Unique Queens Solution

Out of the 92 solutions to the 8-queens problem, only 12 are essentially different, in the sense that none of these 12 solutions are symmetries of each other. There are 8 symmetries of a square, 4 rotations (clockwise: 0°, 90°, 180°, 270°) and 4 reflections (horizontal, vertical, up-diagonal, down-diagonal). These 8 symmetries can be generated by 'rotate' (90° rotation-clockwise) and 'mirror' (horizontal reflection).
Let R be 'rotate' and M be 'mirror'

On a NxN board, [top-left square is (1,1)]

	Symmetry	Generated by
Rotations:		
	90°	R
	180°	R^2 (= R∘R)
	270°	R^3
	0°	R^4

	R(i,j)	=	(j, N+1-i) -- 90° clockwise
e.g.	R(6, 2)	=	(2, 3)
	R^2(i,j)	=	R(j, N+1-1)
		=	(N+1-i, N+1-j)
e.g.	R^2(6, 2)	=	(3, 7)
	R^3(i,j)	=	R(N+1-i, N+1-j)
		=	(N+1-j, i)
e.g.	R^3(6, 2)	=	(7, 6)

Reflections:

Horizontal	**M**
Down_Diag	$R \circ M$
Vertical	$R^2 \circ M$
Up-Diag	$R^3 \circ M$

	$M(i,j)$	=	$(N+1-i, j)$
e.g.	$M(6, 2)$	=	$(3, 2)$ -- **Horizontal**
	$R \circ M(i,j)$	=	$R\,(N+1-i, j)$
		=	$(j, N+1-(N+1-i))$
		=	$(j, i))$
e.g.	$R \circ M(6, 2)$	=	$(2, 6)$ -- **Down_Diag**
	$R^2 \circ M(i,j)$	=	$R^2\,(N+1-i, j)$
		=	$(N+1-(N+1-i), N+1-j)$
		=	$(i, N+1-j)$
e.g.	$R^2 \circ M(6, 2)$	=	$(6, 7)$ -- **Vertical**
	$R^3 \circ M(i,j)$	=	$R^3\,(N+1-i, j)$
		=	$(N+1-j, N+1-i)$
e.g.	$R^3 \circ M(6, 2)$	=	$(7, 3)$ -- **Up_Diag**

Representation on Arrays

A solution to the N-Queens problem is represented by an array where

$q@i = j$ **iff a queen is at** (i,j) **on the board**

iff " $(i, q@i)$ "

A symmetry, a rotation, say, maps

$$(i,j) \quad \rightarrow \quad (j, N+1-i)$$

In array form

$$(i,q@i) \quad \rightarrow \quad (q@i, N+1-i)$$

```
rotate (a: ARRAY[INTEGER]):ARRAY[INTEGER] is
        local
                i, n, it: INTEGER
        do
                !! Result.make (1, a.count);
                from
                        i := 1;
                        n := a.count
                until
                        i > n
                loop
                        it := a.item (i);
                        Result.put (n + 1 - i, it);
                        i := i + 1
                end
        end ; -- rotate

mirror (a: ARRAY[INTEGER]):ARRAY[INTEGER] is
                -- mirrors board (reverses array)
        local
                i, n, it: INTEGER
        do
                !! Result.make (1, a.count);
                from
                        i := 1;
                        n := a.count
                until
                        i > n
                loop
                        it := a.item (i);
                        Result.put (it, n + 1 - i);
                        i := i + 1
                end
        end ; -- mirror
```

Consider the solution:

	1	2	3	4	5	6	7	8
1	Q							
2					Q			
3								Q
4						Q		
5			Q					
6							Q	
7		Q						
8				Q				

In array form this is

1 5 8 6 3 7 2 4

Rotating board, clockwise 90°, we get

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1								**Q**
2		**Q**						
3				**Q**				
4	**Q**							
5							**Q**	
6					**Q**			
7			**Q**					
8						**Q**		

In array form this is,

8 2 4 1 7 5 3 6

Rotating again, R^2 (180°), we get,

5 7 2 6 3 1 4 8

Rotating once more, R^3 (270°), we get

3 6 4 2 8 5 7 1

Reflecting this solution Horizontally, we get,

1 7 5 8 2 4 6 3

and succesive rotations give us

8 4 1 3 6 2 7 5

6 3 5 7 1 4 2 8

and 4 2 7 3 6 8 5 1

This gives us 8 solutions in all

1 5 8 6 3 7 2 4 -- initial solution.

