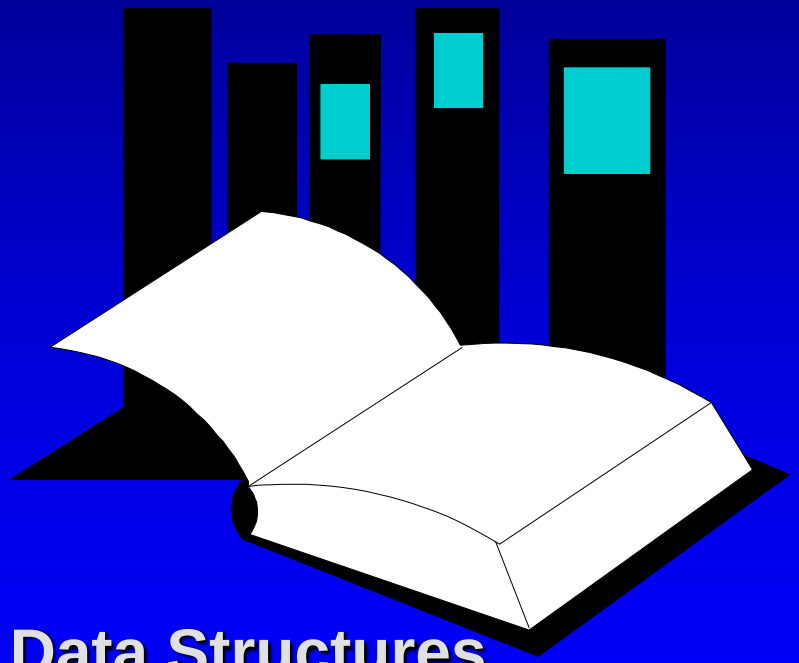
Using a Stack

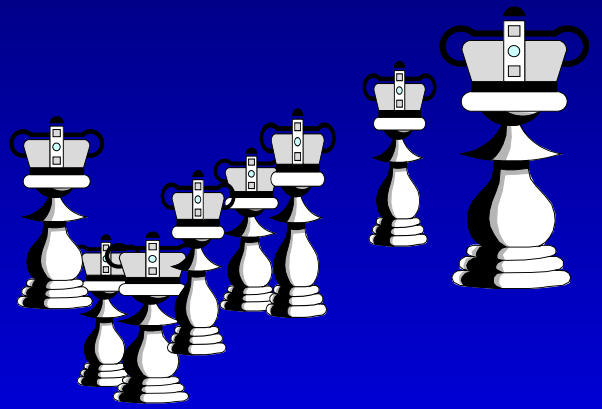


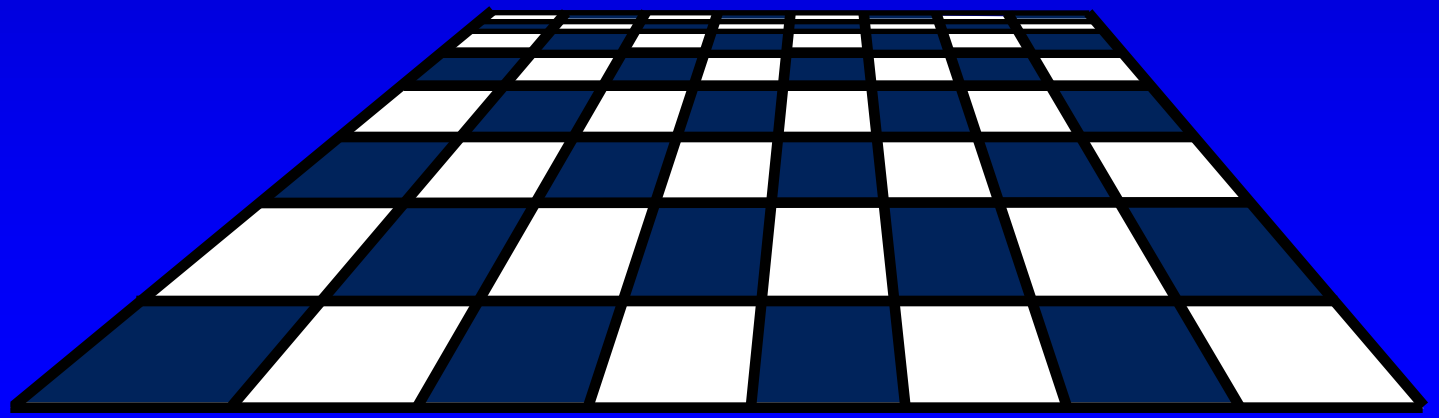
Data Structures and Other Objects Using Java

- Chapter 6 introduces the **stack** data type.
- Several example applications of stacks are given in that chapter.
- This presentation shows another use called **backtracking to solve the N-Queens problem**.

The N-Queens Problem

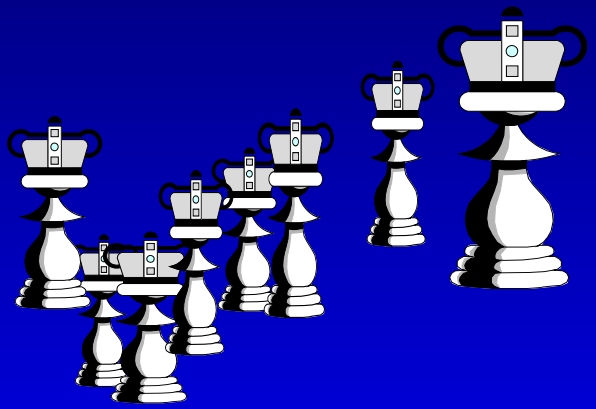
- Suppose you have 8 chess queens...
- ...and a chess board

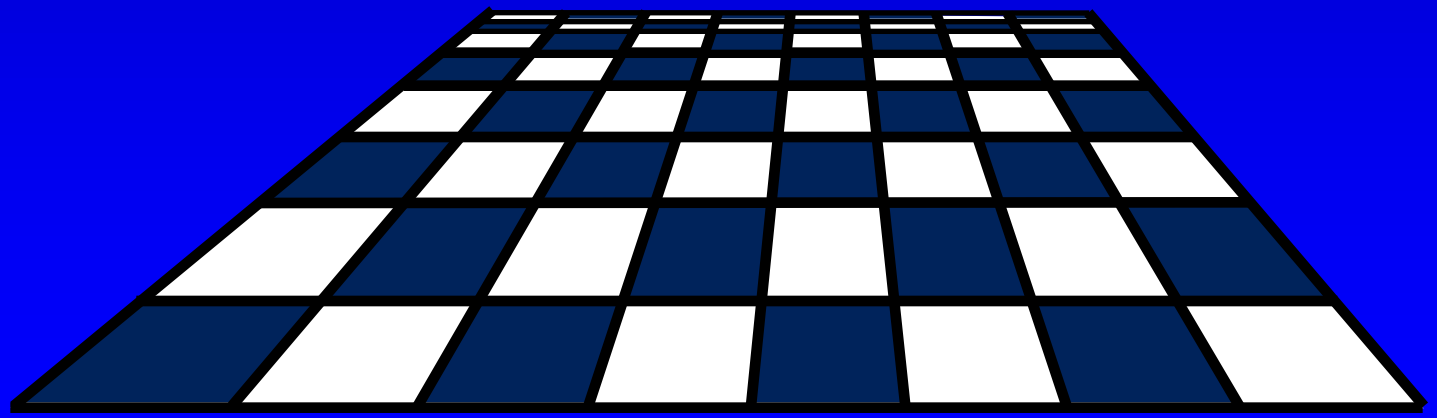




The N-Queens Problem

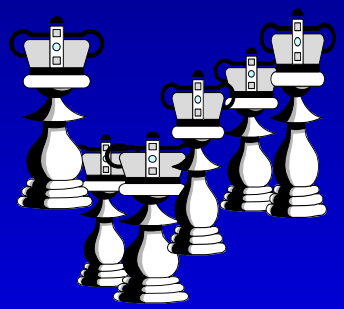
Can the queens be placed on the board so that no two queens are attacking each other ?

