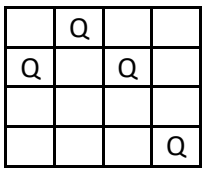
ITCS 6150 – Intelligent Systems

Programming Project 2
Archit Parnami
(800960108)

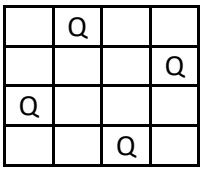
SOLVING N-QUEENS PROBLEM

1. Problem Formulation

- The n-queens problem consists of an n x n board with n queens. Each queen belongs to one column and is randomly placed in any row at the beginning. The object is to reach a goal state.



Initial State



Goal State

- **State:** Each unique combination of placing n-Queens on n x n board is called a state. There are $n!$ possible states for this problem. A state can be programmatically represented by an array of size n where each element has the location of each queen.

- **Initial State:** a randomly generated state
- **Actions**: Each queen can either move *Up* or *Down*.
- **Transition model**: Given a state and action, this returns the resulting state
- **Goal test**: A goal state is one in which no queens are attacking each other i.e. no queens are in conflict.

.

2. The Hill Climbing Algorithm

Heuristic function h(x, y): Number of conflicts if a queen is at position (x, y) on the board

- The hill climbing algorithm for n-Queens problem is a greedy search approach in which the heuristic values h(x, y) is calculated for every position on n x n board.
- The objective of the search is to find the location for which h(x, y) is zero.
- The algorithm progresses when it finds minimum h(x, y) which is also less than the current state cost.
- The program results in no success if all the positions available result in h(x,y) either equal to greater than current state cost.
- The hill climbing algorithm can be formally represented as:

