SOLVING SUDOKU WITH MATLAB

Raluca Marinescu, Andrea Garcia, Ivan Castro, Eduard Enoiu

Mälardalen University, Västerås, 25.03.2011

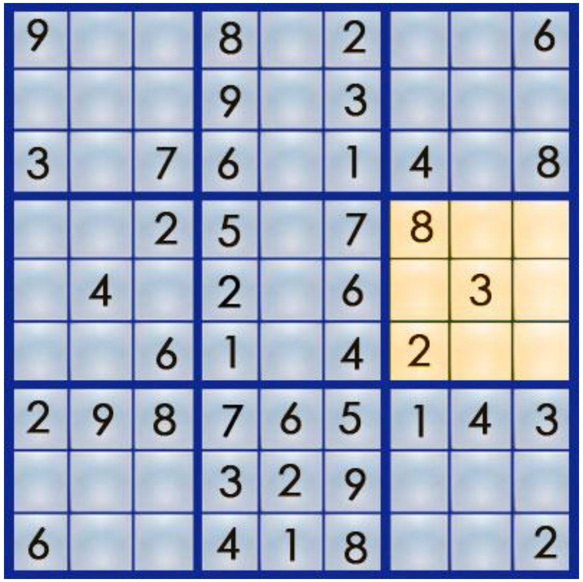
Background

- From the Japanese: ”SU”, means ”number” and ”DOKU”, means ”single”. The board is composed by a **9x9 grid**, sub-divided into 9 squares containing a **3x3** grid.

- The purpose of the game is to insert numbers (1-9) in the board **without repeating** a number in each row, column or sub-square.

- Sudoku is classified as an **NP-complete** problem, which means there is no known efficient algorithm to solve the puzzles.

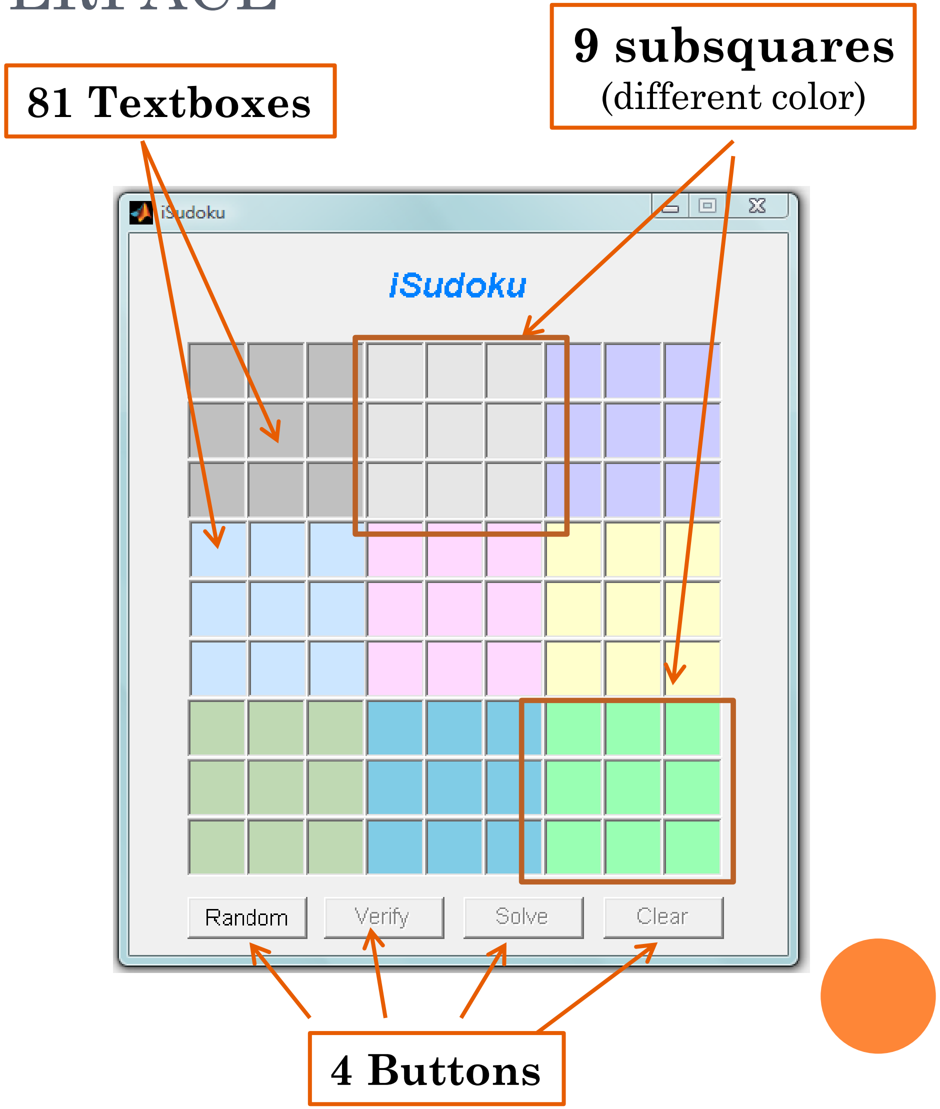
- It has led **researchers** to some advances in algorithm design and implementation.