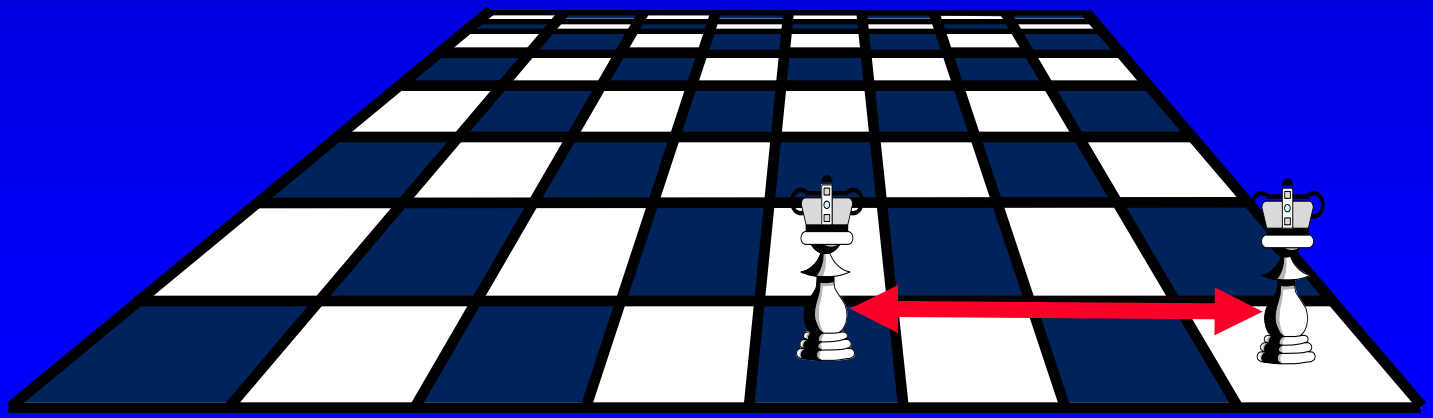




The N-Queens Problem

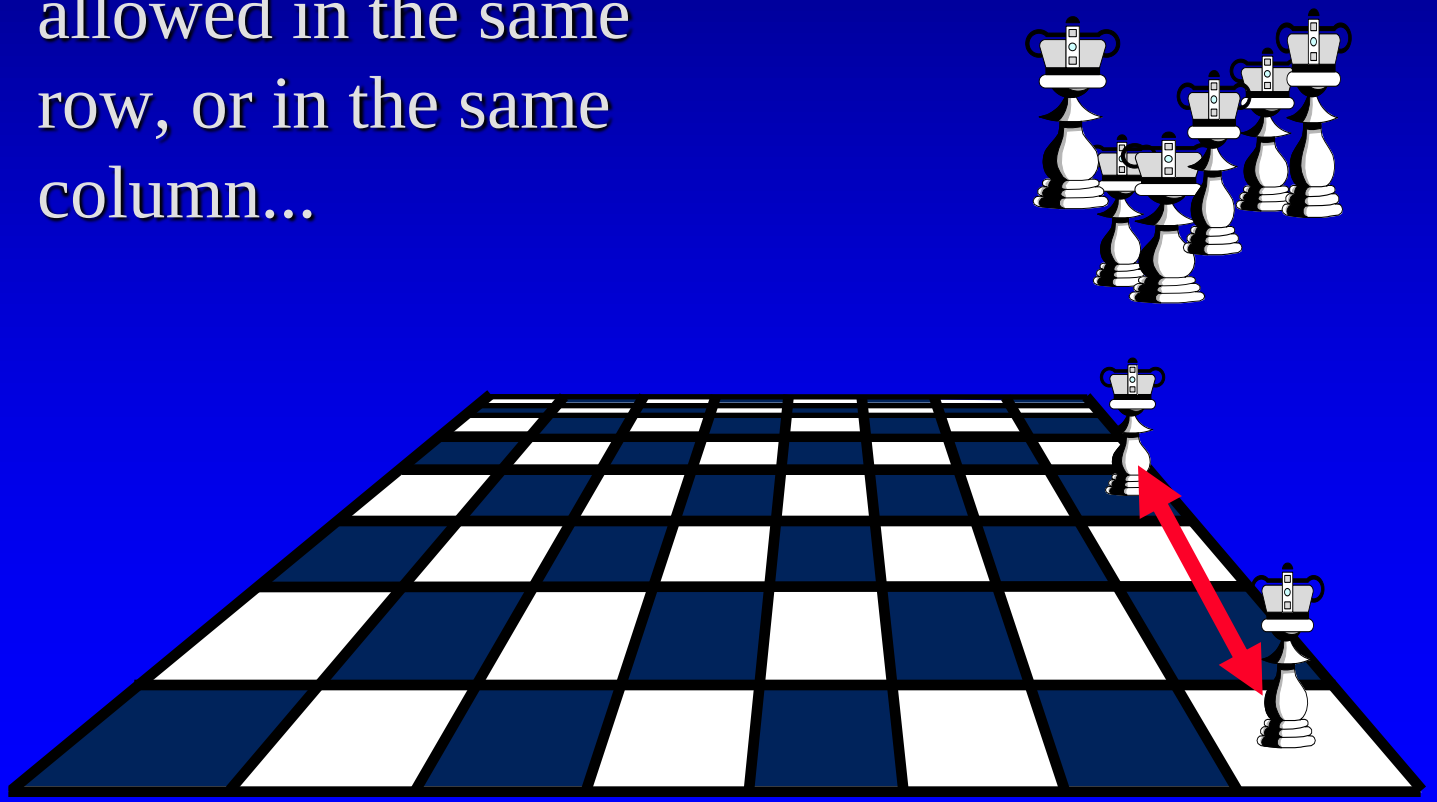
Two queens are not allowed in the same row...





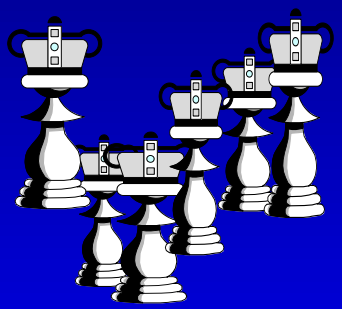
The N-Queens Problem

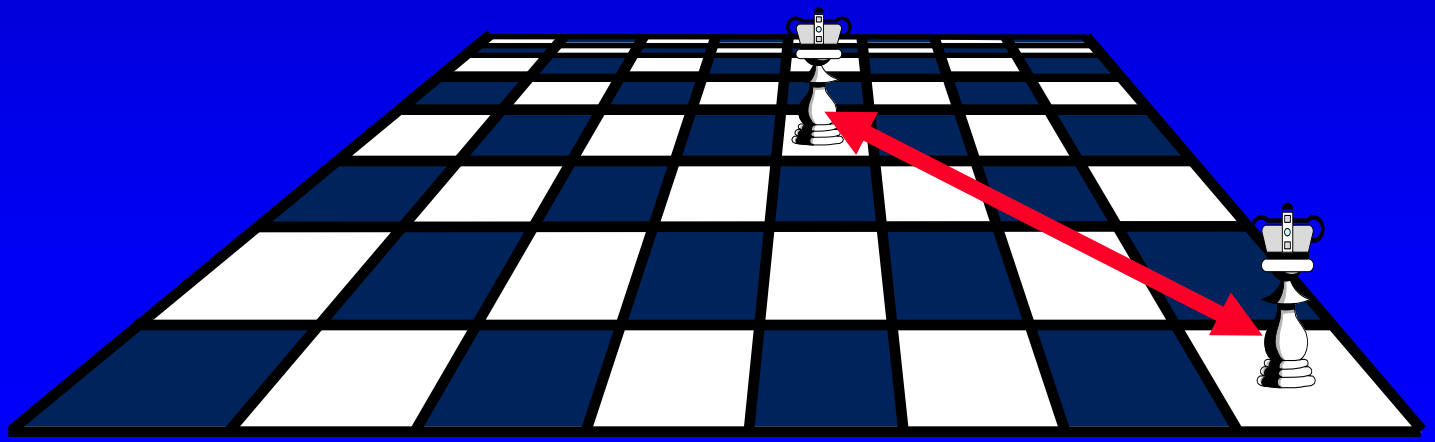
Two queens are not allowed in the same row, or in the same column...