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

  current ← MAKE-NODE(problem.INITIAL-STATE)
  loop do
    neighbor ← a highest-valued successor of current
    if neighbor.VALUE ≤ current.VALUE then return current.STATE
    current ← neighbor
```

Fig 4.2 Artificial Intelligence: A Modern Approach

- **Hill Climbing with Random Restart:**
  - Hill climbing can often get stuck and fail to find the state where h(x, y) = 0.
  - The solution to this problem is to start the search again from randomly placed queens on the board.
  - The number of times the program is going to restart until it finds a solution can be specified.
  - Given enough iterations the probability of Hill Climbing with Random Restart finding a solution approximated to one.

3. The Min Conflict Algorithm (Constrained Satisfaction Algorithm)

- **Constraints:** Each queen in the n–queens problem is under a constraint that the number of conflicts (Attacks) should minimize to zero in the goal state.
- **Heuristic Function h(q, x, y):** The heuristic is defined for each queen 'q' at position(x,y) on the board as the number of queens in conflict with the queen q.
- At each iteration, a queen under constraint is randomly selected. The all the possible h(q, x, y) is computed for queen 'q'. If there exists 'h' such that it is less than or equal to current cost heuristic, then such the queen is moved to that location (x, y) for such 'h'. The Ties are broken randomly.
