**Programming Project 2:**

Solving N queen problem using random restart.

Compiler: Java

Input: ‘N’ – no. of queens to be solved

Output: A solved state in which ‘X’ represents the presence of a queen in the chess board.

N-queen problem formulation:

State: A state of N queen board consists of N queens on the board one per column

Initial State: Any random state with one queen per column on the chess board

Successor function: Movement of queen in each column from one to another square. By this each state has 56 successors.

Goal test: Check whether there is any conflict between any two queens on the board.

Program Structure:

The program has four classes NqueenSolver, RandomHill, BoardState, QueenState.

NqueenSolver is the main class in which it takes the input for number of queens to be solved, calls other classes to solve the board and prints the board.

RandomHill class executes the random restart hill climbing algorithm.

BoardState class maintains the chess board by generating the successors of the current board and their corresponding heuristics.

QueenState class maintains the position of queen in the chess board.

Variables:

Queensize - number of queens to be solved

Restartcount - counter for number of resarts

Stepsclimbed - counter for number of stepsclimbed

heuristicValue – heuristic of the board

queenRow – Queen’s row position

queenCol – Queen’s column position

Procedure to compute the heuristic function:

The heuristic cost function h is the number of pairs of queens that are attacking each other, either directly or indirectly. This value will be zero for the global minimum

**public** **int** heuristicEvaluation()

{

**for**(**int** i=0; i<qCount-1; i++){

**for**(**int** j=i+1; j<qCount; j++){

**if**(queenPosition[i].isConflicting(queenPosition[i],queenPosition[j]))

//Compares the queen in a column with all the queens in the columns right of it.

{

heuristicValue++; //Increments the heuristics when ever a conflict is found

}

}

}

**return** heuristicValue;

}

Other Methods:

randomHill() - This method executes the random restart hill climbing algorithm.

hillClimbing()-This method executes the hill climbing algorithm.

randominitBoard()-This method initializes the chess board to a random initial state.

successorBoard(BoardState initialBoard)- This method generates the successor boards with corresponding queen states and its heuristics depending on the start state.

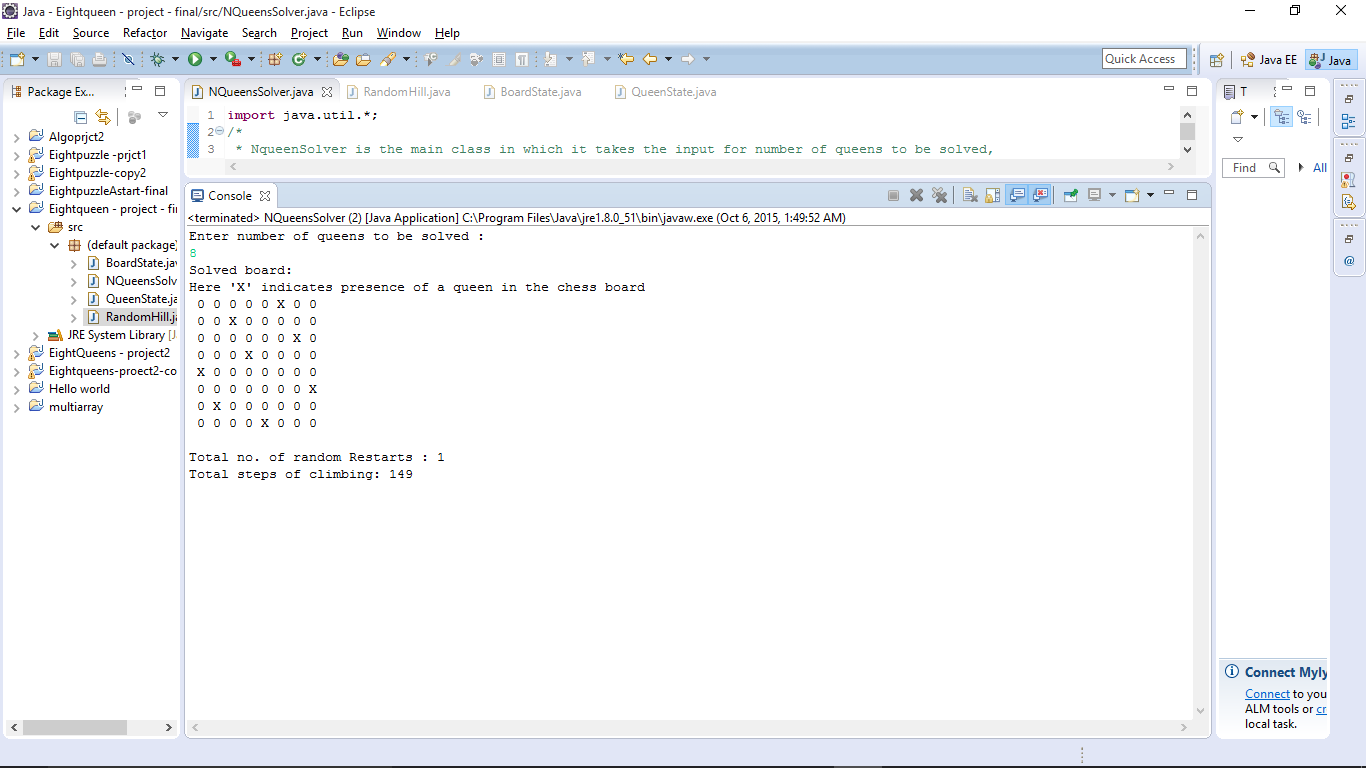
compareTo(BoardState board)- This method compares the heuristics of two board states.