The N-Queens Problem

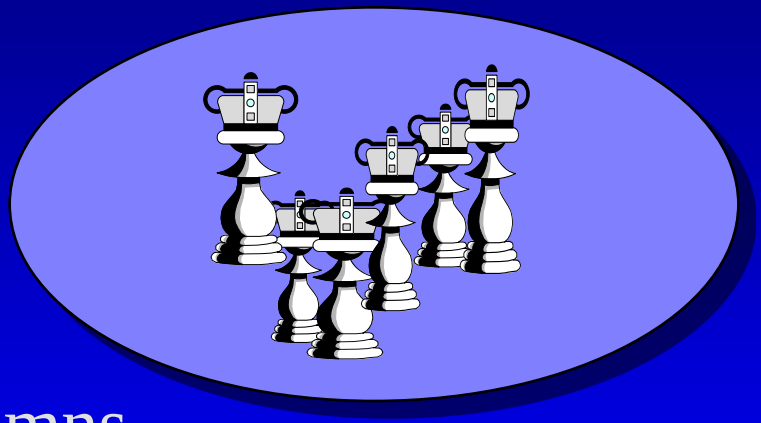
Two queens are not allowed in the same row, or in the same column, or along the same diagonal.

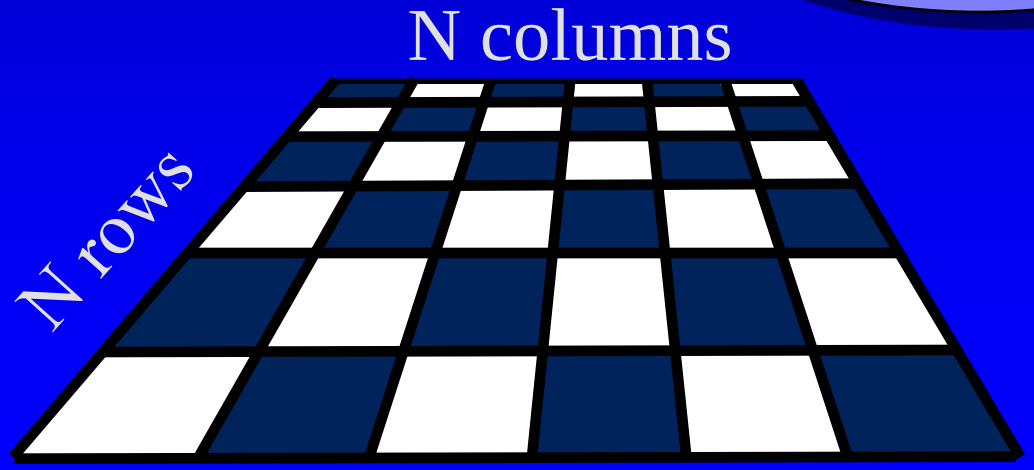




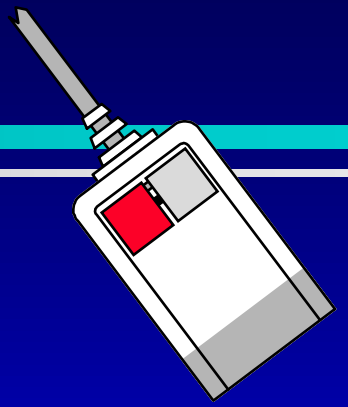
The N-Queens Problem

The number of queens, and the size of the board can vary.

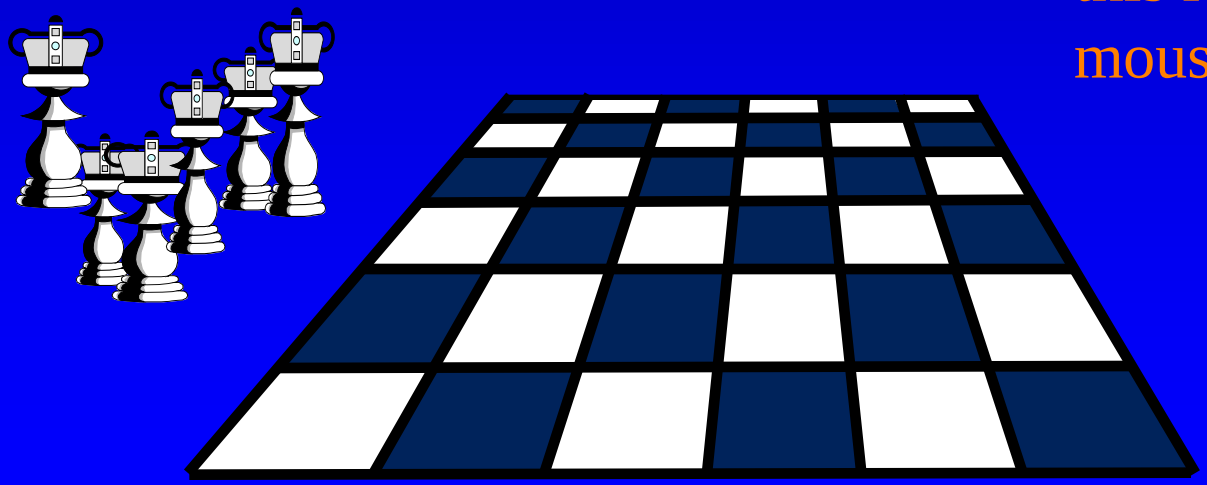




The N-Queens Problem



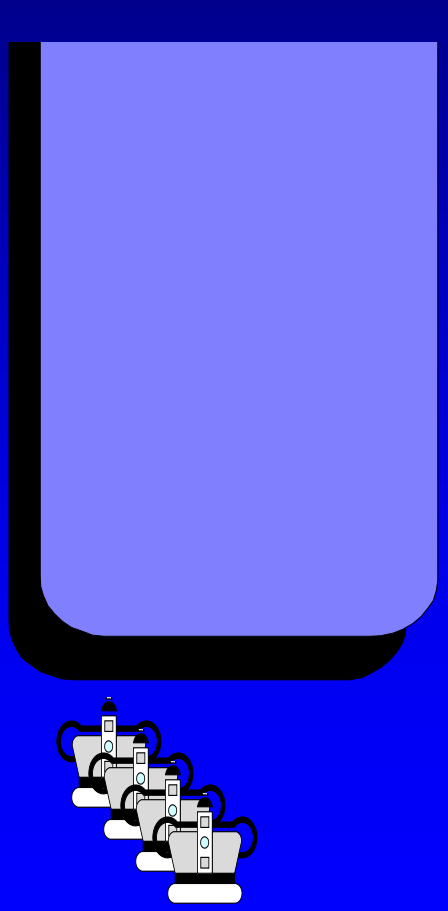
We will write a program which tries to find a way to place N queens on an N x N chess board.

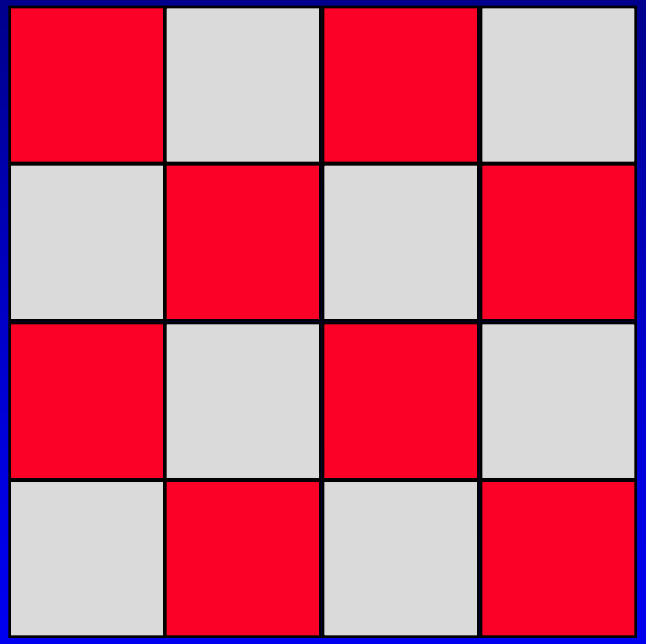


If you can run ega or vga graphics,
you can double click on this icon with the left mouse button:

How the program works

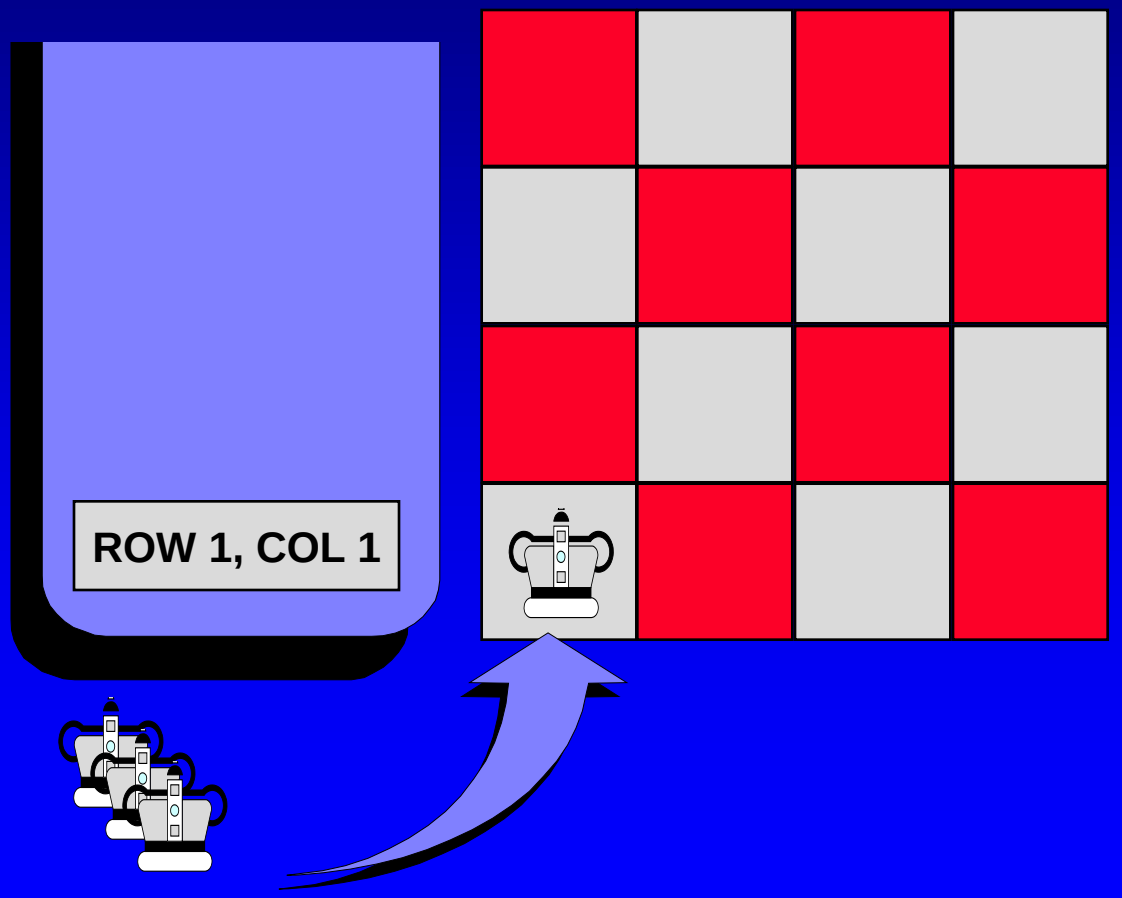
The program uses a stack to keep track of where each queen is placed.





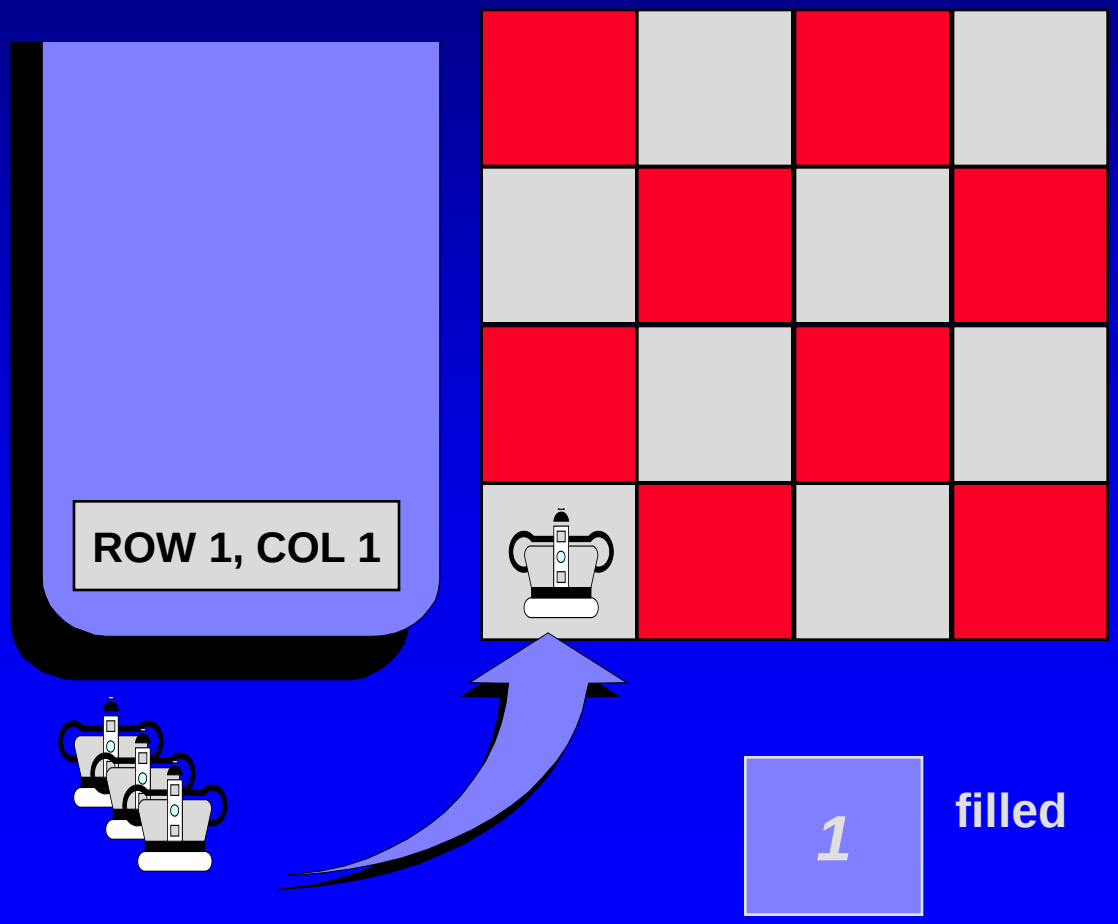
How the program works

Each time the program decides to place a queen on the board, the position of the new queen is stored in a record which is placed in the stack.



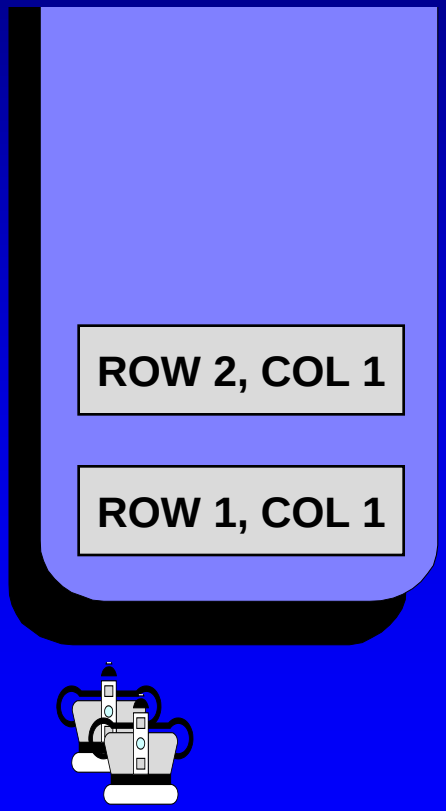
How the program works

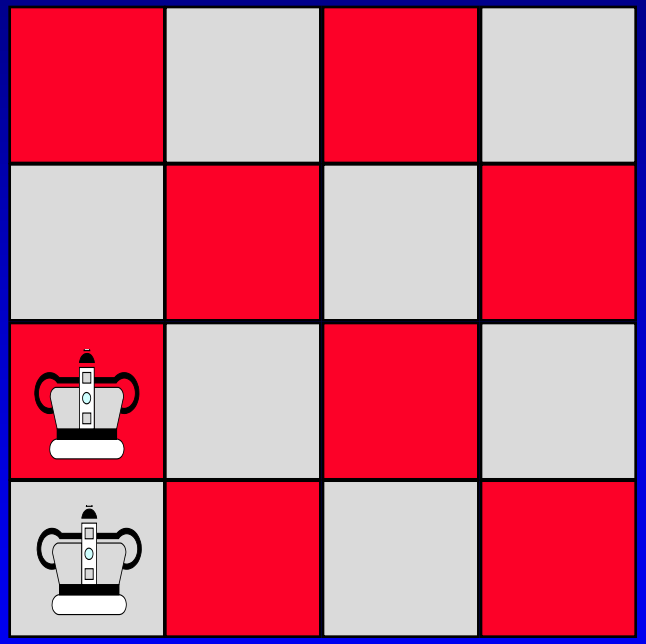
We also have an integer variable to keep track of how many rows have been filled so far.



