****

*Jani Heinikoski*

Visual Sudoku Solver Plan

Lappeenranta-Lahti University of Technology LUT

CT70A9110 Software Development Skills: Front-End

05.03.2021

**Table of Contents**

**Contents**

[1 Abstract 3](#_Toc65870059)

[2 Sudoku 4](#_Toc65870060)

[2.1 Sudoku’s definition in the project’s context 4](#_Toc65870061)

[2.2 Solving sudokus 4](#_Toc65870062)

[3 Graphical User Interface 7](#_Toc65870063)

[4 Reference List 9](#_Toc65870064)

# Abstract

This document is a preliminary plan for the Software Development Skills: Front-End course project. My course project is a web based visual (i.e., shows the user how the program solves the sudoku,) sudoku solver. The web application consists purely out of HTML5, CSS3 and vanilla JavaScript although the CSS3 is compiled from SCSS-files by the SASS pre-processor; the application is purely a front-end application. The application is going to be deployed to GitHub Pages when it is finished. The project and its source code are publicly [available at GitHub](https://github.com/jani-heinikoski/sds_front_end).

**Note** that everything in this document is merely preliminary planning, the finished project is likely to differ by quite a bit from this document.

# Sudoku

## Sudoku’s definition in the project’s context

In this project a solvable (and possibly true) sudoku is defined by the following rules:

* The sudoku’s area is exactly 9x9 cells. Each cell can have the following values where the epsilon represents an empty cell.
* The sudoku has at least 17 clues (i.e., prefilled numbers). 17 clues is the minimum, because there **does not exist any true** 9x9 16 clue sudoku (Emerging Technology from the arXivarchive page 2012).
* The 17 clues are correctly marked; same number does not appear in its 3x3 grid, column or row.

True in this context means that the sudoku has one and exactly one solution (Emerging Technology from the arXivarchive page 2012). This project will not check if the sudoku is a true sudoku due to the amount of time it would take but it will prevent the user from entering less than 17 clues (since such sudokus are proven to be not true and it reduces the brute forcing complexity a lot).

## Solving sudokus

Before we define how to solve a sudoku, we must define what is a solvable sudoku. In this project, a correct sudoku may have multiple solutions (although the solver only shows the first it finds). This project will **not** consider if the sudoku is a true sudoku.

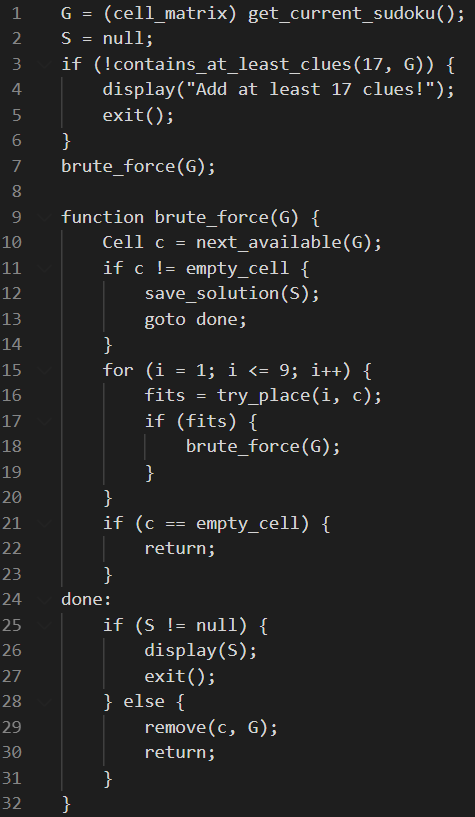
We know that there are no true sudokus that have less than 17 clues so the possible ways to fill the rest of the grid are . To prove that a sudoku is a true sudoku, we would have to show that the sudoku has only one and only one possible solution which would take an enormous amount of time using brute force.

The algorithm which solves the sudokus will be implemented in vanilla JavaScript. I know that there are much more efficient solving algorithms nowadays (e.g., exhaustive search routine) but I wanted to come up with my own algorithm. The solving algorithm I came up with is categorized as a backtracking algorithm (type of a brute force search algorithm). Preliminary plan for the algorithm as a verbal representation:

1. Let there be a partially prefilled 9x9 grid  that represents the sudoku matrix. The grid is prefilled by the user beforehand.
2. Check that the grid has at least 17 clues correctly filled.
3. Find the first available cell to fill.
   1. If there are no available cells
      1. Save the solution to and jump to 5.
4. Try to place each number to the cell found in step 3.
   1. If the number fits, leave it and jump to the step 3.
   2. If no number fits, backtrack.
5. Check if contains a solution.
   1. If it contains, display it and exit else remove the number found in step 3 and backtrack.

The verbal algorithm can easily be transformed to an abstract pseudo code (depicted in Figure 1). The main risk with the backtracking algorithm is that it is not efficient enough (takes too long to solve the sudoku). If the backtracking algorithm does not work, I have created another algorithm which might be able to solve the sudoku by itself or decrease the backtracking algorithm’s complexity. The other algorithm can be found in the same directory as this plan from the file “try\_solve.py”. The algorithm (written in Python) can be simplified as follows:

1. For 81 times (in case the sudoku is empty) do
   1. Get the next empty cell.
      1. If no empty cell can be found, print the sudoku’s current state. Try again if a number was filled during the last try (first replace the -1 cells back to empty) else return.
   2. Try to fit numbers from 1 to 9 in the empty cell. Increase a counter each time a number fits.
   3. If the counter is equal to one, insert the only fitting number to the cell else insert -1.