isConflicting(QueenState q1,QueenState q2)- his method checks if there is any conflict between any two queens of the chess board.

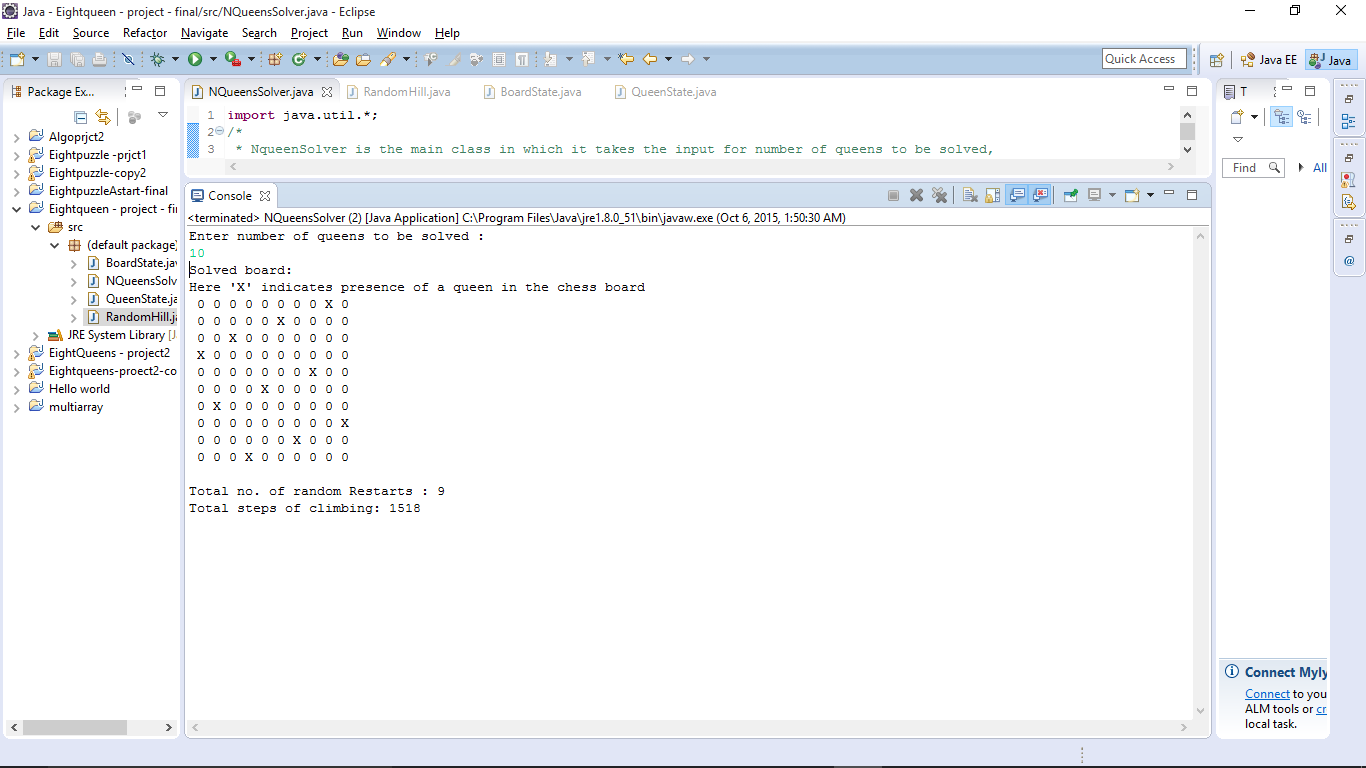
shiftQueen(int shift, int queens)- This method changes the position of a queen in a column of the board.

Execution Results:

N=8:



N = 10



N = 30

