solve 8 queens problem.

Problem Staement :

Solve 8 queens problem using prolog.

Software Required :

Prolog

Theory :

The objective of the 8-queens problem, a traditional chessboard puzzle, is to arrange eight queens on an 8x8 chessboard so that no two queens pose a threat to one another. This implies that no two queens may be situated in the same diagonal, column, or row. This is a written example of how to solve the 8-queens problem:

```
. Q . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. . . Q . . . .
Q . . . . . . .
```

In this instance, the eight queen places on the chessboard are represented by the 'Q' characters. As you can see, no row, column, or diagonal is occupied by more than one queen.

Finding every queen configuration on the chessboard that satisfies the non-attack constraint—that is, ensuring that no two queens pose a threat to one another—is the first step in solving the eight-queens issue. Because it supports both constraint logic programming and logic programming, Prolog is a popular language for handling this kind of problem.

The task of solving the 8-queens issue in Prolog entails figuring out how to arrange eight queens on an 8x8 chessboard so that no two of them pose a threat to one another. This implies that no two queens may be situated in the same diagonal, column, or row. A Prolog program to resolve the 8-queens problem is provided here:

```
% Predicate to check if a queen can be placed safely in a given row and column.
is_safe(_, []).
is_safe(Queen, [Row/Col|Queens]) :-
    Queen #\= Row,
    Queen #\= Col,
    abs(Queen - Row) #\= abs(Col - Queens),
    is_safe(Queen, Queens).

% Predicate to find a solution for N-queens.
queens(N, Solution) :-
    length(Solution, N),
    Solution ins 1..N, % Initialize the domain of the variables.
    all_distinct(Solution), % Ensure queens are placed in different columns.
    is_safe(1, Solution), % Check safety of placement for each queen.
    labeling([], Solution). % Find a valid labeling.

% Predicate to print a solution.
print_solution(Solution) :-
    length(Solution, N),
    write('Solution: '), writeln(Solution),
    draw_board(Solution, N, N).
```

```
% Predicate to draw the chessboard.
draw_board(_, 0, _).
draw_board(Solution, Row, N) :-
    Row > 0,
    draw_row(Solution, Row, N),
    NextRow is Row - 1,
    draw_board(Solution, NextRow, N).

draw_row([], _, 0).
draw_row([Col|Queens], Row, N) :-
    N > 0,
    (Col =:= Row -> write('Q ') ; write('. ')),
    NextN is N - 1,
    draw_row(Queens, Row, NextN).

% Predicate to solve and print all solutions.
all_solutions(N) :-
    findall(Solution, queens(N, Solution), Solutions),
    length(Solutions, NumSolutions),
    write('Number of solutions: '), writeln(NumSolutions),
    maplist(print_solution, Solutions).



% Example usage: Solve the 8-queens problem and print all solutions.
:- all_solutions(8).
```

Conclusion:

Within this course:

is_safe/2 determines if a queen may be positioned on the board securely and without endangering other queens.

queens/2, where N is the number of queens, specifies the primary predicate for resolving the N-queens issue.

To print the solutions in a format that is legible by humans, use print_solution/1.

	The chessboard with the queens arranged is drawn using the draw_board/3 and draw_row/3 functions. A helpful predicate called all_solutions/1 locates and outputs every solution to the N-queens problem.