Sudoku Solver 1st Year Computing Project.

Report.

Alexandre Benoit (ab7012)

Assignment Description: Design and Implement in C++ a Sudoku Solver. The program, given in input an incomplete but feasible Sudoku Grid, should find out how to fill in the blanks in order to obtain a valid complete Sudoku Grid.

Table of Contents:

I°/ User Interface
II°/ Input & Output
III°/ Algorithm (Solving Method)
IV°/ Modular Decomposition
V°/ Overview of Features of Sudoku Solver.
VI°/ What I learnt throught this project
VII°/ Test Inputs.
VIII°/ Schematic Diagram of Sudoku Solver.

Head of 1st Year Computing Laboratory:
Max Cattafi

I/ User Interface.

Oddly enough, the first thing I started thinking about as soon as I got the assignment was the User Interface. I really wanted to create a clean Graphical User Interface (GUI) for this Sudoku Solver. In my mind, it would greatly help the input of the Sudoku Grid and offer more design possibilities than just using the console / terminal.

I had to make a choice between how to do it. Even though it would be harder to do, I of course had to choose a cross-platform method that would enable the Sudoku Solver to work on Windows, OSX and also Linux. Many ways of doing so exist today and I will give a brief description of each and explain why I choose this one.

Qt : The first one we will look at is Qt. The last version I looked at has its own Integrated Development Environment (IDE) to be able to compile directly the programs inside a Qt window. A lot of the reviews I read about it describe Qt being very good but nevertheless, it takes time to adapt to the new IDE. It seems like a very profesional product.

wxWidgets. : Then, I had a look at wxWidgets. I found out that it only works with C++ (which is okay because I was planning on programming in C++ anyway). After reading a bit about it, I noticed that a lot of people complained about the fact that it was not

as cross-platform as they thought it would be and the same code could behave differently from platform to platform with plateform-specific bugs. I decided to find an alternative.

GTk+ / GTkmm : We are now going to talk about the solution I went with in the end: GTk+. On its own it work only for C but an add-on GTkmm integrates it with C++ (I had some problems with this at a start!). The advantage of it is that it behaves exactly the same on all platforms. This is an advantage for me as I wouldn't have tested my sudoku solver on all platforms to verify that it works correctly (I don't have OSX). According to reviews, it is very easy to use and I found a lot of tutorials related to Gtk+ & Gtkmm.

Quick View of Gtk+ functions:

- window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

The first "window" is called a Widget and defined using the function GtkWidget *window. The function after the equal sign creates a window top level (in front).

- GtkWidget *menu_item = gtk_menu_item_new_with_label(...);
  g_signal_connect (menu_item, "activate", G_CALLBACK(.....), window;

These two lines are used to create sub-menus by assigning labels using the first line and callback functions when the sub-menu is selected.

- Solve_button = gtk_button_new_with_label ("Solve Sudoku");

This line creates a button and using the g_signal_connect we can assign a callback function to it. The next line can change its size.

- gtk_widget_set_size_request (solve_button, [size]);

