Programming Project 2
Solving N-queens problem using hill-climbing and its variants

Team Member: Shubhangi Alande (801104235)

N-Queen:

Problem Formulation: Consider an N × N chessboard. Place N queens on the board such that no two queens are attacking each other. The queen is the most powerful piece in chess and can attack from any distance horizontally, vertically, or diagonally. Thus, a solution must place the queens such that no two queens are in the same row, the same column, or along the same diagonal. Here we have implemented hill climbing search algorithms to solve the n-queens problem. The problem formulation for the n-queens problem can be described as below:

1. Initial State:
   Incremental Formulation: This method starts with placing the n queens one by one on the board.
   Complete-State Formulation: This method starts with placing all the n queens placed randomly on the board at once.

2. Goal State: n queens on the board with no two queens in the same row, column or the diagonal.

-

	Q		**Q**
Q		**Q**	

Initial State

	Q		
			Q
Q			
		Q	

Goal State

3. States: Any arrangement of 0 to n queens on the board is a state.
4. Actions: Adding a queen to an empty square of the board.
5. Transition Model: A board with the chosen queen added to the specified square.

Hill Climbing Search: For n-queens, hill climbing search starts with randomly placed n queens on an n * n board. The number successors for any state are given by the

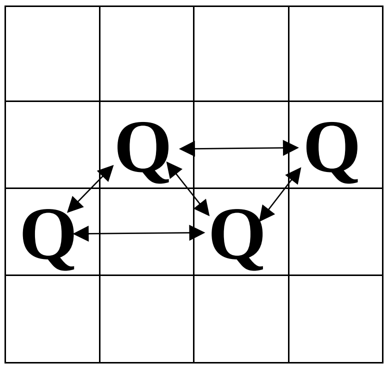
Number of successors = n * (n – 1)

These successors are derived by moving each queen on the board to its either up, left, down or right square. The successor with lowest heuristic is chosen.

Heuristic Function: The heuristic function calculates the number of conflicts in each state. The best state to be chosen next for evaluation is the state with the lower number of conflicts than the current state. Conflict is no of pairs of queens that can attack each other. For example, for the initial state given below heuristic value h =5

Programming Project 2
Solving N-queens problem using hill-climbing and its variants

Team Member: Shubhangi Alande (801104235)



Goal state is when h=0 for any successor state in any iteration. The heuristic value of the successor state must always be less than the current state. In case of the multiple successor states with the same least heuristic value, the first state out of all is selected for expansion. However, the problem of 'plateau' can arise in this approach, search space is flat, or sufficiently flat that the value returned by the target function is indistinguishable from the value returned for nearby regions due to the precision used by the machine to represent its value. In such cases, the hill climber may not be able to determine in which direction it should step and may wander in a direction that never leads to improvement. We solve this problem with a variant of steepest ascent hill climbing.

Sideways Moves: In this approach, the algorithm proceeds further after getting stuck at the Local Maxima (Plateau). It selects any successor whose heuristic value is equal to the current state node. Because of this another problem of infinite iterations may rise but we can put limit on the number of sideway moves allowed.

Random Restart: Another way of solving the local maxima problem involves repeated explorations of the problem space. Random-restart hill-climbing conducts a series of hill-climbing searches from randomly generated initial states, running each until it halts or makes no progress. This enables comparison of many optimization trials and finding a most optimal solution thus becomes a question of using enough iterations on the data.

Random Restart with Sideway Moves: Like the above variant with sideway moves, this algorithm allows sideway moves for random restart variant of hill climbing search. With random restart hill climbing, the probability of getting goal state tends to be equal to 1. Thus, this gives a much higher success rate than the basic version of hill climbing search algorithm.

Program Structure: The value of no of queens in N-queen problem is taken from the user (it should be greater than 3). The user can select from any of the four variants of the hill climbing search. All the algorithms reports the number of steps taken to find the solution and the number of iterations the algorithm completes for both success and failure. After running the algorithm, the average success and failure rates gets calculated and displayed on the console to the user. The details about the important classes, functions and variables is given below.

Classes:

1. **NqueenBoard:**

   Functions:

   -Getboardsize(): return user input n (# of queens)
   -Setgameboard(): randomly selects 1 of 3 randomly generated board
   -Calculateheuristic(int[,] tempboard): calculates the heuristic of the board sent as parameter. rows are first traversed then columns to check for conflicts horizontally and vertically, respectively. the logic to traverse both diagonal directions is split into the top-half (of the board), middle diagonal, and bottom-half (of the board).
   -Resetboard(): creates three random game boards. increments the number of the resets each time the method is called.
   -Boardsareequal(int[,] boardone, int[,] boardtwo): checks whether the two boards sent as parameters are the same.

   Variables:
   - `_numberOfQueens`: This value is the number of queens. It is taken from the user and must be greater than 3 as the solution for n-queens problem exists for all n greater than 3.
   - `_gameBoard`: This initializes a n * n board to start the algorithm.
   - _QUEEN, _NOT_QUEEN: useful to reset the board to a new initial state.
   - `_regularMoves`, `_sidewaysMoves`: These variables are used to store the values of steps taken in each iteration.

2. **SteepestAscent:**

   Functions:

   - RunAlgorithm(): until the heuristic is 0 and neighbours with a lower heuristic value is found, repeat the following:
     -find & store all possible neighbours with h < h of current board
     -select random neighbour
     -update game board
     -if game board cannot be updated and h != 0, results in failure

Programming Project 2
Solving N-queens problem using hill-climbing and its variants
Team Member: Shubhangi Alande (801104235)

Variables:
- `_totalNumberOfStepsSucceeds`: to store the total number of steps taken to find the solution to the problem.
- `_numberOfSuccessfulIterations`: to store the total number of iterations performed by the algorithm to find the solution.
- `_totalNumberOfStepsFails`: to store the total number of steps taken to find the solution to the problem when it fails to find the solution.
- `_numberOfFailedIterations`: to store the total number of iterations performed by the algorithm to find the solution to the problem when it fails to find.

