Sudoku Generator & Solver

Final Project – Master’s Degree, 1st. Semester

Mario Contreras

ITESO

Master’s Degree in Computer Systems Student

Tlaquepaque, Jalisco, Mexico

ms705070@iteso.mx

Erick González

ITESO

Master’s Degree in Computer Systems Student

Tlaquepaque, Jalisco, Mexico

ms705070@iteso.mx

*Abstract*—Sudoku, a mathematical game, has gain popularity over the years. Sudoku players can find it challenging, but software developers can also find challenging to generate a valid and soluble grid, same for solving a Sudoku. We present a simple Java application that using backtracking algorithms can generate a new grid or to solve it.

Keywords—Sudoku; backtracking

# Introduction

Sudoku is a game that uses a 9x9 grid divided in regions. A Sudoku grid is a type of Latin square, which is a square *nxn* matrix filled with *n* symbols and symbols are restricted to be different in each row and column. Figure 1 shows a Latin square of 5x5.

A full Sudoku grid, also known as solution, contains numbers from 1 to 9 on each row and column, and each subgrid or region cannot repeat a number. From the solution, some numbers are removed and the cell is leaved as a blank cell. The cells that contains numbers are typically called clues. Figure 2 and 3 are examples of a solution grid and a game grid with 43 clues. The game consists of filling the blank cells without breaking the Sudoku grid constraints.

Fig. 1. A Latin Square

Sudoku is not the original name for this game. It was published by *Dell Pencil Puzzles and Word Games* in 1979 as “Number Place” [1]. A Japanese magazine published this game in 1984 and it was named as “Sudoku” that can be translated as “single numbers”. Today, Japanese use the original name while other regions use the word Sudoku.

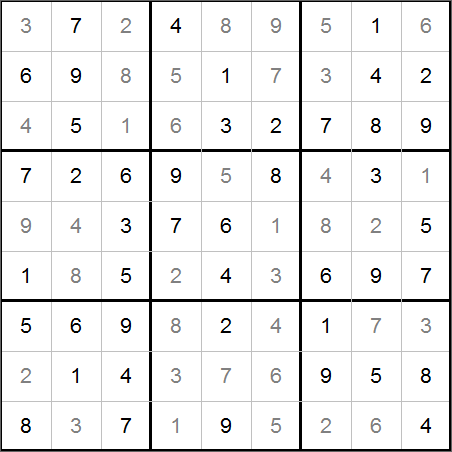
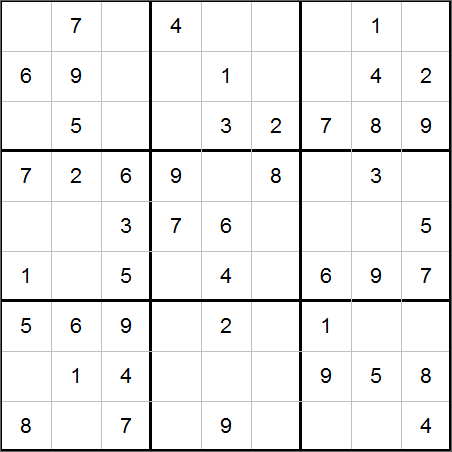
Solving a Sudoku is a challenging task, and it is more challenging if the number of clues is too small. The challenge is not a direct relation with the number of clues in the initial grid, but if we have more numbers to place, there are more mental operations to perform. Mathematicians have calculated the minimum number of clues that are required in an initial Sudoku with a unique solution is 17. McGuire et al., created algorithms to search for at least one 16-clue Sudoku but they couldn’t find it, therefore they concluded 17 is the minimum of clues required for a Sudoku [2].

Fig. 3. A Sudoku grid with 44 clues

Fig. 2. A Sudoku grid (solved)

# Math Background

In a *n*x*n* Sudoku, each cell can be 1..*n*. Therefore, a 9x9 Sudoku, which has 81 cells, could have 981 possible combinations. However, that’s if we don’t consider any of the Sudoku rules. First of all, a Sudoku is a Latin square, therefore we cannot use the same number twice in each row or column. In [1] we can find that there are 5,524,751,496,156,892,842,531,225,600 Latin squares of order 9. If all rules are considered, we reduce the number. We can also consider that if we take one row or column and swap it with another, it is essentially the same grid (an equivalent grid). If equivalent grids are removed, the number of grids according to Delahaye in [1] is 5,472,730,538, which is still too high.

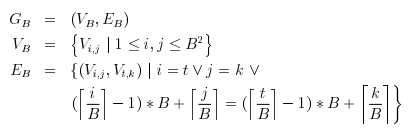
Due to the restrictions, one way to use graph coloring. The graph is formed with 81 vertices, one per cell in the grid. Edges represents a connection between numbers that form part of the solution. Yildirim et al represent such graph as shown in figure 4. In Figure 4, B is the order of the Sudoku puzzle (typically 9).

Fig. 4. Formal notation of a Sudoku graph by Yildirim et al.