8 2 4 1 7 5 3 6

5 7 2 6 3 1 4 8

3 6 4 2 8 5 7 1

1 7 5 8 2 4 6 3

8 4 1 3 6 2 7 5

6 3 5 7 1 4 2 8

and 4 2 7 3 6 8 5 1

We see that each of the 7 solutions generated from the initial one are 'alphabetically / numerically' greater than the initial one.

Eiffel Routines for Unique Solution

To output just the unique solutions we determine at output if there are any symmetric solutions of the current solutions that are 'less' that the current one; if not then we output this solution but if there is a symmetric solution 'less' than the current one then this current solution is skipped. We can express this using a boolean function, is_unique**.**

```
is_unique (a: ARRAY [INTEGER]): BOOLEAN is
    local
        i, j: INTEGER;
        b: ARRAY [INTEGER]
    do
        from
            b := clone (a);
            i := 1;
            j := 8
        until
            i = j
        loop
            if  i = 4 then
                b := mirror (b)
            else
                b := rotate (b)
            end ;
            if  lt (b, a) then
                j := i
            else
                i := i + 1
            end
        end ;
        Result := i = 8
    end ; --is_unique
```

This function considers all symmetries, b, of the initial solution, a. If a symmetric soluton, b, is found such that lt(b,a) then the solution is not unique and it or its base symmetric version was output before. If no such, b, is found then the current solution,a, is a base symmetric form and so can be output.

The routine that generates just the unique solutions is given as:

```
unique_queens (i: INTEGER) is
        local
                j: INTEGER
        do
                if  i > q.count then
                        if  is_unique (q) then
                                clear_matrix;
                                queens2matrix;
                                print_matrix
                        end
                else
                        from
                                j := 1
                        until
                                j > q.count
                        loop
                                if  safe (i, j) then
                                        set_queen (i, j);
                                        unique_queens (i + 1);
                                        reset_queen (i, j)
                                end ;
                                j := j + 1
                        end
                end
        end ; -- unique_queens
```

The 12 unique solutions are:

Soln #1

1 5 8 6 3 7 2 4

```
  1 2 3 4 5 6 7 8
1 x
2         x
3               x
4           x
5     x
6             x
7   x
8       x
```

Soln #2

1 6 8 3 7 4 2 5

```
  1 2 3 4 5 6 7 8
1 x
2           x
3               x
4     x
5             x
6       x
7   x
8         x
```

Soln #3

2 4 6 8 3 1 7 5

```
   1 2 3 4 5 6 7 8
1    x
2        x
3            x
4                x
5      x
6  x
7              x
8          x
```

Soln #4

2 5 7 1 3 8 6 4

```
   1 2 3 4 5 6 7 8
1    x
2          x
3              x
4  x
5      x
6                x
7            x
8        x
```

Soln #5

2 5 7 4 1 8 6 3

```
  1 2 3 4 5 6 7 8
1   x
2         x
3             x
4       x
5 x
6               x
7           x
8     x
```

Soln #6

2 6 1 7 4 8 3 5

```
  1 2 3 4 5 6 7 8
1   x
2           x
3 x
4             x
5       x
6               x
7     x
8         x
```

Soln #7

2 6 8 3 1 4 7 5

```
  1 2 3 4 5 6 7 8
1   x
2           x
3               x
4     x
5 x
6       x
7             x
8         x
```

Soln #8

2 7 3 6 8 5 1 4

```
  1 2 3 4 5 6 7 8
1   x
2             x
3     x
4           x
5               x
6         x
7 x
8       x
```

Soln #9

2 7 5 8 1 4 6 3

```
  1 2 3 4 5 6 7 8
1   x
2             x
3         x
4               x
5 x
6       x
7           x
8     x
```

Soln #10

3 5 2 8 1 7 4 6

```
  1 2 3 4 5 6 7 8
1     x
2         x
3   x
4               x
5 x
6             x
7       x
8           x
```

Soln #11

3 5 8 4 1 7 2 6

```
  1 2 3 4 5 6 7 8
1     x
2         x
3             x
4       x
5 x
6           x
7   x
8         x
```

Soln #12

3 6 2 5 8 1 7 4

```
  1 2 3 4 5 6 7 8
1     x
2           x
3   x
4         x
5             x
6 x
7           x
8       x
```