3. **SidewaysMove:**

functions:

- RunAlgorithm(): until the heuristic is 0 and neighbours with a lower heuristic value is found, repeat the following:

  -find & store all possible neighbours (h <= h of current board)
  -select random neighbour
  -update game board
  -if number of consecutive sideways moves exceeds the limit (of 100) and a non-sideways move cannot be found, results in failure

Variables:
- _consecutiveSidewaysMoves: to store of the number of sideway moves taken.
- _LIMIT_CONSECUTIVE_SIDEWAYS_MOVES: limit for no of sideways moves to be taken.
-
-

4. **RandomRestart:**

functions:

-RunAlgorithmWithoutSideways(): restarting the game board (by calling resetboard()) any time the board is 'stuck' at h > 0

-RunAlgorithmWithSideways(): restarting the game board (by calling resetboard()) any time the board is 'stuck' at h > 0

Programming Project 2
Solving N-queens problem using hill-climbing and its variants
Team Member: Shubhangi Alande (801104235)

---**Execution Results**---

1. **Steepest-Ascent Hill Climbing Algorithm:**

```
Number of Queens: 8

Number of Iterations: 347

Success/Fail Analysis

Success Rate: 51.41

Failure Rate: 122.59

Average Number of Steps When It Succeeds: 6

Average Number of Steps When It Fails: 5

-----------------------------------

Starting Board: 4

Current State (Heuristic): 4

0 0 1 0 0 0 0 0

0 1 0 0 0 0 0 1

0 0 0 1 1 0 0 0

0 0 0 0 0 0 1 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 1 0 0

0 0 0 0 0 0 0 0


Step No.1

Current State (Heuristic): 2

0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 1
```

```
0 0 0 1 1 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 0 1 0 0
0 0 0 0 0 0 0 0
Failure!
Current State (Heuristic): 2
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 1
0 0 0 1 1 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 0 1 0 0
0 0 0 0 0 0 0 0
Total Steps Taken: 1
```

2. Hill Climbing Sideways Moves Algorithm

```
Number of Queens: 8
Number of Iterations: 396
Success/Fail Analysis
Success Rate: 126.9
Failure Rate: 47.1
Average Number of Steps When It Succeeds: 45
Average Number of Steps When It Fails: 102
```

Programming Project 2
Solving N-queens problem using hill-climbing and its variants

Team Member: Shubhangi Alande (801104235)

```
-----------------------------------

Step No.1

Current State (Heuristic): 5

0 0 0 1 0 0 0 0

0 1 0 0 0 1 0 1

0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 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


Step No.2

Current State (Heuristic): 5

0 0 0 1 0 0 0 0

0 1 0 0 0 1 0 0

0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 1

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0

Sideways Move!


Step No.3

Current State (Heuristic): 5
```

```
0 0 0 1 0 0 0 0

0 1 0 0 0 1 1 0

0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 0

Sideways Move!


Step No.4

Current State (Heuristic): 5

0 0 0 1 0 0 0 0

0 1 0 0 0 0 1 0

0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0 1 0 0

Sideways Move!


Step No.5

Current State (Heuristic): 5

0 0 0 1 0 0 0 0

0 1 0 0 0 0 1 0
```

```
0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0

0 0 0 0 1 1 0 0

0 0 1 0 0 0 0 0

Sideways Move!


Step No.6

Current State (Heuristic): 4

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 1 0 0 0 0 0 0

0 0 0 0 1 1 0 0

0 0 1 0 0 0 0 0


Step No.7

Current State (Heuristic): 4

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 0 0 1 0 0 1
```

```
0 1 0 0 0 0 0 0

0 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0

Sideways Move!


Step No.8

Current State (Heuristic): 2

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 1

0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0


Step No.9

Current State (Heuristic): 2

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 0

1 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0
```

```
Sideways Move!

Step No.10

Current State (Heuristic): 2

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 1 0 0 1 0 0

Sideways Move!


Step No.11

Current State (Heuristic): 2

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0

0 0 1 0 0 0 0 1

0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 0 0 0 1 0 0

Sideways Move!


Step No.12
```

```
Current State (Heuristic): 2

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 1 0 0 1 0 0

Sideways Move!


Step No.13

Current State (Heuristic): 2

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 0

1 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0

Sideways Move!


Step No.14

Current State (Heuristic): 2

0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
1 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
Sideways Move!

Step No.15
Current State (Heuristic): 2
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
1 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
Sideways Move!

Step No.16
Current State (Heuristic): 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 0 0 0 0 0 1

0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0


Success!

Total Steps Taken: 16
```

3. **Hill Climbing Random Restart Algorithm**

```
Number of Queens: 8

Number of Iterations: 108

Success/Fail Analysis

Success Rate: 137

Average Number of Steps When It Succeeds: 37

Average Number of Restarts When It Succeeds: 5


Step No.13

Current State (Heuristic): 2

0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0

0 1 1 0 0 0 0 0

0 0 0 0 0 1 0 0

0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0

0 0 0 1 1 0 0 0
```

```
0 0 0 0 0 0 0 0

Random Restart!


-----------------------------------

Starting Board

Current State (Heuristic): 5

0 0 0 0 0 0 0 0

0 0 1 0 0 1 0 0

0 0 0 0 0 0 1 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

1 0 0 0 0 0 0 0

0 0 0 1 0 0 0 1

0 0 0 0 0 0 0 0


Step No.14

Current State (Heuristic): 4

0 0 0 0 0 0 0 0

0 0 1 0 0 1 0 0

0 0 0 0 0 0 1 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

1 0 0 0 0 0 0 0

0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 1
```

```
Step No.15

Current State (Heuristic): 3

0 0 0 0 0 0 0 0

0 0 1 0 0 1 0 0

0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

1 0 0 0 0 0 0 0

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 1


Step No.16

Current State (Heuristic): 2

0 0 0 0 0 0 0 0

0 0 1 0 0 1 0 0

0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 1

1 0 0 0 0 0 0 0

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0


Step No.17

Current State (Heuristic): 1

0 0 0 0 0 0 0 1

0 0 1 0 0 1 0 0
```

Programming Project 2
Solving N-queens problem using hill-climbing and its variants
Team Member: Shubhangi Alande (801104235)

```
0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

1 0 0 0 0 0 0 0

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

Random Restart!

------------------------------------

Starting Board

Current State (Heuristic): 9

0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 1 1 0

1 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 1 1 1 0 0 1

0 0 0 0 0 0 0 0


Step No.18

Current State (Heuristic): 7

0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 1 1 0

1 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 1 1 1 0 0 0

0 0 0 0 0 0 0 1


Step No.19

Current State (Heuristic): 6

0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 1 1 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 1 1 1 0 0 0

0 0 0 0 0 0 0 0


Step No.20

Current State (Heuristic): 4

0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 1 0 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 1 1 1 0 0 0

0 0 0 0 0 0 1 0
```

```
Step No.21
Current State (Heuristic): 3
0 0 0 0 0 0 0 0
0 1 0 1 0 0 0 0
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 1 0 1 0 0 0
0 0 0 0 0 0 1 0

Step No.22
Current State (Heuristic): 2
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 1 0 1 0 0 0
0 0 0 0 0 0 1 0
Random Restart!
-----------------------------------
Starting Board
Current State (Heuristic): 11
0 0 0 0 0 0 0 0
```

```
0 1 0 0 0 0 0 1

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 1 0 1 0 0 0

0 0 0 0 0 0 0 0

0 0 0 1 0 1 1 0

0 0 0 0 0 0 0 0


Step No.23

Current State (Heuristic): 9

0 0 0 0 0 0 1 0

0 1 0 0 0 0 0 1

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 1 0 1 0 0 0

0 0 0 0 0 0 0 0

0 0 0 1 0 1 0 0

0 0 0 0 0 0 0 0


Step No.24

Current State (Heuristic): 8

0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 1

1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0

0 0 1 0 1 0 0 0
```

```
0 0 0 0 0 0 0 0

0 0 0 1 0 1 0 0

0 0 0 0 0 0 0 0


Step No.25

Current State (Heuristic): 6

0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 1

1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0

0 0 1 0 1 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 1 0 0

0 0 0 1 0 0 0 0


Step No.26

Current State (Heuristic): 3

0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 1

1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0

0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 1 0 0

0 0 0 1 0 0 0 0
```

```
Step No.27

Current State (Heuristic): 2

0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0

0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 0 1 0 0

0 0 0 1 0 0 0 0


Step No.28

Current State (Heuristic): 1

0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0

0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 0 1 0 0

1 0 0 1 0 0 0 0


Step No.29

Current State (Heuristic): 0

0 0 0 0 1 0 0 0

0 1 0 0 0 0 0 0
```

