# [CS 11] Prac 8l – Chess Threats

## Problem Statement

Consider an $r \times c$ chessboard with some chess pieces in it. For simplicity, we ignore the actual rules of chess. For example, chess pieces don't have "color".

We say a square on the chessboard is **unsafe** if a piece can reach it in one move.

Here are the chess pieces we will consider, along with the way the move:

- In one move, a bishop can travel to any other square in either of its 45-degree diagonals, as long as there are no other pieces on the way there.
- In one move, a rook can travel to any other square in its row or column, as long as there are no other pieces on the way there.
- In one move, a knight can travel to any square that's an "L"-step away—that is, move two steps in one direction and then one step in the perpendicular direction. It can travel to that square even if there are pieces in between.
- In one move, a queen can travel to any other square in its row or column, or in any of its 45-degree diagonals, as long as there are no other pieces on the way there.
- In one move, a king can travel *one* square horizontally, vertically, or diagonally.

Given the locations of chess pieces, label which squares are unsafe.

Submit solution [CS 11] Practice 8

**Points:** 160 (partial)
**Time limit:** 6.0s
**Memory limit:** 1G

**Author:** kvatienza (Kevin Atienza)

**Problem type**

**Allowed languages** ~~NONE~~, py3

## Task Details

Your task is to implement a function called `chess_threats`. This function has a single parameter: a `tuple` of $r$ `str`s, each of which is $c$ characters long and represents a row. Each character represents a square and is:

- a `.` if it is free;
- a `B` if it contains a bishop.
- a `R` if it contains a rook.
- a `N` if it contains a knight.
- a `Q` if it contains a queen.
- a `K` if it contains a king.

The function must return a `list` of `str`s representing the same chessboard, but with the unsafe squares labelled with `*`.

## Restrictions

(See 8a for more restrictions)

For this problem:

- Loops and lists are allowed.
- Up to 18 function definitions are allowed.
- Recursion is **disallowed**. (The recursion limit has been greatly reduced.)
- Sets and dictionaries are allowed.
- Generators and comprehensions are allowed.
- The source code limit is $6000$.

## Example Calls

### Example 1 Function Call

```
chess_threats((
    'K.........B.',
    '............',
    '............',
    '...R....K....',
    '............',
    '...N.........',
    '............',
    '.....Q......K',
))
```

### Example 1 Return Value

```
[
    'K*.*.*.....B*',
    '**.*.*....***',
    '...*.*.****..',
    '***R****K*...',
    '.*.*.*.***...',
    '...N.*.*.....',
    '.*..***....**',
    '*****Q******K',
]
```

### Example 1 Explanation

**Hint:** You can print a grid of `str`s by doing:

```
for row in grid:
    print(row)
```

or by doing

```
print(*grid, sep='\n')
```

## Constraints

- The function `chess_threats` will be called at most $1{,}000$ times.
- The sum of the $rc$ across all calls will be at most $500{,}000$.
- $1 \le r \le 100$
- $1 \le c \le 5{,}000$

## Scoring

- You get $100$ ❤ points if you solve all test cases where:
  - $r, c \le 50$
  - the sum of the $rc$ across all calls will be at most $10{,}000$.
- You get $30$ ❤ points if you solve all test cases where:
  - there are at most $5000$ pieces.
- You get $30$ ❤ points if you solve all test cases.

---

## Clarifications

Report an issue

No clarifications have been made at this time.