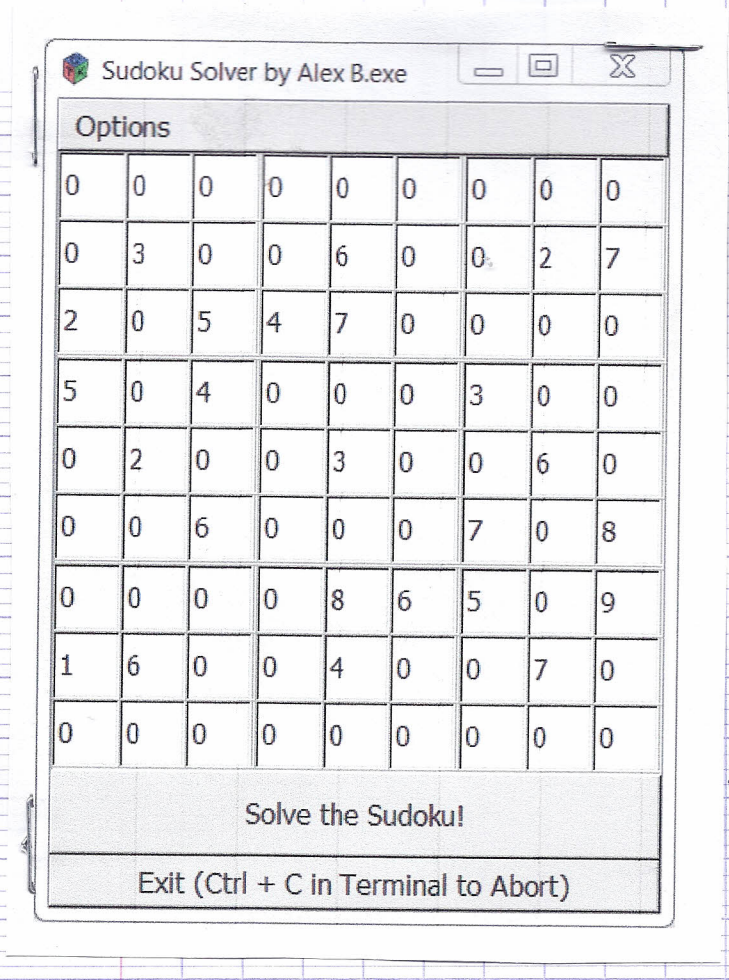


Figure 1: View of the GUI for the Sudoku Solver.

} boxes

3 seperator (shows blocks).

} button.

II°/ Input/Output.

Input : The two main ways of inputing the unsolved Sudoku Grid is to, or input it directly using the GUI, or opening a file that will automatically input the numbers into the GUI Grid. The user can choose between both using the drop-down menu "Options". In addition, he can reset the grid and erase all the entered value and reset them back to 0.

Output : As soon as the solve button is clicked. The program will search for an answer and then directly display it on the GUI grid. If you want to copy/paste this grid this is unconvinient. This is why (again using the drop-down "options" menu) you can save the solution/current grid to a .txt

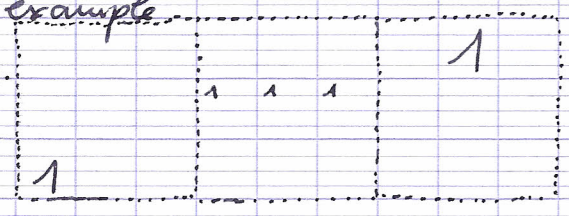
file in the program directory.

III°/ Algorith (Solving Method).

There are many ways to solve a Sudoku when using a computer. Of course some are more efficient than other, and others are simpler to code. I will seperate them into two groups and explain how they function. I will then backup why I choose my version of the code.

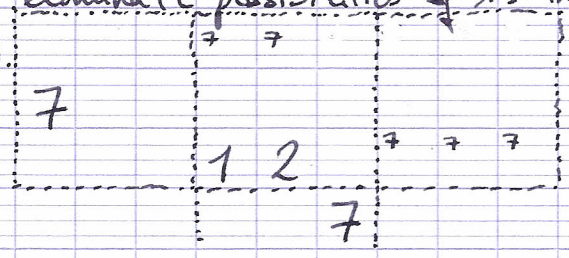
Logic Based: Being a good fan of Sudokus, I first though of doing this assignment using a "Pen & Paper" method. The algorith would do exactly as a human would do when solving a Sudoku. There are several 'logic moves' that one can do when solving a Sudoku. I will briefly explain 3 of them as an example.

1).



As we cannot have ~~&~~ two 1's on the same line/column, we can eliminate possibilities of 1's in the middle block.

2).



An alternative of the prievious logic move can be applied using triplets of numbers or impossible cells as sheilds. In this example, as no 7 can be placed at the bottom of the middle block, we can conclude that its placed at the top and bottom on the left block.

3) incomplete rows/columns/block can be filled out easily.

In addition to the fact that there may be more than 50 of these, I quickly found out how hard it was to code these 'logic' functions and was only able to program one function before I started to try to think of alternatives. This function (solve()) goes through the grid using two 'for loop' and tries out each value from 1 to 9. If it finds only one possible value that obeys the Sudoku rules regarding the other numbers, it inserts it into the grid, if not the cell remains 0. This function is really quick but was able to solve only the EASY sudoku's. I had to think of an alternative.