Programming Project 2
Solving N-queens problem using hill-climbing and its variants

Team Member: Shubhangi Alande (801104235)

```
0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 0 1 0 0

1 0 0 0 0 0 0 0


Success!

Total Steps Taken: 29

-----------------------------------

Starting Board

Current State (Heuristic): 10

0 0 0 0 0 0 1 0

1 0 0 0 0 1 0 1

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 1 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0


Step No.1

Current State (Heuristic): 8

0 0 0 1 0 0 1 0

1 0 0 0 0 1 0 1

0 0 0 0 0 0 0 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 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0


Step No.2

Current State (Heuristic): 7

0 0 0 1 0 0 1 0

1 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 0 1 0 0 0

0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0


Step No.3

Current State (Heuristic): 6

0 0 0 1 0 0 1 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 0 1 0 0 0

0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 0
```

```
1 0 0 0 0 0 0 0


Step No.4

Current State (Heuristic): 4

0 0 0 1 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 0 1 0 0 0

0 0 1 0 0 0 0 0

0 0 0 0 0 0 1 0

1 0 0 0 0 0 0 0


Step No.5

Current State (Heuristic): 3

0 0 0 1 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 0 0 0 0 0

0 0 1 0 1 0 0 0

0 0 0 0 0 0 1 0

1 0 0 0 0 0 0 0


Step No.6

Current State (Heuristic): 2
```

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

0 0 0 0 1 0 0 0

0 0 0 0 0 0 1 0

1 0 0 0 0 0 0 0


Step No.7

Current State (Heuristic): 1

0 0 1 0 0 0 0 0

0 0 0 0 0 1 0 1

0 0 0 1 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 0 0 0 0 1 0

1 0 0 0 0 0 0 0


Step No.8

Current State (Heuristic): 0

0 0 1 0 0 0 0 0

0 0 0 0 0 1 0 0

0 0 0 1 0 0 0 0

0 1 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 1

0 0 0 0 1 0 0 0

0 0 0 0 0 0 1 0

1 0 0 0 0 0 0 0

Success!

Total Steps Taken: 8
```

4. **Hill Climbing Random Restart Algorithm with sideways move:**

```
Number of Queens: 8

Number of Iterations: 409

Success/Fail Analysis

Success Rate: 100

Average Number of Steps When It Succeeds: 59

Average Number of Restarts When It Succeeds: ~ 1


Starting Board

Current State (Heuristic): 4

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 1

0 0 0 0 0 1 0 0

0 0 0 0 0 0 0 0

1 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0
```

Programming Project 2
Solving N-queens problem using hill-climbing and its variants
Team Member: Shubhangi Alande (801104235)

```
Step No.1

Current State (Heuristic): 4

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 1 0 0 0

0 0 1 0 0 0 0 1

0 0 0 0 0 1 0 0

1 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

Sideways Move!


Step No.2

Current State (Heuristic): 4

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0

0 0 1 0 0 0 0 1

0 0 0 0 0 1 0 0

1 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!


Step No.3

Current State (Heuristic): 3
```

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

Step No.4
Current State (Heuristic): 2
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0

Step No.5
Current State (Heuristic): 2
0 1 0 1 0 0 0 0
0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1
```

```
0 0 0 0 0 1 0 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!


Step No.6

Current State (Heuristic): 1

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 0 1 0 0

1 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0


Step No.7

Current State (Heuristic): 1

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0

1 0 0 0 0 0 0 0
```

```
0 0 0 0 1 0 0 0

Sideways Move!


Step No.8

Current State (Heuristic): 1

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 0

1 0 1 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!


Step No.9

Current State (Heuristic): 1

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 0 0 0 0 0 1

0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0

0 0 1 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!
```

```
Step No.10

Current State (Heuristic): 1

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 1

0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!


Step No.11

Current State (Heuristic): 1

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 1

1 0 0 0 0 0 0 0

0 0 1 0 0 0 0 0

0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!


Step No.12
```

```
Current State (Heuristic): 1

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 1

0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!


Step No.13

Current State (Heuristic): 1

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 0 1 0 0

0 1 0 0 0 0 0 1

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!


Step No.14

Current State (Heuristic): 1

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 1

0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!


Step No.15

Current State (Heuristic): 1

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 0 0 0 0 0 1

0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0

0 0 1 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!


Step No.16

Current State (Heuristic): 1

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 1

0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!


Step No.17

Current State (Heuristic): 1

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 0 1 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 1

Sideways Move!


Step No.18

Current State (Heuristic): 1

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 1

1 0 0 0 0 0 0 0

0 0 1 0 0 0 0 0

0 0 0 0 0 1 0 0
```

```
0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!


Step No.19

Current State (Heuristic): 1

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 0 1 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 1 0 0 0

Sideways Move!


Step No.20

Current State (Heuristic): 1

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 0 0 0 0

0 1 0 0 0 1 0 0

0 0 0 0 0 0 0 1
```

```
0 0 0 0 1 0 0 0

Sideways Move!


Step No.21

Current State (Heuristic): 1

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 0 1 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 1

0 0 0 0 1 0 0 0

Sideways Move!


Step No.22

Current State (Heuristic): 1

0 0 0 1 0 0 0 0

0 0 0 0 0 0 1 1

1 0 0 0 0 0 0 0

0 0 1 0 0 0 0 0

0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

Sideways Move!
```

Programming Project 2
Solving N-queens problem using hill-climbing and its variants
Team Member: Shubhangi Alande (801104235)

```
Step No.23

Current State (Heuristic): 0

0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0

0 0 1 0 0 0 0 0

0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0

0 0 0 0 0 0 1 0

0 0 0 0 1 0 0 0


Success!

Total Steps Taken: 23
```

Programming Project 2
Solving N-queens problem using hill-climbing and its variants

Team Member: Shubhangi Alande (801104235)

