← Project 3: Sudoku solving

Due by midnight, Saturday, 11/10 (or late on Sunday)

> Many many thanks to Dr. Garrison for this project description and starting code. THANK YOU BASED BILLIAM

In this assignment, you will be implementing a **backtracking** algorithm to solve Sudoku puzzles. If you aren't familiar (or even if you are and need a refresher), **have a look at this site**.

Starting point

Download and extract these materials. Contained are:

- `Sudoku.java`, **the file you will modify.**
- 5 `.su` files, which are **plain text files** containing Sudoku puzzles to solve.
    - Since they're plain text files (like java files), you can open them in your text/code editor.

`Sudoku.java` contains the following:

- a `main` method which can accept some command-line arguments.
- the `readBoard` method, which reads a board from a file. `main` calls this.
    - "Unsolved" (empty) cells contain 0s.
- the `printBoard` method, which prints the board to the console, which will be very helpful.
- the `solve` backtracking template method.
- and finally, 8 "skeleton" methods for you to fill out.
    - you know, cause Halloween. 💀

Testing

We will be testing your code on the command line. You are welcome to use an IDE, but before submitting, please test that your Java files can be compiled and run on the command line without it.

We should be able to run your program like so:

```
java Sudoku -t
```

to run all the test methods, or:

```
java Sudoku 2-easy.su
```

to load and solve a Sudoku board file.

The `main` we gave you handles this command-line interface. But right now, nothing happens :)

Your task

This is a tough project. As such, **60% of the grade is for correctness, and 40% is for the thoroughness of your testing methods.**

You will implement four methods to solve the problem: `isFullSolution`, `reject`, `extend`, and `next`.

To complement those four, there are four testing methods: `testIsFullSolution`, `testReject`, `testExtend`, and `testNext`. Each of these should make **several tests** of the corresponding solving methods.

The puzzle-solving algorithm

By implementing the four solving methods, you will be constructing a backtracking solver program. Important points:

- You must not change any of the original numbers already on the board.
- You must implement all the Sudoku rules: each row, column, and 3x3 square may only contain each number once.

- You only need to find *one* solution, not all solutions. Once you find a solution, you can stop.
- If a board is not solvable, your program must say so.

The puzzle-solving methods

These methods make up the four parts of the backtracking template. These methods are **not recursive!** Only the `solve` method is, and we already wrote that for you ;)

- `boolean isFullSolution(int[][] board)`
  - This takes a board as a 2D array, and returns `true` if it is complete (no empty cells) and satisfies all the rules.
  - If there are any empty cells, or if there are any rule violations, it returns false.
- `boolean reject(int[][] board)`
  - This takes a *partial* solution, and returns `true` if it is **impossible** to continue with this board.
  - For example, if there are two 3's on one row, there is no reason to keep solving this board.
- `int[][] extend(int[][] board)`
  - This takes a *partial* solution, and constructs a **new** partial solution.
    - When I say "new" I mean use `new` to allocate a new 2D array!
  - What I mean by "constructs a new partial solution" is it **places a number into a previously-empty cell.**
  - If there are no more possible cells to place a number in, it should return **null.**
- `int[][] next(int[][] board)`
  - The partner to `extend`. This takes a *partial* solution, and constructs another **new** partial solution.
  - It will **change the most-recently-placed number** to the next possible option.
    - So if the most-recently-placed number was a 1, this would change it to a 2…
    - …or a 2 to a 3, or a 3 to a 4, and so on.
  - If there are no options for the most-recently-placed number, it should return **null.**

Do not write all the methods at once. Start with `isFullSolution` and `reject`. Test them extensively with your testing methods (by running your program with `java Sudoku -t`).

The testing methods

Re-read starting at Chapter 2.16 to review the concepts behind writing test methods. Write your testing methods *at the same time you write the solving methods*. Have a look at **this 8-Queens example** to see some example testing methods.

For each of the testing methods, try to follow these guidelines:

- Come up with a variety of partial solutions that will test all possible paths through your solving method.
- Call the method with those partial solutions.
- Check that the method actually returns what you expect it to return.
- Print out the test cases and results, so that you can easily see if things are looking right.
- Include enough test cases that you are convinced that your solving method is working properly.

You can make new board files following the guidelines below, and then use `readBoard` to load them for testing.

For example, if I were testing the `reject` method, I would give it…

- a board with two of the same number in the same row.
- a board with two of the same number in the same column.
- a board with two of the same number in a 3x3 square.
    - (probably a few versions of that, to be thorough.)
- the boards we gave you (`1-trivial.su` etc.) to make sure it *doesn't* reject those.
- a solved board (look online for one) to make sure it *doesn't* reject that.

Example boards

We've included 5 example boards (the `.su` files). If you open them in your text editor, you will see they look something like this:

```
3__79425_
2__56_197
79_82_4_3
_2__78_41
57_1_6_82
1483_2976
9_7_856_4
4526_7839
_164397_5
```

There are 9 rows, and each has 9 columns. Any non-number character is treated as an empty (unsolved) cell.

You can open one of the existing files and "File > Save As..." to make your own boards. Make a bunch! It's fine!

Submission

You will submit a ZIP file named `username_proj3.zip` where `username` is your Pitt username.

Do not put a folder in the zip file, just the following file(s):

- `Sudoku.java`
- Any *extra* boards besides the ones we gave you
- And if anything is wrong/not working, a plain text (.txt) file with notes to the grader about what is not working.

> Do **not** submit any IDE project files.

Submit to the Box folder at this link. Drag your zip file into your browser to upload. **If you can see your file, you uploaded it correctly!**

You can also re-upload if you made a mistake and need to fix it.