# SudokuSolver GUI

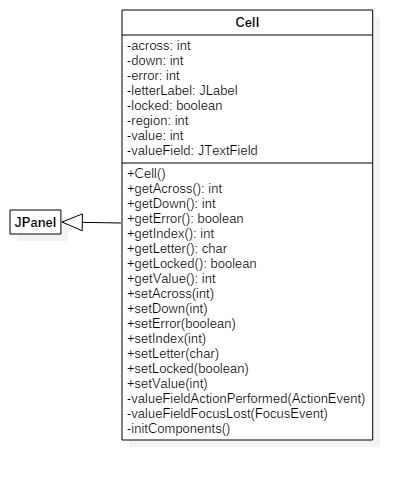
Our application was written in Java (target JRE 8u66) using NetBeans 8.1 IDE. We created a Cell class which inherits from JPanel. It represents a single cell in a Sudoku grid. Cell class contains a JTextField attribute named valueField that represents the number cell and a JLabel attribute named letterLabel to show the letter clue (S, M, L which stands for small, medium and large). Figure 5 is the Cell class diagram.

Fig. 5. Cell class diagram

A cell has a value (1..9) and a letter (S, M, L), but it also has a reference to the column (across) and row (down) in the complete Sudoku grid. The subgrid or region is also stored. If a cell represents a clue, it is considered to be locked.

Two valueField events were handled (actionPerformed and focusLost) to prevent invalid values (letters, symbols, etc.). We use getter and setter methods to control the cell appearance and to validate input. When a cell is locked (it’s a clue), the setter method named setLocked() change the text color and it also disables the text field. setError() is used to mark a cell as an invalid entry from the user and it will be rendered with red text. If the value is 0, setValue() will store 0 in value field, but the textfield will be empty. Figure 6 includes the setLocked() and setError() methods code.

Fig. 6. setLocked() and setError() methods

public void setLocked(boolean locked) {

this.locked = locked;

this.valueField.setEditable(!locked);

valueField.setBackground(

javax.swing.UIManager.getDefaults().getColor(

"TextField.background"));

if(locked)

valueField.setForeground(Color.BLUE);

else if(error)

valueField.setForeground(Color.RED);

else

valueField.setForeground(Color.BLACK);

}

public void setError(boolean error) {

this.error = error;

if(this.error)

valueField.setForeground(Color.RED);

else if(locked)

valueField.setForeground(Color.BLUE);

else

valueField.setForeground(Color.BLACK);

}

Cell class, and AboutDialog, a JDialog subclass, are used by SudokuSolver class. Figure 7 is a partial class diagram of SudokuSolver. SudokuSolver class inherits from JFrame and represents the main form. It contains 81 attributes named cell*xy* where *x* and *y* goes from 0 to 8. Those attributes represent the actual cells that are rendered in the form. The name property of those attributes were configured in design time with a 2-character string that represents their relative position. The SudokuSolver constructor uses the name property to populate the cells array which is used as a “shortcut” to manipulate the actual cells. It is easier to have an array of references and access the cells by index (cells[i]) instead of using the variable name (cell24).

We wrote a Difficulty enum (Easy, Medium, Hard) to represent the difficulty level. By default, the application starts with easy difficulty selected. Difficulty just represents how many clues the grid will have when a new game is created. Easy will contain 51 clues, medium 36 and hard will have 21.

The game’s basic options are new, hint, evaluate and solve. In order to demonstrate our algorithms, we split new in “New”, “New (Blank)” and “New (Solved)” and solve was split in “Solve” and “Solved (Backtracking)”. “New (Blank)” creates a new empty grid without any values. “New” generates a new full and valid Sudoku grid and it will be stored in solution array, but the player will show just a few clues. The cells array will contain some of the values (clues) and the rest will be cleared randomly. Due to the algorithm is too fast to see the progress, a copy of the algorithm was used in “New (Solved)” option which contains a delay between each number is selected so the player can see the progress when selecting numbers This copy was implemented in doInBackground method of NewGridTask class, which inherits from SwingWorker. This allows to work with the UI without locking so the user won’t think it was hung. Due to the backtracking logic used, numbers will be shown on screen then it will go back a few cells if a number has a conflict with the rest of the values. Due to this option will show all the values, it doesn’t really present a playable grid (all values will be filled) and it is for demonstration proposes only.

The backtracking algorithms to generate a new solution and to solve an existing grid, will be discussed in the next sections.

Fig. 7. SudokuSolver class diagram

# New Grid and New Game – Generate a New Sudoku Using Backtracking

Blah, blah… method, backtracking… blah…

.

# Solve (Backtracking) Option – Calculate the Solution Using Backtracking

##### References

[1] J.-P. Delahaye, “The Science behind Sudoku,” *Sci. Am.*, vol. 294, no. 6, pp. 80–87, 2006.

[2] G. Mcguire, B. Tugemann, and G. Civario, “There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration,” 2013.