---------------------------------Source Code-------------------------------------

```
using System;

namespace NQueenusingHillClimbing
{
    class Program
    {
        static void Main(string[] args)
        {
            string _formatter="##.##";
            Random rand = new Random();
            int _noOfIterations = rand.Next(500) + 100;
            int queen = 0;

            int[] steepestAscentResults = { 0, 0, 0, 0 };
            int[] Sideways = { 0, 0, 0, 0 };
            int[] Restart = { 0, 0, 0, 0 };
            int[] RestartWithSideWay = { 0, 0, 0, 0 };
            string UserDecision = string.Empty;

            do
            {
                while (queen <= 3)
                {
                    Console.Write("Enter the number of Queens (Should be more than 3):
");

                    queen = Convert.ToInt16(Console.ReadLine());
                }
                int choice = -1;
                Console.WriteLine("Select Algorithm to solve N-Queen problem ");
                Console.WriteLine("1. Steepest-Ascent");
                Console.WriteLine("2. Sideways Move");
                Console.WriteLine("3. Random Restart");
                Console.WriteLine("4. Random RestartT With Sideways Move");
                Console.Write("Select option= ");
                choice = Convert.ToInt16(Console.ReadLine());
                switch (choice)
                {
                    case 1:
                        for (int iteration = 0; iteration < _noOfIterations; iteration++)
                        {
                            SteepestAscent game = new SteepestAscent(queen);
                            int[] results = game.RunAlgorithm();
                            steepestAscentResults[0] += results[0];
                            steepestAscentResults[1] += results[1];
                            steepestAscentResults[2] += results[2];
                            steepestAscentResults[3] += results[3];
                        }

Console.WriteLine("\n*****************************************************************
******");
                        Console.WriteLine("Steepest-Ascent Hill Climbing Algorithm");
                        Console.WriteLine("Number Of Queens: " + queen);
                        Console.WriteLine("Number of Iterations: " + _noOfIterations);
                        Console.WriteLine("Success/Fail Analysis");
                        Console.WriteLine("Success Rate: {0}", ((steepestAscentResults[1]
* 100) / (float)_noOfIterations + '%').ToString(_formatter));
                        Console.WriteLine("Failure Rate: {0}", ((steepestAscentResults[3]
* 100) / (float)_noOfIterations + '%').ToString(_formatter));
                        if (steepestAscentResults[1] != 0)
                            Console.WriteLine("Average Number of Steps When It Succeeds:
" + (steepestAscentResults[0] / steepestAscentResults[1]));
                        if (steepestAscentResults[3] != 0)
                            Console.WriteLine("Average Number of Steps When It Fails: " +
(steepestAscentResults[2] / steepestAscentResults[3]));
                        //Console.ReadLine();
                        break;
                    case 2:
                        for (int iteration = 0; iteration < _noOfIterations; iteration++)
                        {
                            SidewaysMove game = new SidewaysMove(queen);
                            int[] results = game.RunAlgorithm();
                            Sideways[0] += results[0];
                            Sideways[1] += results[1];
                            Sideways[2] += results[2];
                            Sideways[3] += results[3];
                        }


Console.WriteLine("\n********************************************************************
***");
                        Console.WriteLine("Hill Climbing Sideways Moves Algorithm");
                        Console.WriteLine("Number of Queens: " + queen);
                        Console.WriteLine("Number of Iterations: " + _noOfIterations);
                        Console.WriteLine("Success/Fail Analysis");
                        Console.WriteLine("Success Rate: {0}", ((Sideways[1] * 100) /
(float)_noOfIterations + '%').ToString(_formatter));
                        Console.WriteLine("Failure Rate: {0}", ((Sideways[3] * 100) /
(float)_noOfIterations + '%').ToString(_formatter));

                        if (Sideways[1] != 0)
                            Console.WriteLine("Average Number of Steps When It Succeeds:
" + (Sideways[0] / Sideways[1]));
                        if (Sideways[3] != 0)
                            Console.WriteLine("Average Number of Steps When It Fails: " +
(Sideways[2] / Sideways[3]));
                        //Console.ReadLine();
                        break;

                    case 3:
                        for (int iteration = 0; iteration < _noOfIterations; iteration++)
                        {
                            RandomRestart game = new RandomRestart(queen);
                            int[] results = game.RunAlgorithmWithoutSideways();
                            Restart[0] += results[0];
                            Restart[1] += results[1];
                            Restart[2] += results[2];
                        }


Console.WriteLine("\n******************************************************************
**********");
                        Console.WriteLine("Hill Climbing Random Restart Algorithm");
                        Console.WriteLine("Number of Queens: " + queen);
                        Console.WriteLine("Number of Iterations: " + _noOfIterations);
                        Console.WriteLine("Success/Fail Analysis");
                        Console.WriteLine("Success Rate: {0}", ((Restart[1] * 100) /
(float)_noOfIterations + '%').ToString(_formatter));

                        Console.WriteLine("Average Number of Steps When It Succeeds: " +
(Restart[0] / _noOfIterations));
                        Console.WriteLine("Average Number of Restarts When It Succeeds: "
+ (Restart[2] / _noOfIterations));
                        //Console.ReadLine();
                        break;

                    case 4:
                        for (int iteration = 0; iteration < _noOfIterations; iteration++)
                        {
                            RandomRestart game = new RandomRestart(queen);
                            int[] results = game.RunAlgorithmWithSideways();
                            RestartWithSideWay[0] += results[0];
                            RestartWithSideWay[1] += results[1];
                            RestartWithSideWay[2] += results[2];
                        }


Console.WriteLine("\n******************************************************************
*********");
                        Console.WriteLine("Hill Climbing Random Restart Algorithm with
sideways move");
                        Console.WriteLine("Number of Queens: " + queen);
                        Console.WriteLine("Number of Iterations: " + _noOfIterations);
                        Console.WriteLine("Success/Fail Analysis");
                        Console.WriteLine("Success Rate: {0}", ((RestartWithSideWay[1] *
100) / (float)_noOfIterations + '%').ToString(_formatter));

                        Console.WriteLine("Average Number of Steps When It Succeeds: " +
(RestartWithSideWay[0] / _noOfIterations));
                        if ((RestartWithSideWay[2] / (float)_noOfIterations) > 0 &&
(RestartWithSideWay[2] / (float)_noOfIterations) < 1)
                            Console.WriteLine("Average Number of Restarts When It
Succeeds: ~ 1");
                        else
                            Console.WriteLine("Average Number of Restarts When It
Succeeds: " + (RestartWithSideWay[2] / (float)_noOfIterations));
                        //Console.ReadLine();
                        break;
```

```
                }
                Console.WriteLine("Do You want to continue with other algorithm?  Yes or 
No");

                UserDecision = Console.ReadLine().ToUpper();

            } while (UserDecision != "NO");
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NQueenusingHillClimbing
{
    public class SteepestAscent: NqueenBoard
    {
        private int _totalNumberOfStepsSucceeds = 0;
        private int _numberOfSuccessfulIterations = 0;
        private int _totalNumberOfStepsFails = 0;
        private int _numberOfFailedIterations = 0;
        private Boolean _boardChanged = true;

        public SteepestAscent(int numOfQueens) : base(numOfQueens)
        {

        }

        /// <summary>
        /// calls until the heuristic is 0 and neighbours with a lower heuristic value is 
found. Fails if h=0 not found.
        /// </summary>
        /// <returns>Success / Failure analysis</returns>
        public int[] RunAlgorithm()
        {
            this.StartGameBoard();
            this._boardChanged = true;
            int currentStateHeuristic = this.CalculateHeuristic(this._gameBoard);
            List<int[,]> possibleMoves = new List<int[,]>();
            this.ResetRegularMoves();

            while (this.CalculateHeuristic(this._gameBoard) != 0 && (this._boardChanged 
== true))
            {
                int columnNumber = 0;
                int[,] possibleMove = new int[this.GetBoardSize(),this.GetBoardSize()];
                for (int i = 0; i < this.GetBoardSize(); i++)
                {
                    for (int j = 0; j < this.GetBoardSize(); j++)
                    {
                        possibleMove[i,j] = this._gameBoard[i,j];
                    }
                }