9			8		2			6
			9		3			
3		7	6		1	4		8
		2	5		7	8		
	4		2		6		3	
		6	1		4	2		
2	9	8	7	6	5	1	4	3
			3	2	9			
6			4	1	8			2

Graphical User Interface

- Implemented using Matlab's GUI Design Environment (**GUIDE**).
- Used **drag & drop** components to create the layout.
- Each component has:
  - A list of properties that can be edited (color, size, position, etc)
  - **Callback functions** to model its behavior.



81 Textboxes

9 subsquares
(different color)

4 Buttons

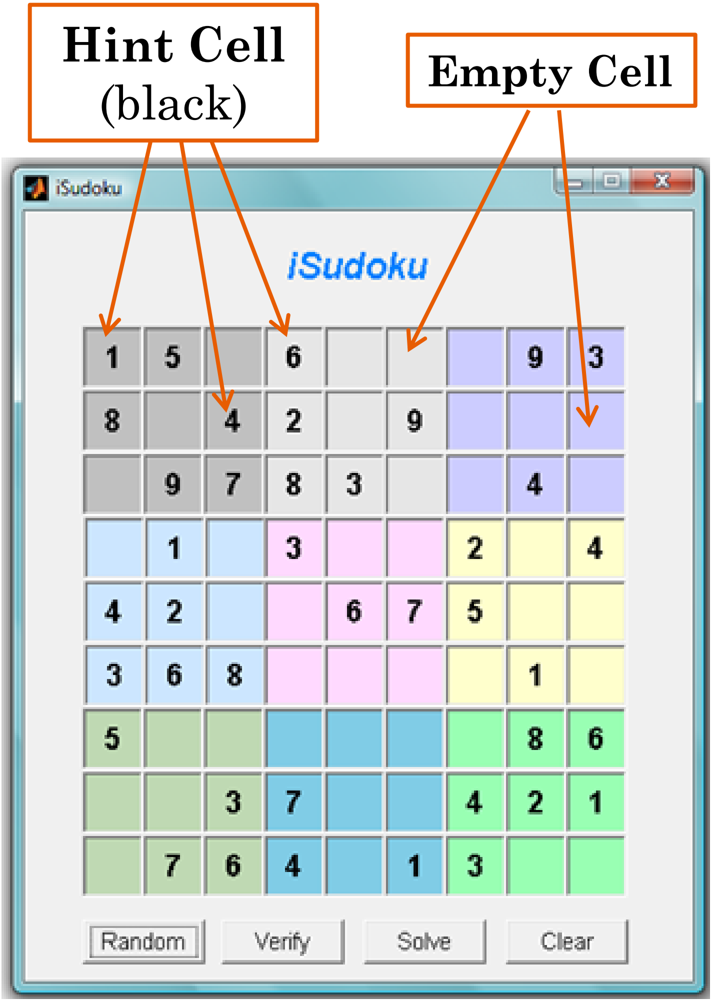
Random Button

Objective

- Creates and displays a **random game** on the board.
- Selects a game from a database of several games.

Characteristics

- The only button **enabled** when the application is **started**.
- Once a game has been displayed, the remaining buttons get enabled.