How the program works

Each time we try to place a new queen in the next row, we start by placing the queen in the first column...

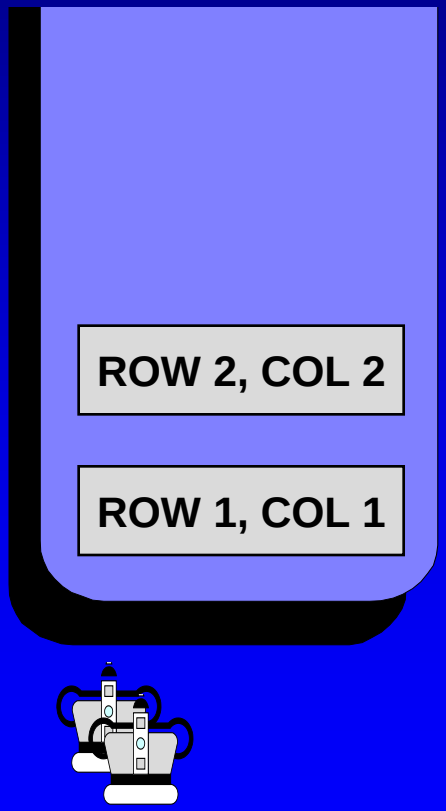


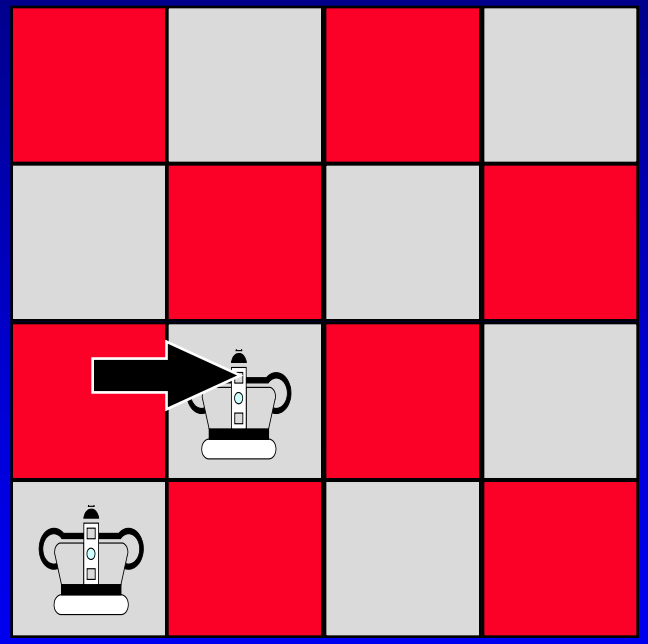


1 filled

How the program works

...if there is a conflict with another queen, then we shift the new queen to the next column.

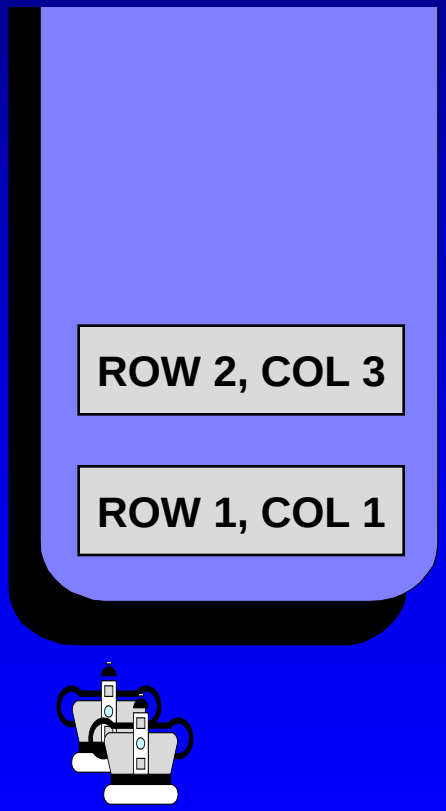


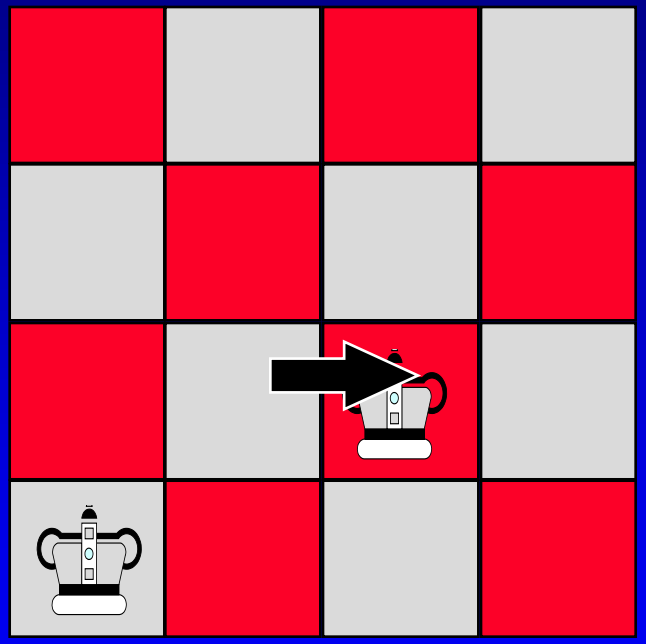


1 filled

How the program works

If another conflict occurs, the queen is shifted rightward again.

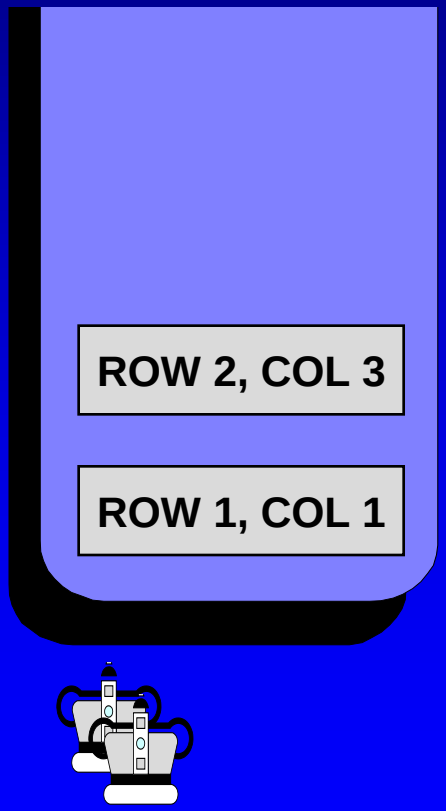


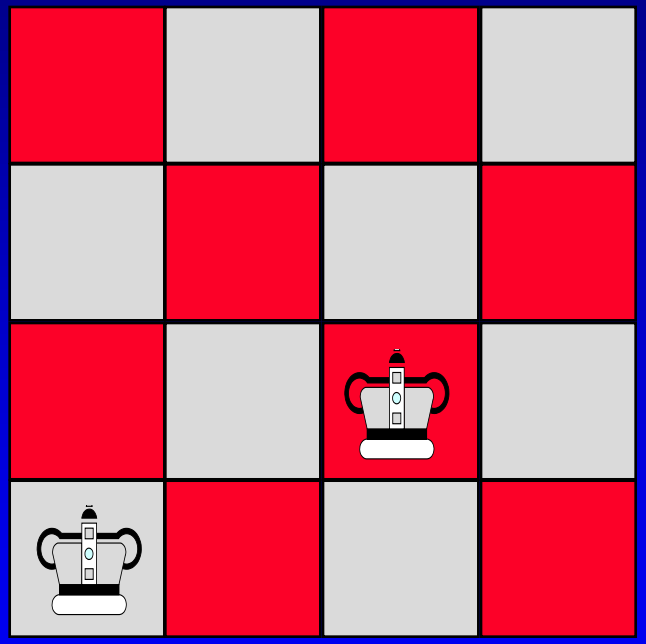


1 filled

How the program works

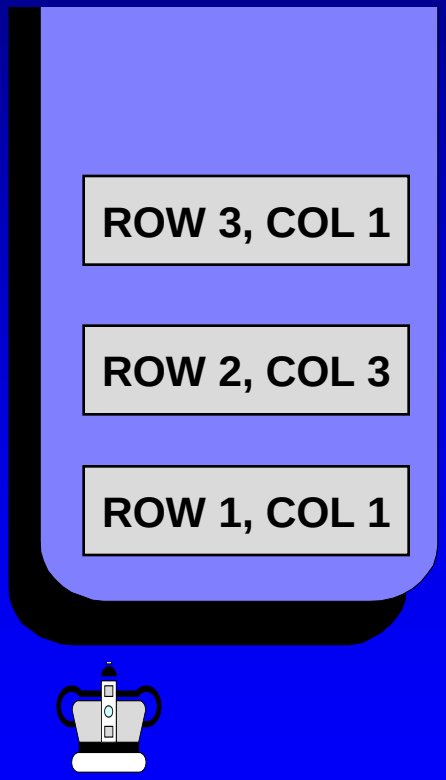
When there are no conflicts, we stop and add one to the value of filled.