                currentStateHeuristic = this.CalculateHeuristic(this._gameBoard);
                this._boardChanged = false;

                while (columnNumber < this.GetBoardSize())
                {
                    int currentColumnQueenPosition = -1;

                    for (int i = 0; i < this.GetBoardSize(); i++)
                    {
                        if (possibleMove[i,columnNumber] == this._QUEEN)
                            currentColumnQueenPosition = i;
                        possibleMove[i,columnNumber] = this._NOT_QUEEN;
                    }

                    for (int i = 0; i < this.GetBoardSize(); i++)
                    {
                        possibleMove[i,columnNumber] = this._QUEEN;

                        int[,] newMove = new
int[this.GetBoardSize(),this.GetBoardSize()];
                        for (int k = 0; k < this.GetBoardSize(); k++)
                        {
                            for (int j = 0; j < this.GetBoardSize(); j++)
                            {
                                newMove[k,j] = possibleMove[k,j];
                            }
                        }
                        if (this.CalculateHeuristic(this._gameBoard) >
this.CalculateHeuristic(newMove))
                            possibleMoves.Add(newMove);
                        possibleMove[i,columnNumber] = this._NOT_QUEEN;
                    }

                    possibleMove[currentColumnQueenPosition,columnNumber] = this._QUEEN;

                    columnNumber += 1;
                }

                Random rand = new Random();

                if (possibleMoves.Count != 0)
                {
                    int pick = rand.Next(possibleMoves.Count);

                    int minHeuristic = currentStateHeuristic;

                    if (minHeuristic > this.CalculateHeuristic(possibleMoves[pick]))
                    {
                        minHeuristic = this.CalculateHeuristic(possibleMoves[pick]);
                        this._gameBoard = this.CopyBoard(possibleMoves[pick]);
                        this._boardChanged = true;

                        this._regularMoves += 1;
                        Console.WriteLine("\nStep No." + _regularMoves);
                        this.PrintGameBoard();
                        possibleMoves.Clear();
                    }
                }
                else
                {
                    if (possibleMoves.Count == 0)
                    {
                        this._boardChanged = false;
                        possibleMoves.Clear();
                    }
                }
            }

            if (this.CalculateHeuristic(this._gameBoard) == 0)
            {
                Console.WriteLine("\nSuccess!");
                this._totalNumberOfStepsSucceeds += this._regularMoves;
                this._numberOfSuccessfulIterations += 1;
                Console.WriteLine("Total Steps Taken: " + this._regularMoves);
            }
            else
            {
                Console.WriteLine("\nFailure!");
                this.PrintGameBoard();
                this._totalNumberOfStepsFails += this._regularMoves;
                this._numberOfFailedIterations += 1;
                Console.WriteLine("Total Steps Taken: " + this._regularMoves);
            }

            return new int[] { _totalNumberOfStepsSucceeds,
_numberOfSuccessfulIterations, _totalNumberOfStepsFails,
                _numberOfFailedIterations };
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NQueenusingHillClimbing
{
    public class SidewaysMove : NqueenBoard
    {
        private int _totalNumberOfStepsSucceeds = 0;
        private int _numberOfSuccessfulIterations = 0;
        private int _totalNumberOfStepsFails = 0;
        private int _numberOfFailedIterations = 0;
        private Boolean _boardChanged = true;
        private int _consecutiveSidewaysMoves = 0;
        private readonly int LIMIT_CONSECUTIVE_SIDEWAYS_MOVES = 100;

        public SidewaysMove(int numOfQueens): base(numOfQueens)
        {

        }

        /// <summary>
        /// call until the heuristic is 0 and neighbours with a lower heuristic value is 
found
        /// </summary>
        /// <returns>success/ Failure analysis</returns>
        public int[] RunAlgorithm()
        {
            this.StartGameBoard();
            this._boardChanged = true;
            int currentStateHeuristic = this.CalculateHeuristic(this._gameBoard);
            List<int[,]> possibleMoves = new List<int[,]>();
            this.ResetRegularMoves();

            while (this.CalculateHeuristic(this._gameBoard) != 0 && (this._boardChanged 
== true))
            {
                Boolean moveMade = false;
                int columnNumber = 0;
                int[,] possibleMove = new int[this.GetBoardSize(),this.GetBoardSize()];
                for (int i = 0; i < this.GetBoardSize(); i++)
                {
                    for (int j = 0; j < this.GetBoardSize(); j++)
                    {
                        possibleMove[i,j] = this._gameBoard[i,j];
                    }
                }

                currentStateHeuristic = this.CalculateHeuristic(this._gameBoard);
                this._boardChanged = false;

                while (columnNumber < this.GetBoardSize())
                {
                    int currentColumnQueenPosition = -1;

                    for (int i = 0; i < this.GetBoardSize(); i++)
                    {
                        if (possibleMove[i,columnNumber] == this._QUEEN)
                            currentColumnQueenPosition = i;
                        possibleMove[i,columnNumber] = this._NOT_QUEEN;
                    }

                    for (int i = 0; i < this.GetBoardSize(); i++)
                    {
                        possibleMove[i,columnNumber] = this._QUEEN;

                        int[,] newMove = new
int[this.GetBoardSize(),this.GetBoardSize()];
                        for (int k = 0; k < this.GetBoardSize(); k++)
                        {
                            for (int j = 0; j < this.GetBoardSize(); j++)
                            {
                                newMove[k,j] = possibleMove[k,j];
                            }
                        }
                        if (this.CalculateHeuristic(this._gameBoard) >=
this.CalculateHeuristic(newMove) && this.BoardsAreEqual(this._gameBoard, newMove) ==
false)
                            possibleMoves.Add(newMove);
                        possibleMove[i,columnNumber] = this._NOT_QUEEN;
                    }

                    possibleMove[currentColumnQueenPosition,columnNumber] = this._QUEEN;

                    columnNumber += 1;
                }

                Random rand = new Random();
                int minHeuristic = currentStateHeuristic;

                if (possibleMoves.Count != 0)
                {
                    int pick = rand.Next(possibleMoves.Count);

                    if (minHeuristic > this.CalculateHeuristic(possibleMoves[pick]))
                    {
                        minHeuristic = this.CalculateHeuristic(possibleMoves[pick]);
                        this._gameBoard = this.CopyBoard(possibleMoves[pick]);
                        this._boardChanged = true;
                        this._regularMoves += 1;
                        this._consecutiveSidewaysMoves = 0;
                        Console.WriteLine("\nStep No." + _regularMoves);
                        this.PrintGameBoard();
                        possibleMoves.Clear();
                    }
                    else if (minHeuristic == this.CalculateHeuristic(possibleMoves[pick])
&&
                          this._consecutiveSidewaysMoves <
this.LIMIT_CONSECUTIVE_SIDEWAYS_MOVES)
                    {
                        minHeuristic = this.CalculateHeuristic(possibleMoves[pick]);
                        this._gameBoard = this.CopyBoard(possibleMoves[pick]);
                        this._boardChanged = true;
                        this._consecutiveSidewaysMoves += 1;
                        this._regularMoves += 1;
                        Console.WriteLine("\nStep No." + _regularMoves);
                        this.PrintGameBoard();
                        Console.WriteLine("Sideways Move!");
                        possibleMoves.Clear();
                    }
                }
                else
                {
                    this._boardChanged = false;
                }
            }

            if (this.CalculateHeuristic(this._gameBoard) == 0)
            {
                Console.WriteLine("\nSuccess!");
                this._totalNumberOfStepsSucceeds += this._regularMoves;
                this._numberOfSuccessfulIterations += 1;
                Console.WriteLine("Total Steps Taken: " + this._regularMoves);
            }
            else
            {
                Console.WriteLine("\nFailure!");
                this.PrintGameBoard();
                this._totalNumberOfStepsFails += this._regularMoves;
                this._numberOfFailedIterations += 1;
                Console.WriteLine("Total Steps Taken: " + this._regularMoves);
            }

            return new int[] { _totalNumberOfStepsSucceeds,
_numberOfSuccessfulIterations, _totalNumberOfStepsFails,
                _numberOfFailedIterations };
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace NQueenusingHillClimbing
{

    public sealed class RandomRestart : NqueenBoard
    {
        private int _totalNumberOfStepsSucceeds = 0;
        private int _numberOfSuccessfulIterations = 0;
        private Boolean _boardChanged = true;
        private int _consecutiveSidewaysMoves = 0;
```