Hint Cell (black)

Empty Cell

1	5		6				9	3
8		4	2		9			
	9	7	8	3			4	
	1		3			2		4
4	2			6	7	5		
3	6	8					1	
5							8	6
		3	7			4	2	1
	7	6	4		1	3		

Hint cells cannot be modified *by the user.*

Solve Button

Objective:

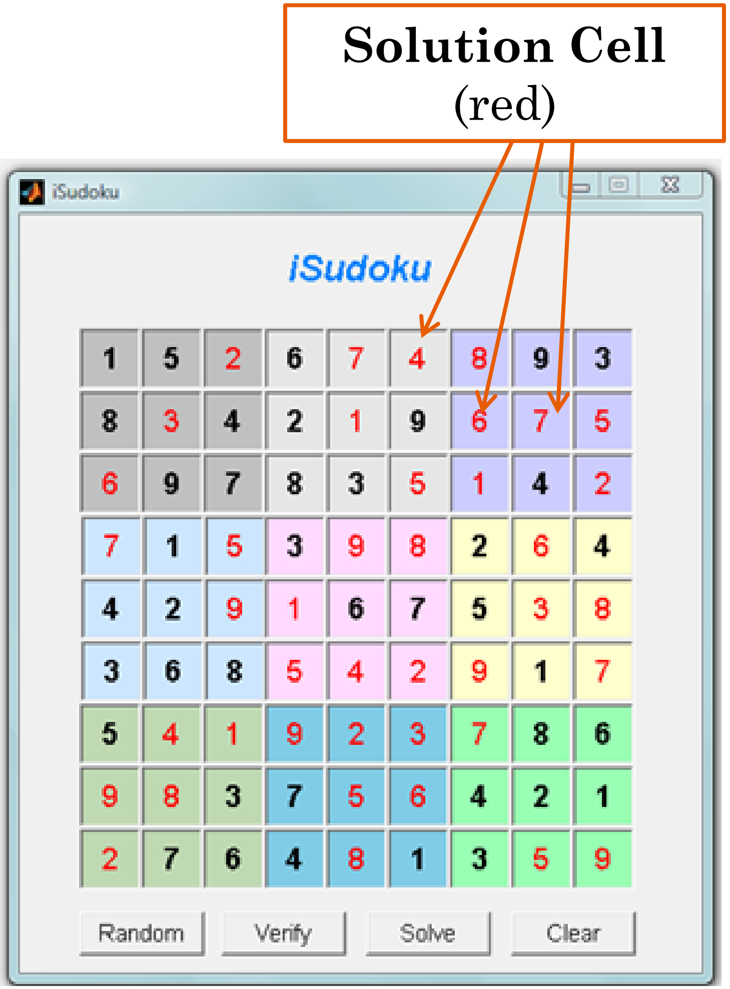
- Solves the current game and displays the solution on the board.

Three main steps:

1. **Read** the current game from the board and generate a **numerical matrix** of 81 elements. Empty cells are substituted by zeros. A **validation** of the **input data** is performed.*

2. **Execute** the Sudoku solver function, *iSudokuALG*, using the numerical matrix as an input.

3. **Retrieve** the solution provided by the Sudoku solver function and populate the board.



Solution Cell

(red)

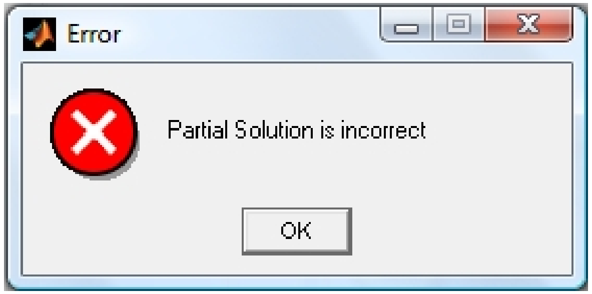
1	5	2	6	7	4	8	9	3
8	3	4	2	1	9	6	7	5
6	9	7	8	3	5	1	4	2
7	1	5	3	9	8	2	6	4
4	2	9	1	6	7	5	3	8
3	6	8	5	4	2	9	1	7
5	4	1	9	2	3	7	8	6
9	8	3	7	5	6	4	2	1
2	7	6	4	8	1	3	5	9

Only* ***integer values from 1-9* *can be inserted in the cells.*

Verify Button

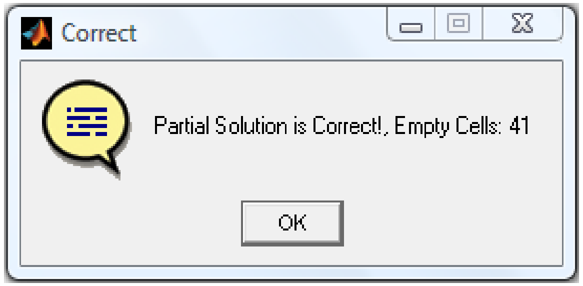
Objective

- Examines the **correctness** of either a partial game or a complete game.