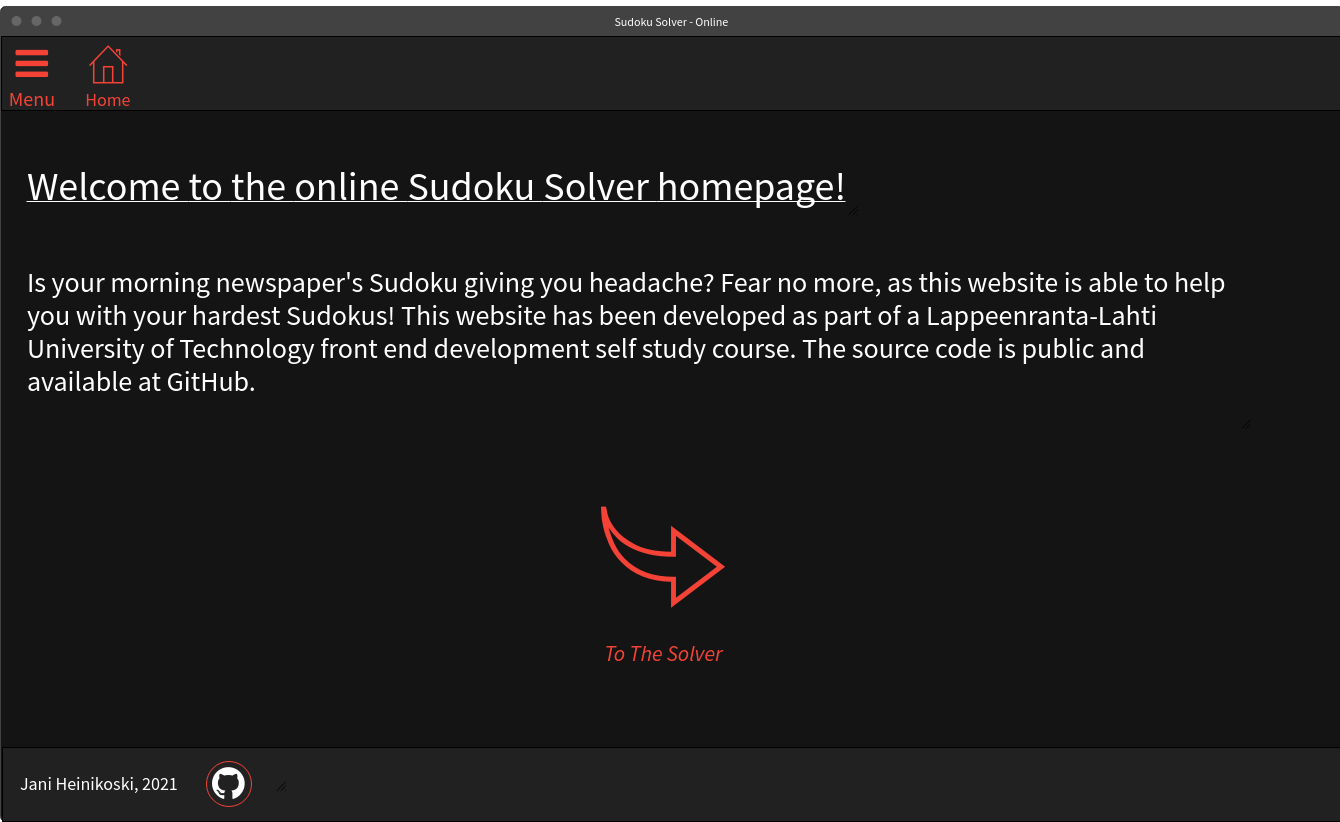
*Figure 1: Verbal backtracking algorithm as pseudo code.*

Note that all algorithms presented here are just rough plans and have obviously not been tested yet so expect changes in the final product.

# Graphical User Interface

The website is going to consist out of three pages: home page, solver page and about page. This plan will cover a mock-up of the home page only; rest will be designed during execution based on the home page’s theme dictated in this plan. The home page will contain a short abstract about the project and why it was created. The solver page will contain the visual sudoku solver itself. Finally, the about page will contain more detailed documentation about the project.

I have chosen to go with a dark theme, but the source code will be written so that it is easy to switch between light and dark themes. The preliminary homepage theme and style is depicted in the Figure 2 below.



*Figure 2: Preliminary homepage theme.*

The website’s homepage will have a link to the solver (see Figure 2) because it is the highlight of the website. The website will also have a “hamburger menu” button in the top-left corner of the website (in all pages) which will open a side navigation panel from the left side of the screen. The navigation panel will have links to all of the pages on the website. It will be similar to the one created in coursework (see coursework’s part 7).

The sudoku field is going to be created using CSS Grid and the sudoku’s cells are most likely going to be number input elements with some custom styling. The about page will mostly have just plain text content and hyperlinks.

# Reference List

* Emerging Technology from the arXivarchive page. 2012. Mathematicians Solve Minimum Sudoku Problem. [Website]. [Cited 05.03.2021]. Available: <https://www.technologyreview.com/2012/01/06/188520/mathematicians-solve-minimum-sudoku-problem/>
* Wikipedia. Revision 30th January 2021‎. Sudoku solving algorithms. [Website]. [Cited 05.03.2021]. Available: <https://en.wikipedia.org/w/index.php?title=Sudoku_solving_algorithms&oldid=1003749776>