Spring 2022 Computer Science 2 Program #2: Tentaizu

I may take some of our assignments from past programming contests, such as I am doing for this program. Since it is easiest, I will use the .pdf of the problem from the contest (separately posted) and then supplement it with a document like this, which explains what I would like students to submit and the format for students to use. This will usually be fairly consistent from assignment to assignment, but please do read the accompanying document for each assignment just in case there are some changes.

Reading Input/Producing Output

Please use standard input and standard output!!!

Your grade will be based on the number of test cases that are correctly solved within the time limit. Since there's a good chance that some programs may take too long on individual cases, it's best to process each case one at a time. I suggest the following set up, since the number of cases is in the input.

```
public static void main(String[] args) {

    Scanner stdin = new Scanner(System.in);
    int numCases = stdin.nextInt();
    for (int loop=1; loop<=numCases; loop++) {

        // Declare other variables here.
        // Read input for one case.
        // Process information.
        // Output answer for one case.
    }

}
```

How to Test Your Code

1. Type up a sample input file. (Strongly suggested to type your own beyond what is provided.)
2. Test on the command line as follows:

```
>>java tentaizu < myinput.in > myoutput.out
```

3. Either visually inspect the contents of myoutput.out to see if it matches the results you expect, or, if you know what the answers should be, store those in a separate file, say, `correct.out` and then use any file comparison tool to see if the files are the same or not. In windows, you can type fc at the command line:

```
fc myoutput.out correct.out
```

In a Unix system like Eustis, you can type:

```
diff -w myoutput.out correct.out
```

Note: The names of the files can be anything you want. Both fc and diff just compare the contents of the two files that are given to them as inputs.

4. If there are differences, inspect them more carefully using a tool such as WinMerge.

Implementation Requirements

This assignment is testing backtracking. Your code should step through, square by square, through the 49 squares and try two options:

1. Place a bomb.
2. Don't place a bomb.

If a square already has a number, you move on, since it's not allowed to have a bomb.

Clearly, trying all possibilities leads to 2^{49} possible options, which would take too long to evaluate. Thus, in some instances, you may be able to decide that either (a) a bomb MUST BE placed, or (b) a bomb can NOT be placed. In these instances, SKIP trying the option that is doomed to fail. This is what backtracking is and how it's utilized in this assignment.

Hint: When you place a bomb in square (r, c), it finalizes the number of bombs adjacent to (r-1, c-1).

If you code the problem appropriately, then any individual puzzle should be solved in under a second. Our input has 25 test cases.

What To Submit

For this assignment, please submit a single Java program named **`tentaizu.java`** which solves the posted problem. This program is sizeable enough that it should have multiple methods. The structure of it should be similar to other backtracking programs shown in class.