Partial Solution is incorrect

In case of an **incorrect** game, the program will display a pop-up window with an **error message.**



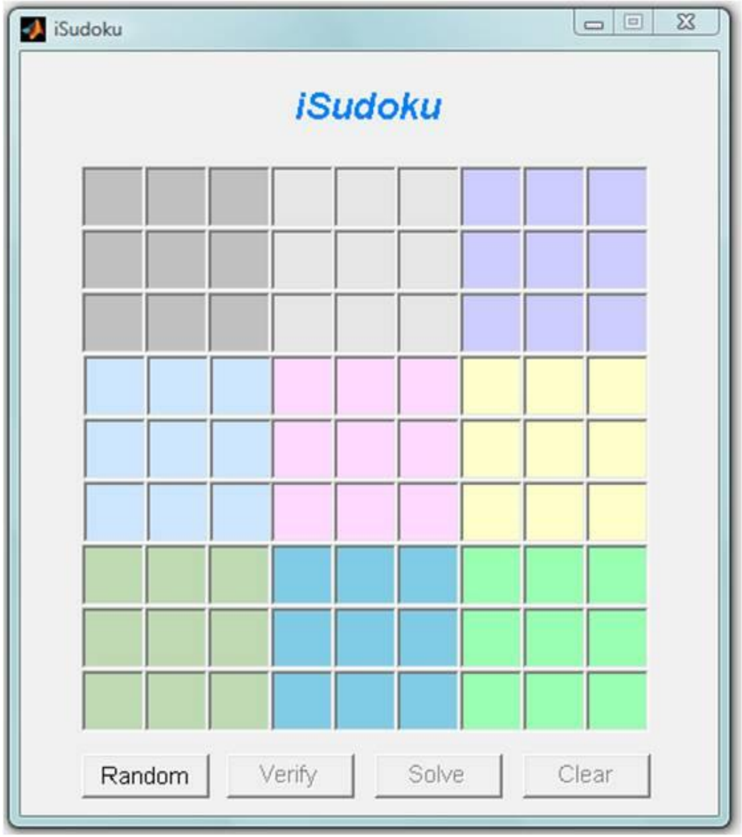
Partial Solution is Correct!, Empty Cells: 41

In case of a **correct** game a pop-up window will state this. In a partial game, the **number of remaining empty cells** will also be stated.

Clear Button

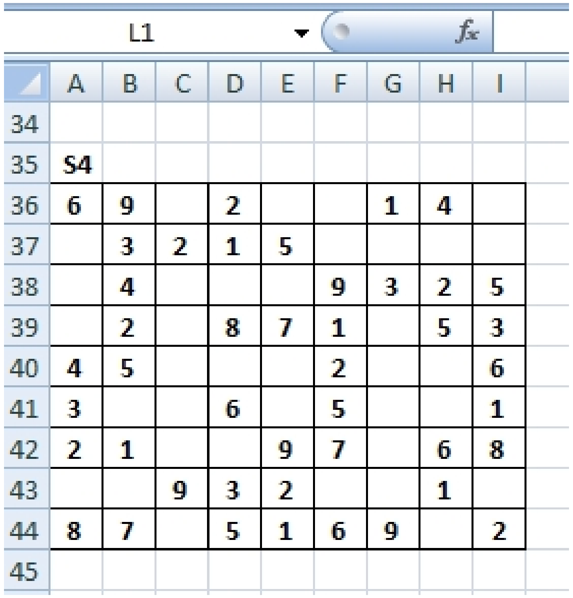
Objective

- It **clears** the board and **disables** all the buttons except the random button, returning the program to its initial state.



Game Database

- Created in to have different games to be **Microsoft Excel** solved.
- Includes games of **different levels** of difficulty.
- The games are **read** when the GUI is **initialized**.
- The reading process is performed by means of MATLAB's built-in function **xlsread**.