- The Min Conflict Algorithm can be formally described as:

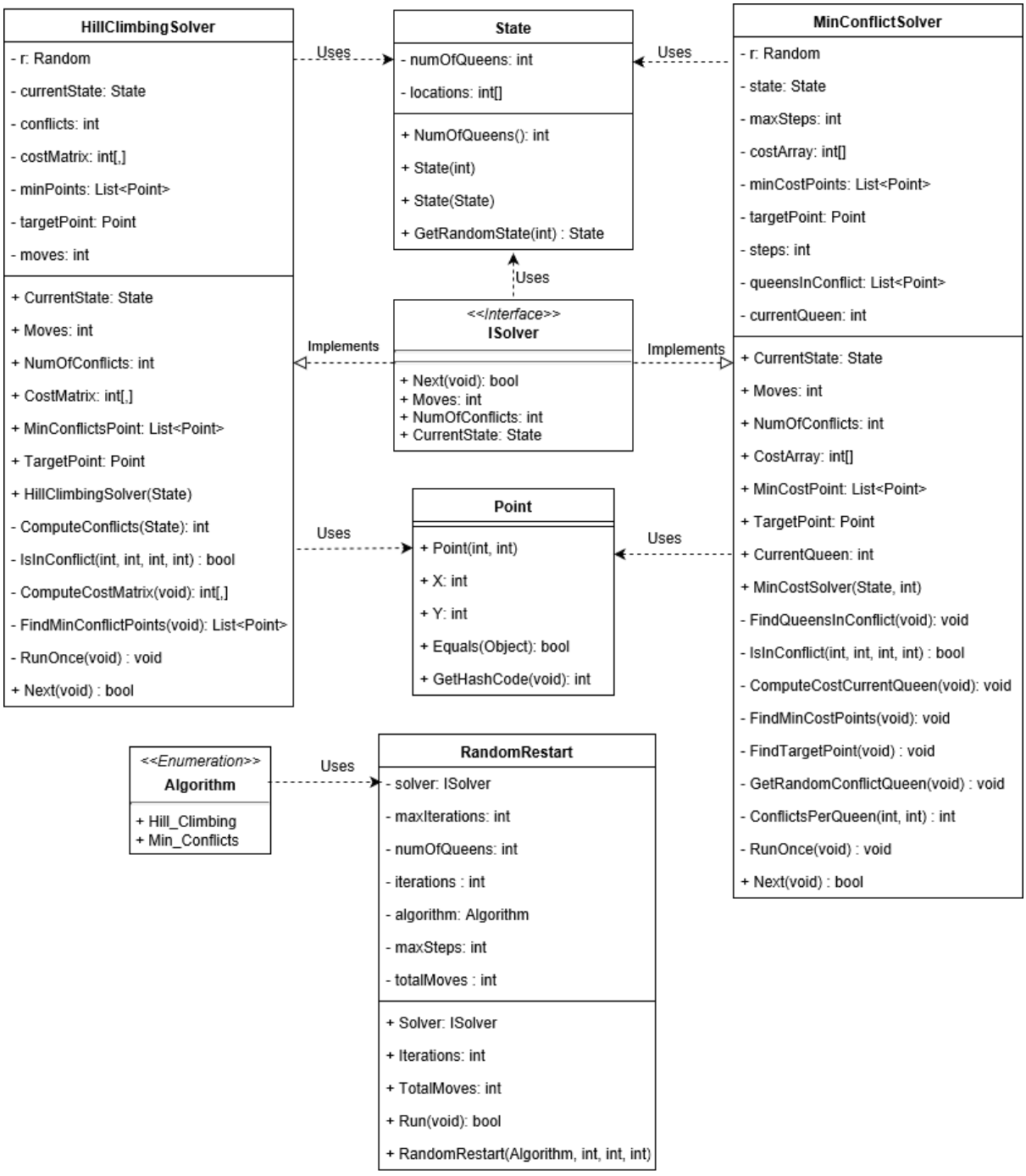
```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
          max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for csp
  for i = 1 to max_steps do
    if current is a solution for csp then return current
    var ← a randomly chosen conflicted variable from csp.VARIABLES
    value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
    set var = value in current
  return failure
```

Figure 6.8 *Artificial Intelligence: A Modern Approach*

4. The Program Structure

Programming Language: C#
IDE: Microsoft Visual Studio Community 2015

Class Diagram:



nQueens – Root Project

- **Program.cs –** Entry point of the application
- **ISolver.cs –** Interface implemented by Hill Climbing and Min Conflict Solvers.
- **State.cs –** Class to describe a state of n-queens problem.
- **Point.cs** – Class to represent a point in 2D.
- **HillClimbingSolver.cs** – Implementation logic for Hill Climbing algorithm.
- **MinConflictSolver.cs –** Implementation logic for Minimum Conflict algorithm.
- **RandomRestart.cs** – Class for implementing random restarts on HillClimbingSolver and MinConflictSolver
- **MainForm.cs** – A Graphical User Interface form and its logic.