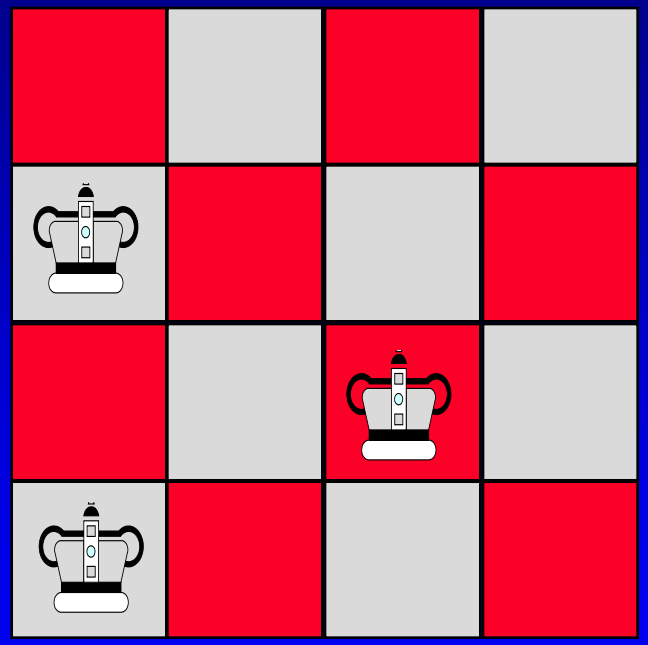




How the program works

Let's look at the third row. The first position we try has a conflict...

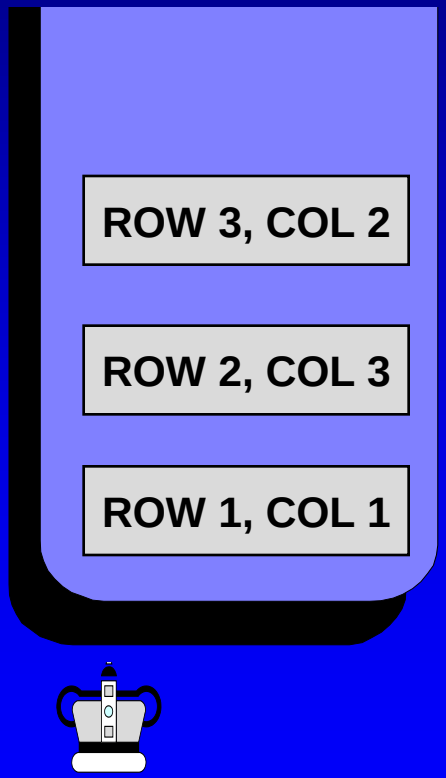


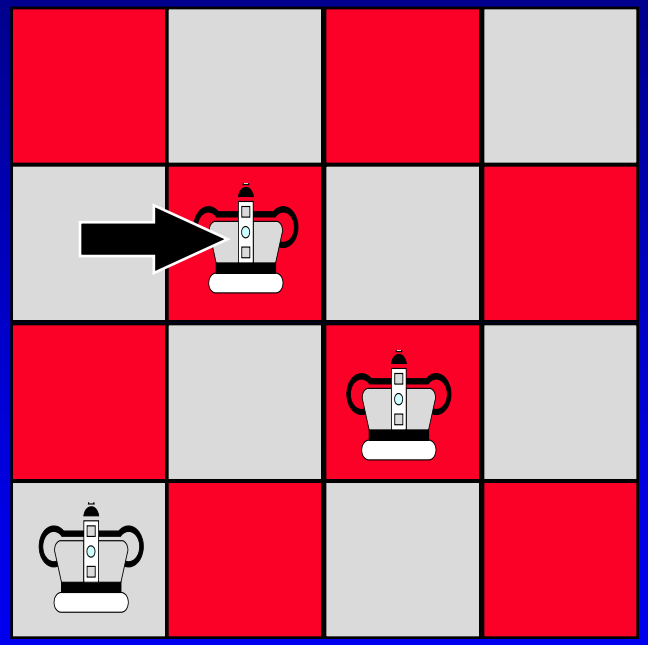


2 filled

How the program works

...so we shift to column 2. But another conflict arises...





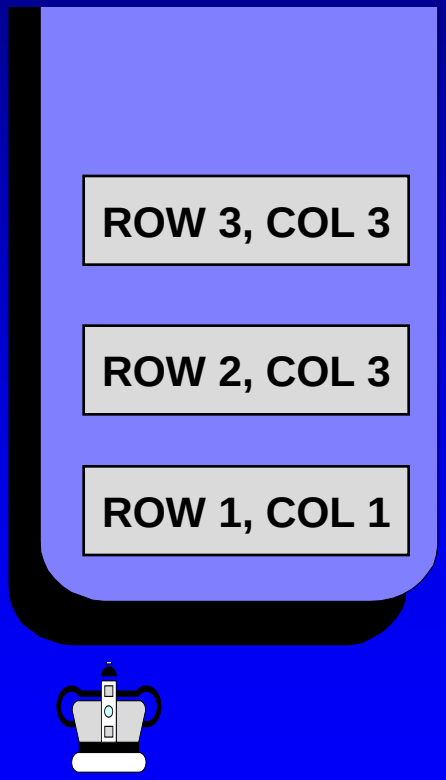
2

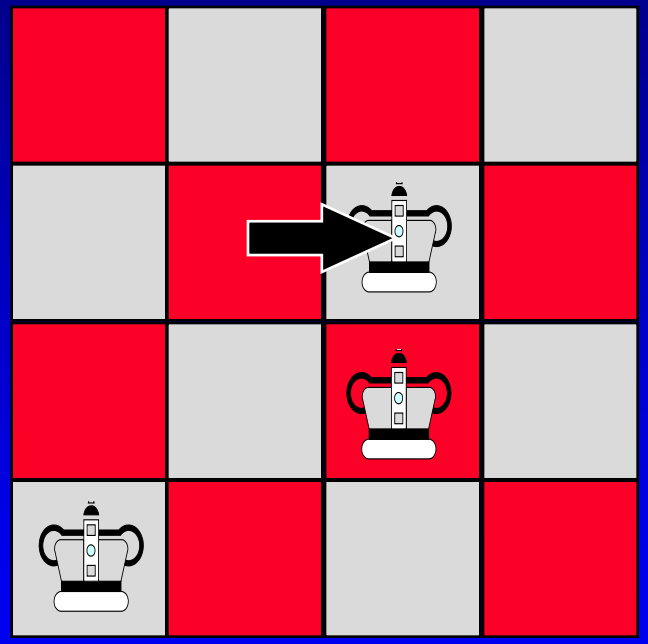
filled

How the program works

...and we shift to the third column.

Yet another conflict arises...