Programming Project 2
Solving N-queens problem using hill-climbing and its variants

Team Member: Shubhangi Alande (801104235)

```
        private readonly int LIMIT_CONSECUTIVE_SIDEWAYS_MOVES = 100;

        public RandomRestart(int numOfQueens) : base(numOfQueens)
        {

        }

        /// <summary>
        /// restarting the game board (by calling resetboard()) any time the board is
'stuck' at h > 0
        /// </summary>
        /// <returns>success/ failure analysis</returns>
        public int[] RunAlgorithmWithoutSideways()
        {
            this.StartGameBoard();
            Boolean boardChanged = true;
            int currentStateHeuristic = this.CalculateHeuristic(this._gameBoard);
            List<int[,]> possibleMoves = new List<int[,]>();
            this.ResetRegularMoves();

            while (this.CalculateHeuristic(this._gameBoard) != 0 && boardChanged == true)
            {
                int columnNumber = 0;
                int[,] possibleMove = new int[this.GetBoardSize(),this.GetBoardSize()];
                for (int i = 0; i < this.GetBoardSize(); i++)
                {
                    for (int j = 0; j < this.GetBoardSize(); j++)
                    {
                        possibleMove[i,j] = this._gameBoard[i,j];
                    }
                }

                currentStateHeuristic = this.CalculateHeuristic(this._gameBoard);
                this._boardChanged = false;

                while (columnNumber < this.GetBoardSize())
                {
                    int currentColumnQueenPosition = -1;

                    for (int i = 0; i < this.GetBoardSize(); i++)
                    {
                        if (possibleMove[i,columnNumber] == this._QUEEN)
                            currentColumnQueenPosition = i;
                        possibleMove[i,columnNumber] = this._NOT_QUEEN;
                    }

                    for (int i = 0; i < this.GetBoardSize(); i++)
                    {
                        possibleMove[i,columnNumber] = this._QUEEN;

                        int[,] newMove = new
int[this.GetBoardSize(),this.GetBoardSize()];
                        for (int k = 0; k < this.GetBoardSize(); k++)
                        {
                            for (int j = 0; j < this.GetBoardSize(); j++)
                            {
                                newMove[k,j] = possibleMove[k,j];
                            }
                        }
                        if (this.CalculateHeuristic(this._gameBoard) >
this.CalculateHeuristic(newMove))
                            possibleMoves.Add(newMove);
                        possibleMove[i,columnNumber] = this._NOT_QUEEN;
                    }

                    possibleMove[currentColumnQueenPosition,columnNumber] = this._QUEEN;

                    columnNumber += 1;
                }

                Random rand = new Random();

                if (possibleMoves.Count != 0)
                {
                    int pick = rand.Next(possibleMoves.Count);

                    int minHeuristic = currentStateHeuristic;

                    if (minHeuristic > this.CalculateHeuristic(possibleMoves[pick]))
                    {
                        minHeuristic = this.CalculateHeuristic(possibleMoves[pick]);
                        this._gameBoard = this.CopyBoard(possibleMoves[pick]);
                        this._boardChanged = true;

                        this._regularMoves += 1;
                        Console.WriteLine("\nStep No." + _regularMoves);
                        this.PrintGameBoard();
                        possibleMoves.Clear();
                    }
                }
                else
                {
                    this.ResetBoard();
                    this.SetGameBoard();
                    this._boardChanged = true;
                    possibleMoves.Clear();
                    Console.WriteLine("Random Restart!");
                    this.StartGameBoard();
                }
            }

            if (this.CalculateHeuristic(this._gameBoard) == 0)
            {
                Console.WriteLine("\nSuccess!");
                this._totalNumberOfStepsSucceeds += this._regularMoves;
                this._numberOfSuccessfulIterations += 1;
                Console.WriteLine("Total Steps Taken: " + this._regularMoves);
            }

            return new int[] { _totalNumberOfStepsSucceeds, 
_numberOfSuccessfulIterations, this.GetResets() };
        }

        /// <summary>
        /// restarting the game board (by calling resetboard()) any time the board is 
'stuck' at h > 0 with side way moves allowed
        /// </summary>
        /// <returns>success/ failure analysis</returns>
        public int[] RunAlgorithmWithSideways()
        {
            this.StartGameBoard();
            Boolean boardChanged = true;
            int currentStateHeuristic = this.CalculateHeuristic(this._gameBoard);
            List<int[,]> possibleMoves = new List<int[,]>();
            this.ResetRegularMoves();

            while (this.CalculateHeuristic(this._gameBoard) != 0 && boardChanged == true)
            {
                int columnNumber = 0;
                int[,] possibleMove = new int[this.GetBoardSize(),this.GetBoardSize()];
                for (int i = 0; i < this.GetBoardSize(); i++)
                {
                    for (int j = 0; j < this.GetBoardSize(); j++)
                    {
                        possibleMove[i,j] = this._gameBoard[i,j];
                    }
                }

                currentStateHeuristic = this.CalculateHeuristic(this._gameBoard);
                this._boardChanged = false;

                while (columnNumber < this.GetBoardSize())
                {
                    int currentColumnQueenPosition = -1;

                    for (int i = 0; i < this.GetBoardSize(); i++)
                    {
                        if (possibleMove[i,columnNumber] == this._QUEEN)
                            currentColumnQueenPosition = i;