- **AboutBox.cs –** An about window for the application.

Heuristic Functions

- **HillClimbingSolver**
  - `private bool isInConflict(int x1, int y1, int x2, int y2)`
    Given the coordinates of two points this function checks whether they are in conflict of not i.e. the queens are attacking each other or not.

  - `private int ComputeConflicts(State state)`
    Given a state this function will compute total number of conflicts in that state i.e. the total number of queens attacking each other.

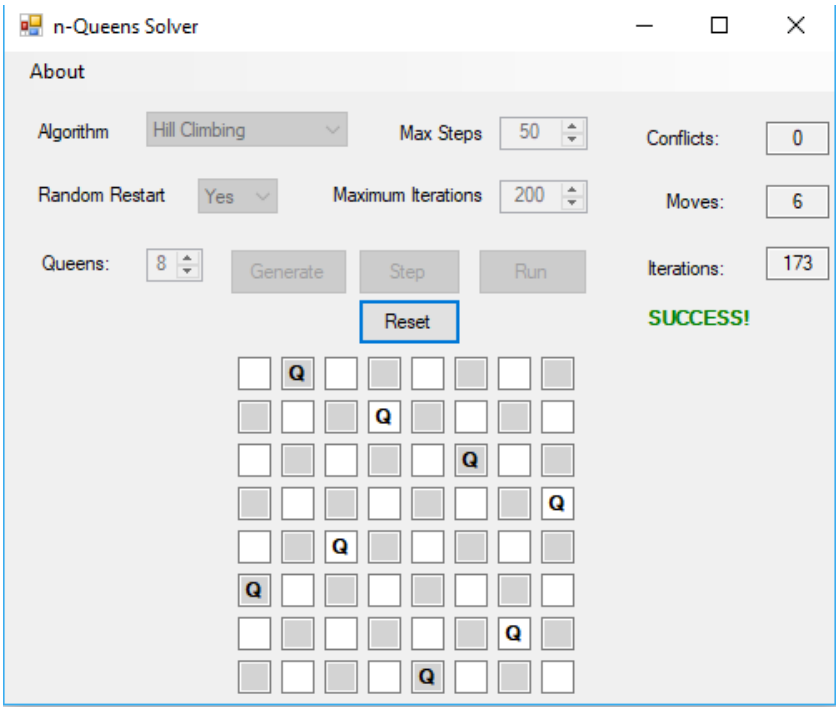
  - `private int[,] ComputeCostMatrix()`
    This function will return a n x n cost matrix where each cell represents the number of conflicts that would result from moving the queen into that position.

- **MinConflictSolver**
  - `private bool isInConflict(int x1, int y1, int x2, int y2)`
    Given the coordinates of two points this function checks whether they are in conflict of not i.e. the queens are attacking each other or not.

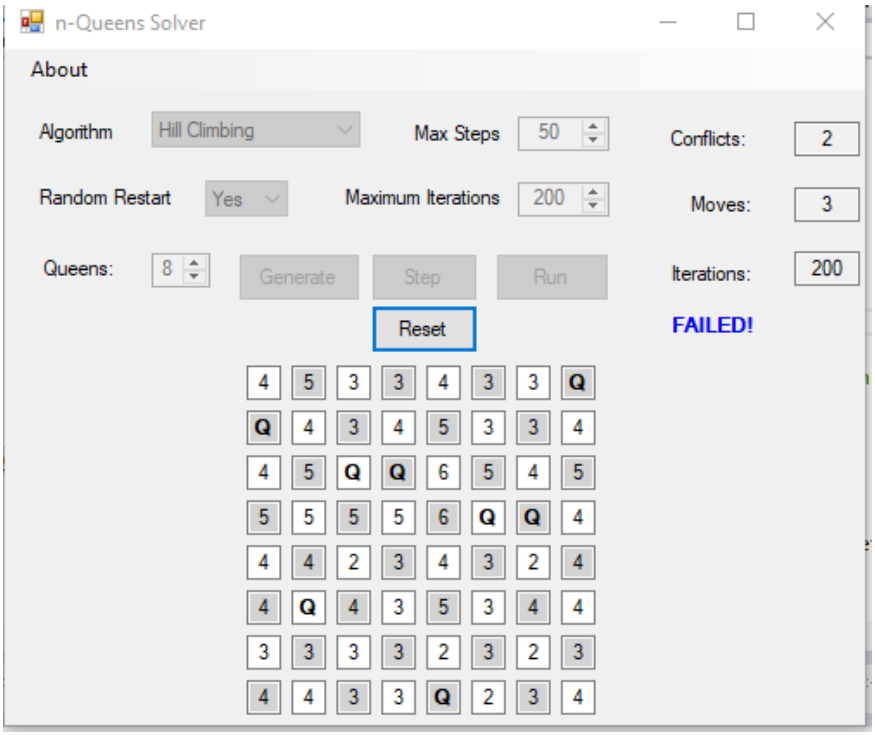
  - `private int ConflictsPerQueen(int x, int y)`
    Return the number of queens that are attacking the queen at location (x, y)

  - `private void ComputeCostCurrentQueen()`
    This function will compute the cost array for currently selected queen (randomly selected constraint variable). The cost array represents the number of conflicts the queen will be in if moved in its column.