L1 fx

	A	B	C	D	E	F	G	H	I
34									
35	S4								
36	6	9		2			1	4	
37		3	2	1	5				
38		4				9	3	2	5
39		2		8	7	1		5	3
40	4	5				2			6
41	3			6		5			1
42	2	1			9	7		6	8
43			9	3	2			1	
44	8	7		5	1	6	9		2
45									

```
% Read predefined games from the input spreadsheet
[num, cellMat]= xlsread('sudoku.xls', 'Games');
```

VERIFICATION FUNCTION

- correctness verification of the puzzle:

```
function [val]=verific(A)
```

- checks if the current element appears twice in the same **line**, **column**, or in the **3-by-3 grid.**
- **Input:** the cell matrix A which contains the current puzzle.
- **Output:** variable ***val*** which can have two values:
  - 0 if the puzzle is correct
  - 1 otherwise

ALGORITHM

- Based on **constraint propagation**
- The key internal function is:

```
% Read predefined games and outputs the solved puzzle
function [A]= iSudokuALG(A)
```

- When a value is assigned to a cell that same value cannot be used as a possible assignment in all related cells;
- If a cell has only one single value for possible assignment, that value is immediately assigned.

ALGORITHM

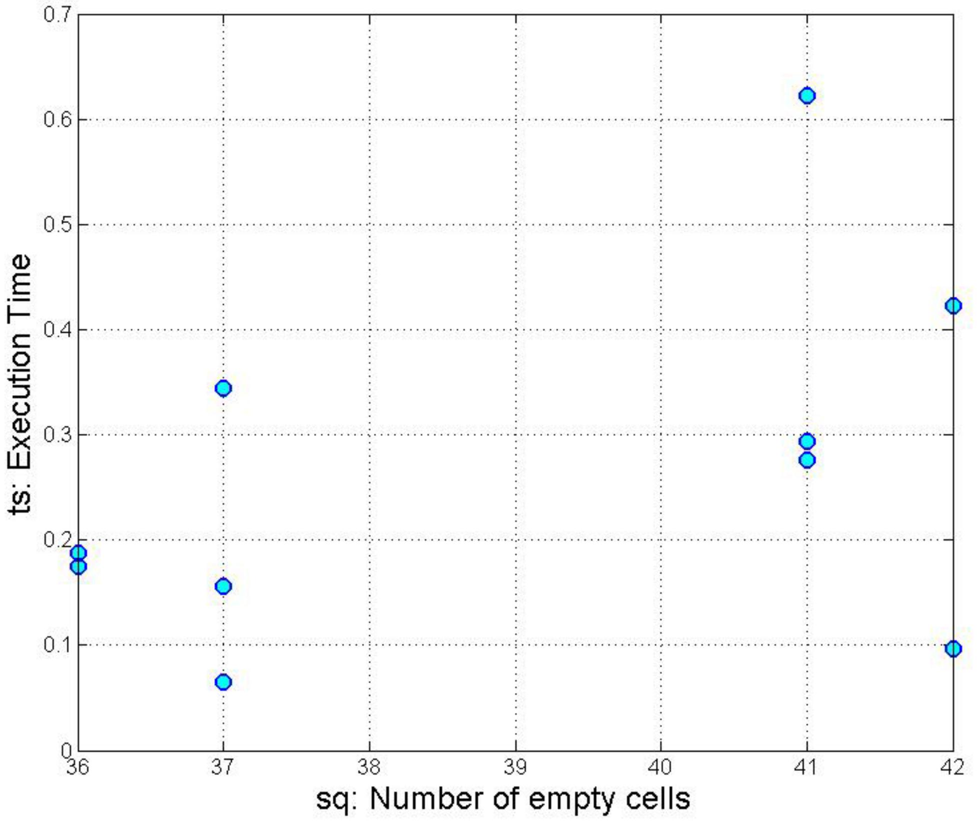
- **Steps:**
  1. Find all the possible values for all the empty cells;
  2. If there is a single possible value, we assign that value to the cell;
  3. Propagate constraints to other cells until you reach the end of the puzzle;
  4. If all the cells have more than one possible value we fill in a tentative value for that cell.
  5. *START AGAIN* ***(When do we stop?)***

ALGORITHM

- *When do we stop?*
  - When there are **no more empty cells** in the puzzle;
  - When for a cell we cannot place any possible value.

ALGORITHM TESTING

- Experimental results for different Sudoku puzzles:



CONCLUSIONS

- Implementing a Sudoku solver in MATLAB allowed us to use many tools and built-in functions presented during the course.

- The combination of a simple, yet effective algorithm with a graphical user interface allowed us to generate games, solve them and verify the given solutions in a simple and quick way.

- Good communication and coordination among the team members made possible the completion of the project before the established deadline.