                        possibleMove[i,columnNumber] = this._NOT_QUEEN;
                    }

                    for (int i = 0; i < this.GetBoardSize(); i++)
                    {
                        possibleMove[i,columnNumber] = this._QUEEN;

                        int[,] newMove = new 
int[this.GetBoardSize(),this.GetBoardSize()];
                        for (int k = 0; k < this.GetBoardSize(); k++)
                        {
                            for (int j = 0; j < this.GetBoardSize(); j++)
                            {
                                newMove[k,j] = possibleMove[k,j];
                            }
                        }
                        if (this.CalculateHeuristic(this._gameBoard) >= 
this.CalculateHeuristic(newMove) && this.BoardsAreEqual(this._gameBoard, newMove) == 
false)
                            possibleMoves.Add(newMove);
                        possibleMove[i,columnNumber] = this._NOT_QUEEN;
                    }

                    possibleMove[currentColumnQueenPosition,columnNumber] = this._QUEEN;

                    columnNumber += 1;
                }

                Random rand = new Random();
                int minHeuristic = currentStateHeuristic;

                if (possibleMoves.Count != 0)
                {
                    int pick = rand.Next(possibleMoves.Count);

                    if (minHeuristic > this.CalculateHeuristic(possibleMoves[pick]))
                    {
                        minHeuristic = this.CalculateHeuristic(possibleMoves[pick]);
                        this._gameBoard = this.CopyBoard(possibleMoves[pick]);
                        this._boardChanged = true;
                        this._consecutiveSidewaysMoves = 0;
                        this._regularMoves += 1;
                        Console.WriteLine("\nStep No." + _regularMoves);
                        this.PrintGameBoard();
                        possibleMoves.Clear();
                    }
                    else if (minHeuristic == this.CalculateHeuristic(possibleMoves[pick]) 
&&
                          this._consecutiveSidewaysMoves < 
this.LIMIT_CONSECUTIVE_SIDEWAYS_MOVES)
                    {
                        minHeuristic = this.CalculateHeuristic(possibleMoves[pick]);
                        this._gameBoard = this.CopyBoard(possibleMoves[pick]);
                        this._boardChanged = true;
                        this._consecutiveSidewaysMoves += 1;
                        this._regularMoves += 1;
                        Console.WriteLine("\nStep No." + _regularMoves);
                        this.PrintGameBoard();
                        Console.WriteLine("Sideways Move!");
                        possibleMoves.Clear();
                    }
                    else
                    {
                        this.ResetBoard();
                        this.SetGameBoard();
                        this._boardChanged = true;
                        possibleMoves.Clear();
                        Console.WriteLine("Random Restart!");
                        this.StartGameBoard();
                    }
                }
                else
                {
                    this.ResetBoard();
                    this.SetGameBoard();
                    this._boardChanged = true;
                    possibleMoves.Clear();
                    Console.WriteLine("Random Restart!");
                    this.StartGameBoard();
                }
            }

            if (this.CalculateHeuristic(this._gameBoard) == 0)
            {
                Console.WriteLine("\nSuccess!");
                this._totalNumberOfStepsSucceeds += this._regularMoves;
                this._numberOfSuccessfulIterations += 1;
                Console.WriteLine("Total Steps Taken: " + this._regularMoves);
            }

            return new int[] { _totalNumberOfStepsSucceeds,
_numberOfSuccessfulIterations, this.GetResets() };
        }
    }
}


using System;

namespace NQueenusingHillClimbing
{
    public abstract class NqueenBoard
    {
        Random rand = new Random();
        private int[,] _gameBoardOne;
        private int[,] _gameBoardTwo;
        private int[,] _gameBoardThree;
        public int[,] _gameBoard;
        public int _numOfResets = -1;
        public int _numberOfQueens;
        public int _regularMoves = 0;
        public int _sidewaysMoves = 0;
        public readonly int _QUEEN = 1;
        public readonly int _NOT_QUEEN = 0;

        public NqueenBoard(int numOfQueens)
        {
            this._numberOfQueens = numOfQueens;
            this._gameBoardOne = new int[numOfQueens,numOfQueens];
            this._gameBoardTwo = new int[numOfQueens,numOfQueens];
            this._gameBoardThree = new int[numOfQueens,numOfQueens];
            this._gameBoard = new int[numOfQueens,numOfQueens];
            this.ResetBoard();
            this.SetGameBoard();

        }

        public int GetBoardSize()
        {
            return this._numberOfQueens;
        }

        public int[,] GetGameBoardOne()
        {
            return this._gameBoardOne;
        }

        public int[,] GetGameBoardTwo()
        {
            return this._gameBoardTwo;
        }

        public int[,] GetGameBoardThree()
        {
            return this._gameBoardThree;
        }

        public void SetGameBoard()
        {
            int n = rand.Next(2);
            if (n == 0)
                this._gameBoard = this.CopyBoard(_gameBoardOne);
            else if (n == 1)
                this._gameBoard = this.CopyBoard(_gameBoardTwo);
            else
                this._gameBoard = this.CopyBoard(_gameBoardThree);
        }

        public int[,] GetGameBoard()
        {
            return this._gameBoard;
        }

        /// <summary>
        /// calculates the heuristic of the board sent as  parameter. that is no of pairs
of queens attacking each other.
        /// </summary>
        /// <param name="tempBoard"></param>
        /// <returns>integer heuristic value</returns>
```

