2. M-Queens

The famous N-queens problem is actually just a crude approximation of the less famous M-queens problem, which consists in finding all *maximally* safe configurations of queens on an $M \times M$ chess board. A configuration is maximally safe if no queen can be added without the configuration becoming unsafe. The usual definition of a safe configuration applies here too: no queen can beat any other queen. So, write a predicate `mqueens/2`, which, given as input a number, unifies the second argument with all maximally safe configurations through backtracking. The answers for input 1, 2 and 3 are given below:

```
?- mqueens(1,L).
L = [1]

?- mqueens(2,L).
L = [1,none]
L = [2,none]
L = [none,1]
L = [none,2]
```

```
?- mqueens(3,L).
L = [1,3,none]
L = [1,none,2]
L = [2,none,1]
L = [2,none,3]
L = [3,1,none]
L = [3,none,2]
L = [none,1,3]
L = [none,2,none]
L = [none,3,1]
```

The order in which solutions are delivered is not important. Each solution should be given only once. The convention is clear, but just to make sure: a solution `[none,3,none,1]` denotes a configuration in which there is no queen in the first column, one queen in column 2 in row 3, no queen in column 3 and one queen in row 1 of column 4.

Hints *We can adapt an ordinary N-queens program to generate (potential) solutions with empty columns as well. However, we need something to check whether or not we can put an extra queen in any of the empty columns. One way to do this is to check whether all squares on the board are attacked by the queens in the solution. For instance, the generation phase might generate* `[3,none,none]`*, but the square (3,2) is not attacked, so we can still place a queen in column three, and so* `[3,none,none]` *is rejected.*

Solution

```
:- use_module(library(lists),[member/2]).
:- use_module(contestlib,[numlist/3]).

mqueens(M,Solution) :-
        findall(sq(A,B),(numlist(1,M,Rows),
                         member(A,Rows),member(B,Rows)),Squares),
        mqueens(M,Squares,[],Solution).

mqueens(M,NotAttacked,PartialSolution,Solution) :-
        ( M == 0 ->
            NotAttacked = [],
            Solution = PartialSolution
        ;
            M1 is M - 1,
            (
              member(sq(M,X),NotAttacked),
              safe(PartialSolution,1,X),
              delete_attacked(NotAttacked,M,X,NotAttacked1)
            ;
              X = none,
              NotAttacked1 = NotAttacked
            ),
            mqueens(M1,NotAttacked1,[X|PartialSolution],Solution)
        ).

safe([],_,_).
safe([X|R],Dist,Y) :-
        ( X == none ->
            true
        ;
            X =\= Y,
            abs(X - Y) =\= Dist
        ),
        Dist1 is Dist + 1,
        safe(R,Dist1,Y).

delete_attacked([],_,_,[]).
delete_attacked([sq(A,B)|R],I,J,Squares) :-
        ( (A =:= I ; B =:= J ; A-B =:= I-J ; A+B =:= I+J) ->
            delete_attacked(R,I,J,Squares)
        ;
            Squares = [sq(A,B)|S],
            delete_attacked(R,I,J,S)
        ).
```