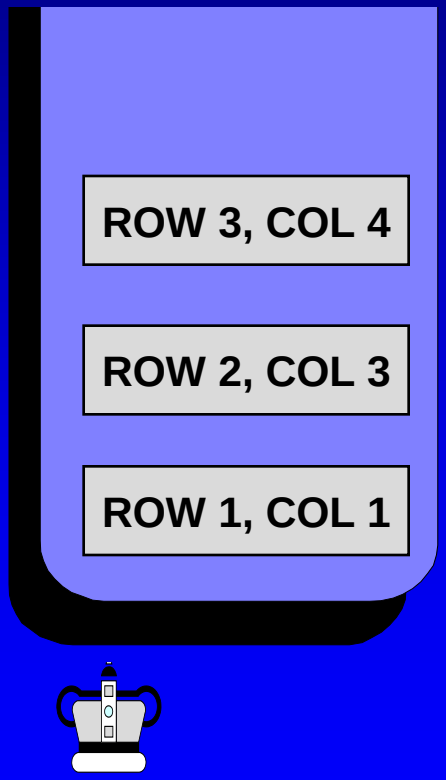


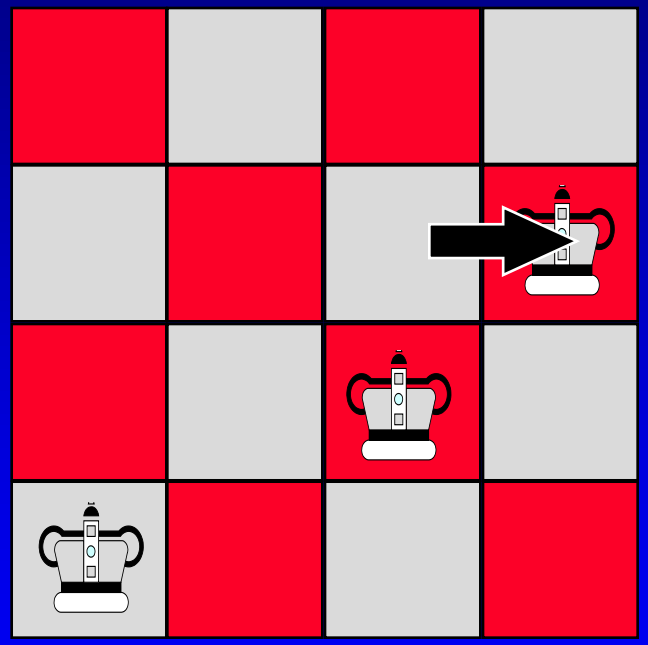
2

filled

How the program works

...and we shift to column 4. There's still a conflict in column 4, so we try to shift rightward again...



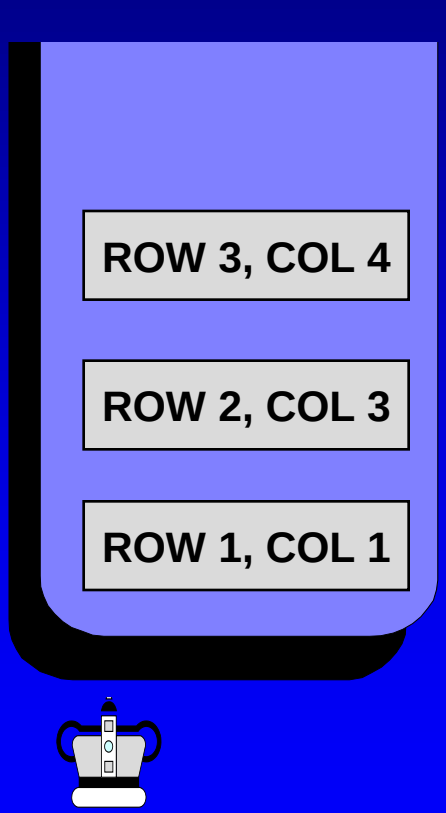


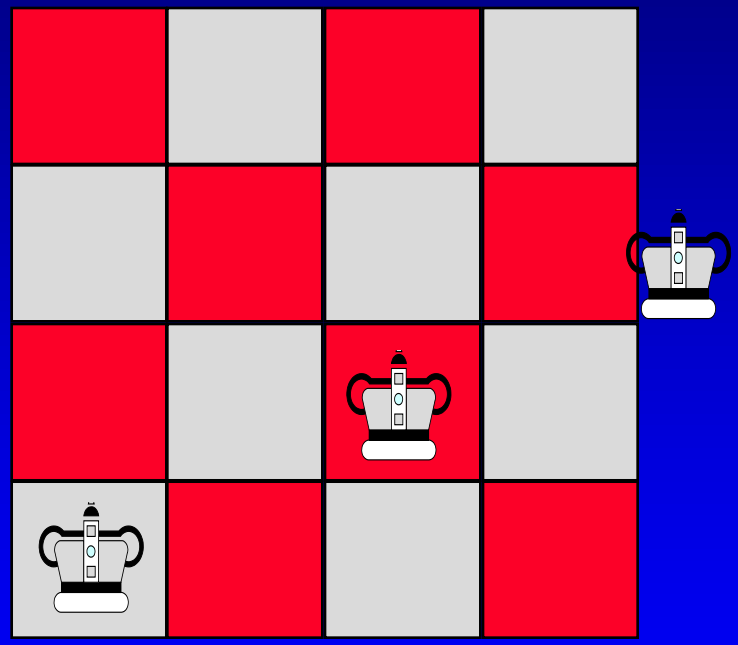
2

filled

How the program works

...but there's nowhere else to go.



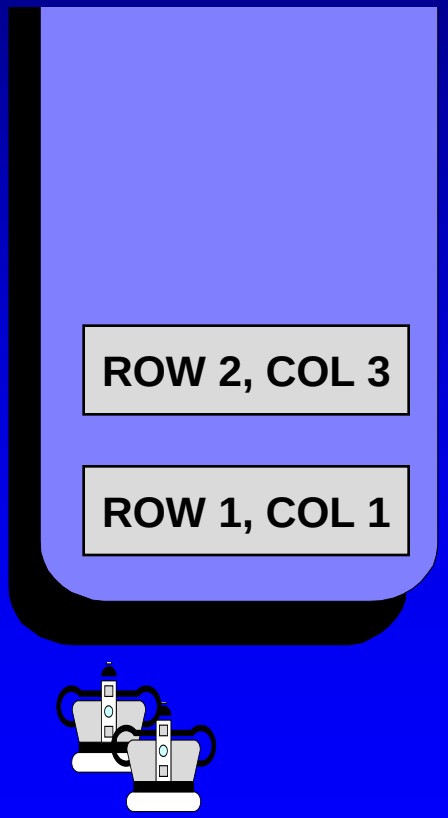


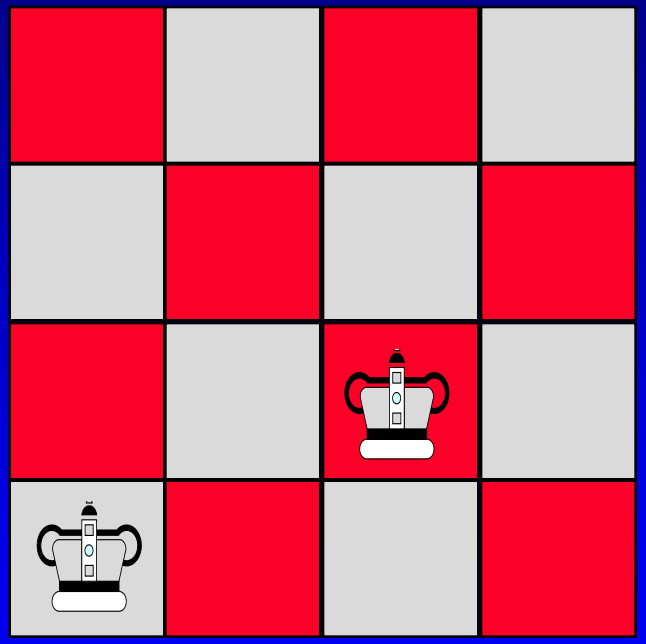
2 filled

How the program works

When we run out of room in a row:

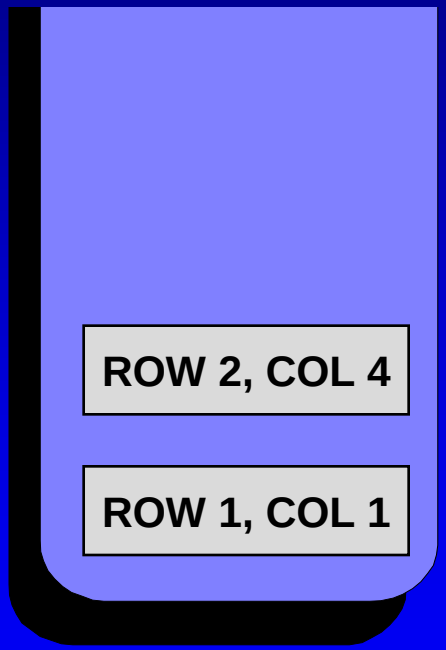
- pop the stack,
- reduce filled by 1
- and continue working on the previous row.