Programming Project 2
Solving N-queens problem using hill-climbing and its variants

Team Member: Shubhangi Alande (801104235)

```
public int CalculateHeuristic(int[,] tempBoard)
{
    int[,] board = tempBoard;
    int heuristic = 0;


    for (int i = 0; i < this._numberOfQueens; i++)
    {
        int noQueens = 0;
        for (int j = 0; j < this._numberOfQueens; j++)
        {
            if (board[i, j] == _QUEEN)
                noQueens += 1;
        }

        if (noQueens > 1)
        {
            for (int queens = noQueens; queens > 1; --queens)
            {
                heuristic += queens - 1;
            }
        }
    }


    for (int i = 0; i < this._numberOfQueens; i++)
    {
        int noQueens = 0;
        for (int j = 0; j < this._numberOfQueens; j++)
        {
            if (board[j,i] == _QUEEN)
                noQueens += 1;
        }

        if (noQueens > 1)
        {
            for (int queens = noQueens; queens > 1; --queens)
            {
                heuristic += queens - 1;
            }
        }
    }


    int numOfQueens = 0;
    int iteration = 0;
    while (iteration < (this._numberOfQueens - 1))
    {
        numOfQueens = 0;
        int xValue = 0;
        int yValue = iteration;

        while (xValue <= iteration)
        {
            if (board[xValue,yValue] == _QUEEN)
                numOfQueens += 1;

            yValue -= 1;
            xValue += 1;
        }

        if (numOfQueens > 1)
        {
            for (int queens = numOfQueens; queens > 1; --queens)
            {
                heuristic += queens - 1;
            }
        }

        iteration += 1;
    }


    numOfQueens = 0;
    int diagonal = this._numberOfQueens - 1;
    for (int i = 0; i < this._numberOfQueens; i++)
    {
        if (board[i,diagonal] == _QUEEN)
            numOfQueens += 1;
        diagonal -= 1;
    }

    if (numOfQueens > 1)
    {
        for (int queens = numOfQueens; queens > 1; --queens)
        {
            heuristic += queens - 1;
        }
    }


    numOfQueens = 0;
    iteration = 1;
    while (iteration < (this._numberOfQueens - 1))
    {
        numOfQueens = 0;
        int xValue = iteration;
        int yValue = this._numberOfQueens - 1;

        while (xValue < this._numberOfQueens)
        {
            if (board[xValue,yValue] == _QUEEN)
                numOfQueens += 1;

            yValue -= 1;
            xValue += 1;
        }
        if (numOfQueens > 1)
        {
            for (int queens = numOfQueens; queens > 1; --queens)
            {
                heuristic += queens - 1;
            }
        }

        iteration += 1;
    }


    numOfQueens = 0;
    iteration = this._numberOfQueens - 2;
    while (iteration > 0)
    {
        numOfQueens = 0;
        int xValue = 0;
        int yValue = iteration;

        while (yValue < this._numberOfQueens)
        {
            if (board[xValue,yValue] == _QUEEN)
                numOfQueens += 1;

            yValue += 1;
            xValue += 1;
        }

        if (numOfQueens > 1)
        {
            for (int queens = numOfQueens; queens > 1; --queens)
            {
                heuristic += queens - 1;
            }
        }

        iteration -= 1;
    }


    numOfQueens = 0;
    diagonal = 0;
    for (int i = 0; i < this._numberOfQueens; i++)
    {
        if (board[i,diagonal] == _QUEEN)
            numOfQueens += 1;
        diagonal += 1;
    }

    if (numOfQueens > 1)
    {
        for (int queens = numOfQueens; queens > 1; --queens)
        {
            heuristic += queens - 1;
        }
    }


    numOfQueens = 0;
    iteration = 1;
    while (iteration < this._numberOfQueens)
    {
        numOfQueens = 0;
        int xValue = iteration;
        int yValue = 0;

        while (xValue < this._numberOfQueens)
        {
            if (board[xValue,yValue] == _QUEEN)
                numOfQueens += 1;

            yValue += 1;
            xValue += 1;
        }
        if (numOfQueens > 1)
        {
            for (int queens = numOfQueens; queens > 1; --queens)
            {
                heuristic += queens - 1;
            }
        }

        iteration += 1;
    }

    return heuristic;
}

/// <summary>
/// creates random game boards
/// </summary>
public void ResetBoard()
{
    int numOfQueens = this._numberOfQueens;

    for (int i = 0; i < numOfQueens; i++)
    {
        // RANDOM QUEEN GENERATION
        int queenPositionY = rand.Next(numOfQueens - 1);

        for (int j = 0; j < numOfQueens; j++)
        {
            if (j == queenPositionY)
                this._gameBoardOne[j,i] = _QUEEN;
            else
                this._gameBoardOne[j,i] = _NOT_QUEEN;
        }
    }


    for (int i = 0; i < numOfQueens; i++)
    {
        // RANDOM QUEEN GENERATION
        int queenPositionY = rand.Next(numOfQueens - 1);

        for (int j = 0; j < numOfQueens; j++)
        {
            if (j == queenPositionY)
                this._gameBoardTwo[j,i] = _QUEEN;
            else
                this._gameBoardTwo[j,i] = _NOT_QUEEN;
        }
    }


    for (int i = 0; i < numOfQueens; i++)
    {
        // RANDOM QUEEN GENERATION
        int queenPositionY = rand.Next(numOfQueens - 1);

        for (int j = 0; j < numOfQueens; j++)
        {
            if (j == queenPositionY)
                this._gameBoardThree[j,i] = _QUEEN;
            else
                this._gameBoardThree[j,i] = _NOT_QUEEN;
        }
    }

    this._numOfResets += 1;
}

public int GetResets()
{
    return this._numOfResets;
}

public void ResetRegularMoves()
{
    this._regularMoves = 0;
}

public void ResetSidewaysMoves()
{

    this._sidewaysMoves = 0;
}
```

Programming Project 2
Solving N-queens problem using hill-climbing and its variants

Team Member: Shubhangi Alande (801104235)

```
        public void StartGameBoard()
        {
            Console.WriteLine("\n\n-----------------------------------");
            Console.WriteLine("Starting Board");
            Console.WriteLine("Current State (Heuristic): " +
this.CalculateHeuristic(this._gameBoard));

            for (int x = 0; x < this._gameBoard.GetLength(0); x++)
            {
                for (int y = 0; y < this._gameBoard.GetLength(1); y++)
                {
                    Console.Write(this._gameBoard[x,y] + " ");
                }
                Console.WriteLine();
            }
        }

        public void PrintGameBoard()
        {
            Console.WriteLine("Current State (Heuristic): " +
this.CalculateHeuristic(this._gameBoard));

            for (int x = 0; x < this._gameBoard.GetLength(0); x++)
            {
                for (int y = 0; y < this._gameBoard.GetLength(1); y++)
                {
                    Console.Write(this._gameBoard[x,y] + " ");
                }
                Console.WriteLine();
            }
        }

        public int[,] CopyBoard(int[,] oldBoard)
        {
            int[,] newBoard = new int[this.GetBoardSize(),this.GetBoardSize()];

            for (int i = 0; i < oldBoard.GetLength(0); i++)
            {
                for (int j = 0; j < oldBoard.GetLength(1); j++)
                {
                    newBoard[i,j] = oldBoard[i,j];
                }
            }
            return newBoard;
        }

        public Boolean BoardsAreEqual(int[,] boardOne, int[,] boardTwo)
        {
            Boolean areEqual = true;

            for (int i = 0; i < this.GetBoardSize(); i++)
            {
                for (int j = 0; j < this.GetBoardSize(); j++)
                {
                    if (boardOne[i,j] != boardTwo[i,j])
                        areEqual = false;
                }
            }
            return areEqual;
        }
    }
}
```