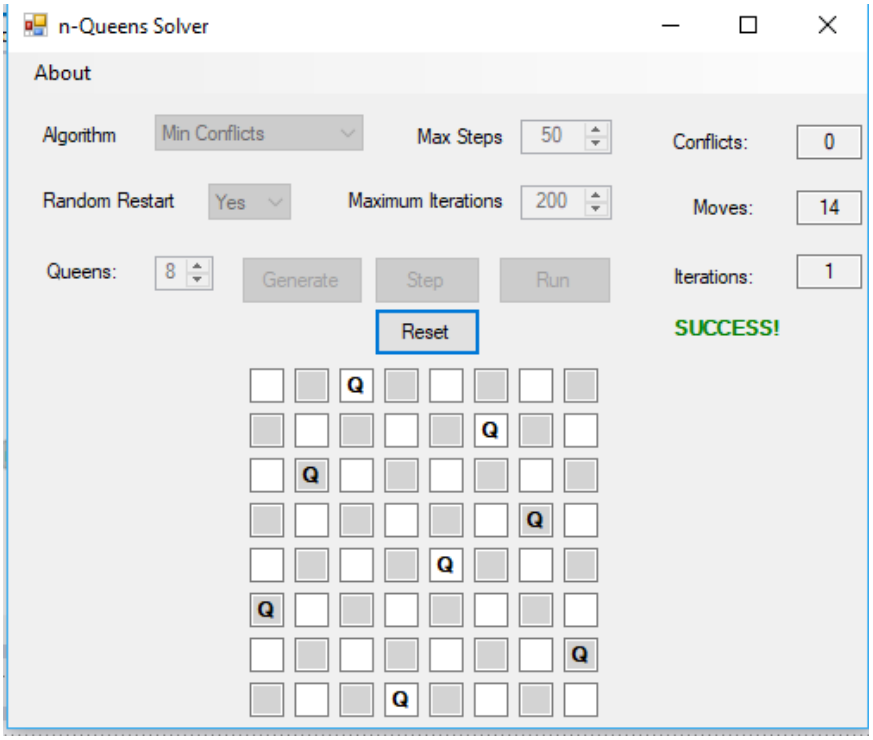
5. Results



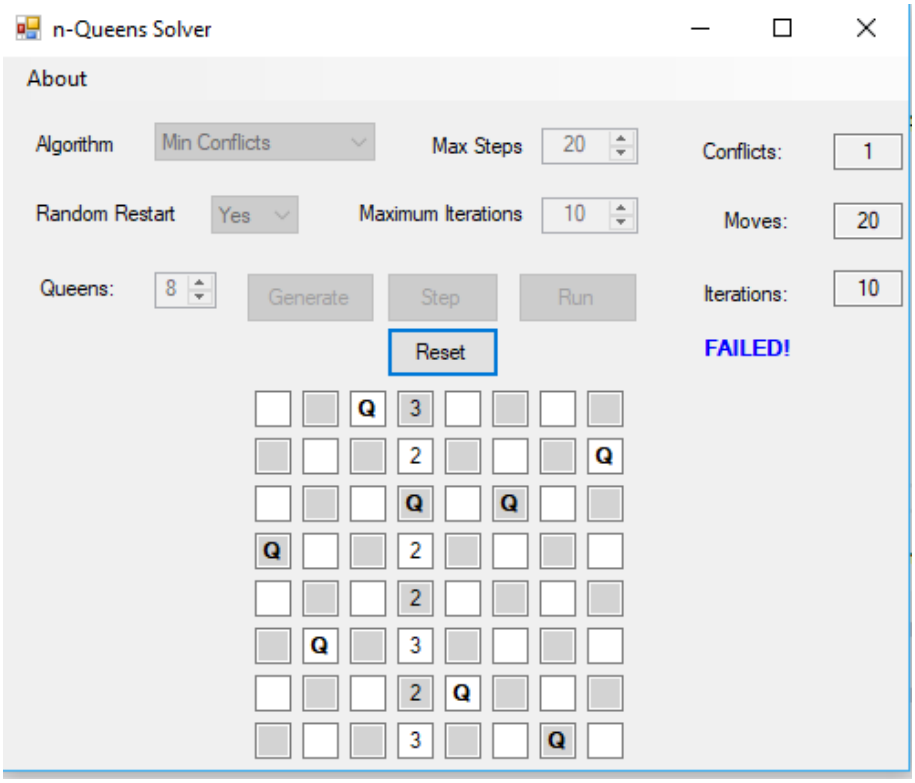
Hill Climbing with random restart, 8-queens, Success



Hill Climbing with random restart, 8-queens, Fail



Min Conflicts with random restart, 8-queens, Success



Min Conflicts with random restart, 8-queens, Fail

COMPARISON OF SUCCESS BETWEEN HILL CLIMBING WITH RANDOM RESTART AND MIN CONFLICTS WITH RANDOM RESTART (MAX MOVES 50, MAX ITERATIONS 500)

QUEENS	HILL CLIMBING		MIN CONFLICT	
	MOVES	ITERATIONS	MOVES	ITERATIONS
8	1	4	7	1
8	5	191	15	82
8	3	162	40	90
8	3	60	17	118
8	4	126	36	30
8	5	81	15	1
AVG	3.5	104	21.6	53.6

QUEENS	HILL CLIMBING		MIN CONFLICT	
	MOVES	ITERATIONS	MOVES	ITERATIONS
10	6	204	12	1
10	5	190	35	73
10	5	62	31	20
10	5	207	41	97
10	6	1	22	47
10	3	148	37	1
AVG	5	135.3	29.6	39.8

QUEENS	HILL CLIMBING		MIN CONFLICT	
	MOVES	ITERATIONS	MOVES	ITERATIONS
20	9	33	31	6
20	8	6	37	26
20	13	99	40	46
20	12	26	32	1
20	11	62	39	19
20	11	18	48	15
AVG	10.6	40.6	37.8	18.8

CONCLUSION

MIN CONFLICT ALGORITHM ON AVERAGE TAKES LESS NUMBER OF ITERATIONS THAN HILL CLIMBING ALGORITHM TO FIND THE SOLUTION THEREFORE IS MORE EFFICIENT.

HILL CLIMBING ALGORITHM ON OTHER HAND REQUIRES LESS NUMBER OF MOVES TO REACH THE SOLUTION THAN MIN CONFLICT WHEN THEY ACTUALLY FIND THE SOLUTION.