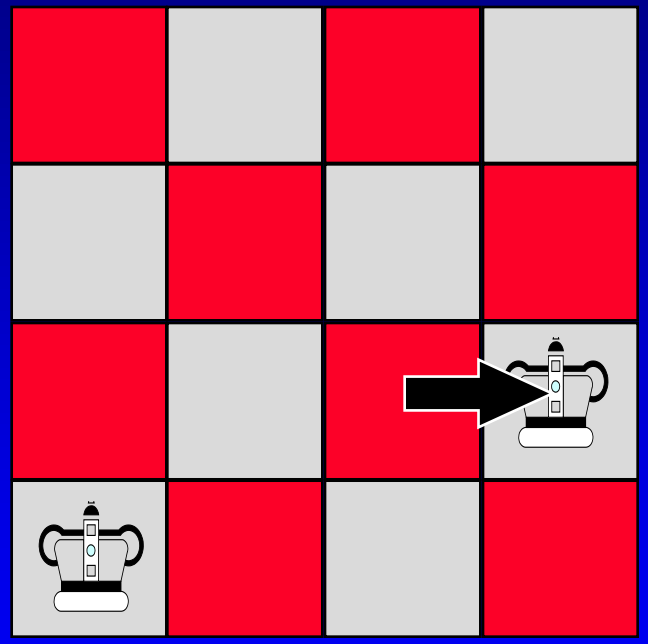




How the program works

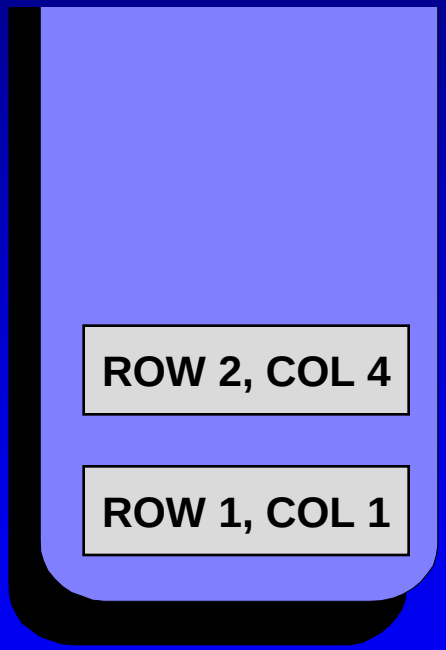
Now we continue working on row 2, shifting the queen to the right.

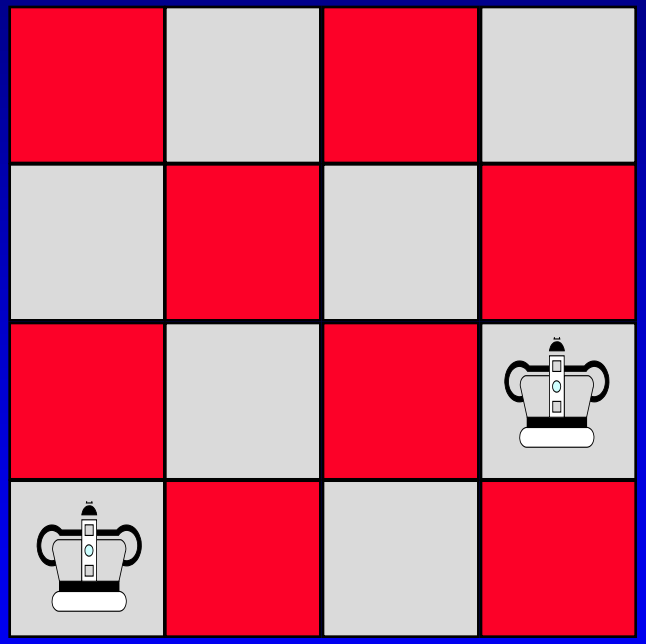




How the program works

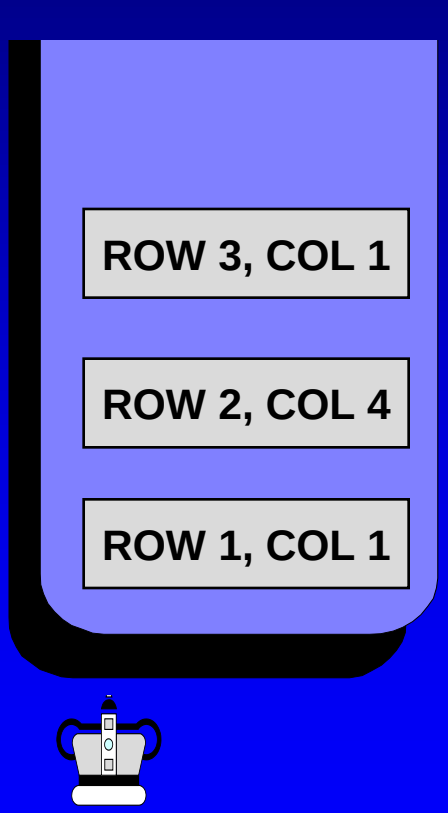
This position has no conflicts, so we can increase filled by 1, and move to row 3.

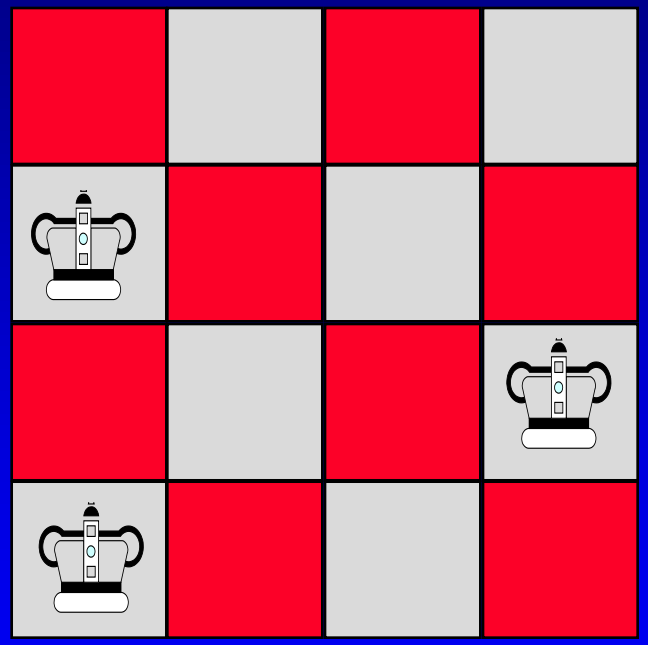




How the program works

In row 3, we start again at the first column.





2 filled

Pseudocode for N-Queens

- Initialize a stack where we can keep track of our decisions.
- Place the first queen, pushing its position onto the stack and setting filled to 0.
- repeat these steps
  - if there are no conflicts with the queens...
  - else if there is a conflict and there is room to shift the current queen rightward...
  - else if there is a conflict and there is no room to shift the current queen rightward...

Pseudocode for N-Queens

- repeat these steps
  - if there are no conflicts with the queens...

Increase filled by 1. If filled is now N, then the algorithm is done. Otherwise, move to the next row and place a queen in the first column.

Pseudocode for N-Queens

- repeat these steps
  - if there are no conflicts with the queens...
  - else if there is a conflict and there is room to shift the current queen rightward...

Move the current queen rightward, adjusting the record on top of the stack to indicate the new position.

Pseudocode for N-Queens

- repeat these steps
  - if there are no conflicts with the queens...
  - else if there is a conflict and there is room to shift the current queen rightward...
  - else if there is a conflict and there is no room to shift the current queen rightward...

Backtrack!
Keep popping the stack, and reducing filled by 1, until you reach a row where the queen can be shifted rightward. Shift this queen right.

Pseudocode for N-Queens

- repeat these steps
  - if there are no conflicts with the queens...
  - else if there is a conflict and there is room to shift the current queen rightward...
  - else if there is a conflict and there is no room to shift the current queen rightward...

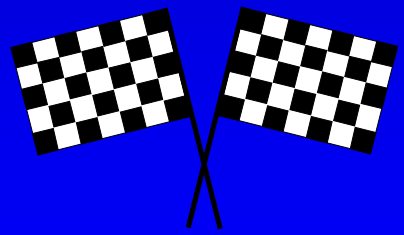
Backtrack!
Keep popping the stack, and reducing filled by 1, until you reach a row where the queen can be shifted rightward. Shift this queen right.

Watching the program work

You can double click the left mouse button here to run the demonstration program a second time:

Summary

- Stacks have many applications.
- The application which we have shown is called **backtracking**.
- The key to backtracking: Each choice is recorded in a stack.
- When you run out of choices for the current decision, you pop the stack, and continue trying different choices for the previous decision.



THE END