Try all numbers: The second group of algorithms are much more slower but are more effective. I would call them "try all numbers". It would basically try every number possible in the blank cells until the Sudoku is solved. There are nevertheless ways to make it faster. For example only insert numbers that comply with the Sudoku rules. In other words, don't even try a Sudoku where you inputed a 2 in a blank cell wher its row already has a 2. The method I choose is often refered to as backtracking. (solve_2(int i, int j)). It will again go through the grid and if the cell is already occupied, it will go to the next one. If it is free, it replaces the cell with a number and moves to the next cell and check if is free or not and continues. If it gets stuck, it starts again making previous cell's value return to 0.

In the end I decided to implement both at the same time that would enable my Sudoku solver to be very fast and still be able to solve all sudoku's possible. It took my program 0,003 seconds to solve the hardest sudoku ever made (Source: telegraph.co.uk). If you give

the program enough time it will be able to find a solution to all the input grids. If the input grid is wrong (in other words, 2 numbers on the same line for example) it will automatically display an error message.

IV°/ Modular Decomposition.

My source code contains the following functions: (other than main)

★ Functions that relate to the GUI:

1) Void initialize_GUI(int *; char ***)

This function opens the main window and creates all the widgets.

2) Void menu_event (GtkWidget *, gpointer)

This function assigns the different actions to the sub-menu widgets

3) Void open_error ()

This function is called when the Sudoku cannot be solved and in all cases due to wrong input numbers.

★ Functions that relate to the solve algorithm.

1) Void solve()

This is the main solve function that can solve easy Sudoku's and fill in the easy gaps.

2) void solve_2(int i, int j)

This function is called only if the first function was not able to successfully solve the Sudoku but keepts its current progress.

3) bool verif_1(int i, int j, int k)

Used by 1st solve function, when k isn't placed in grid yet

4) bool verif_2(int i, int j)

Used by 2nd solve function, when k is placed already in the grid and we only need to check if it's right.

These different functions made the program look simpler and helped me not get lost when I was programming (I hope it will

help you understand it!)

V°/ Overview of features of Sudoku Solver.

- Clean & effective Graphical User Interface (GUI)
- Different Possibilities with the Drop-Down Menu
- Double input from file or entered directly into the Grid
- Possibility to save solution to new .txt file
- Very effective algorithm
- Check errors in the input directly at the start of the program
- Display an error in a new window
- Can correct error after closing the error window (it doesn't quit app.)
- Can solve virtualy any Sudoku
- Clean Modular decomposition
- Displays in the terminal the time it took to solve the Sudoku.

VI°/ What I learnt through this project.

- How to use GTk+ / GTkmm to create a GUI
  - this involves creating menus / buttons / windows and developping relations between what the user does & response of program
- using the different libraries I was not familiar with
  - this includes the <cmath> & <time.h> library.
- using programing skills to solve real-world problems
- ameliorate my modular decomposition skills that I rarely used in past programs (they often became very confusing)
- creating 2D arrays / matrix (matrices) and input/outputing with them using double for loops!
- using static variables that last for the whole program
- increase strengh in domains such as input/output files & logic

VII°/ Test Inputs.

These test Sudokus can be found under Sudoku Solver / 2. Sudoku Examples. The first four (Easy, Medium, Hard, Evil) were taken from websudoku.com and the last one (Insane) is supposed to be the hardest one ever created (created by a Finnish Mathematician) and is from telegraph.co.uk.
Logic Solve: only using solve() Hard Solve: only using solve_2()

EASY.

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

logic solve → 0,002s

final version → 0,003s.

0,011s → hard solve

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

MEDIUM.

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

logic solve → N/A (but filled 4 squares)

final version → 0,015s

0,019s → hard solve

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

HARD

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

logic solve

N/A (but filled 1 square)

final version

0,022s

0,024s

hard solve

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

EVIL

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

logic solve

N/A

final version

0,724s

0,716s

hard solve

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

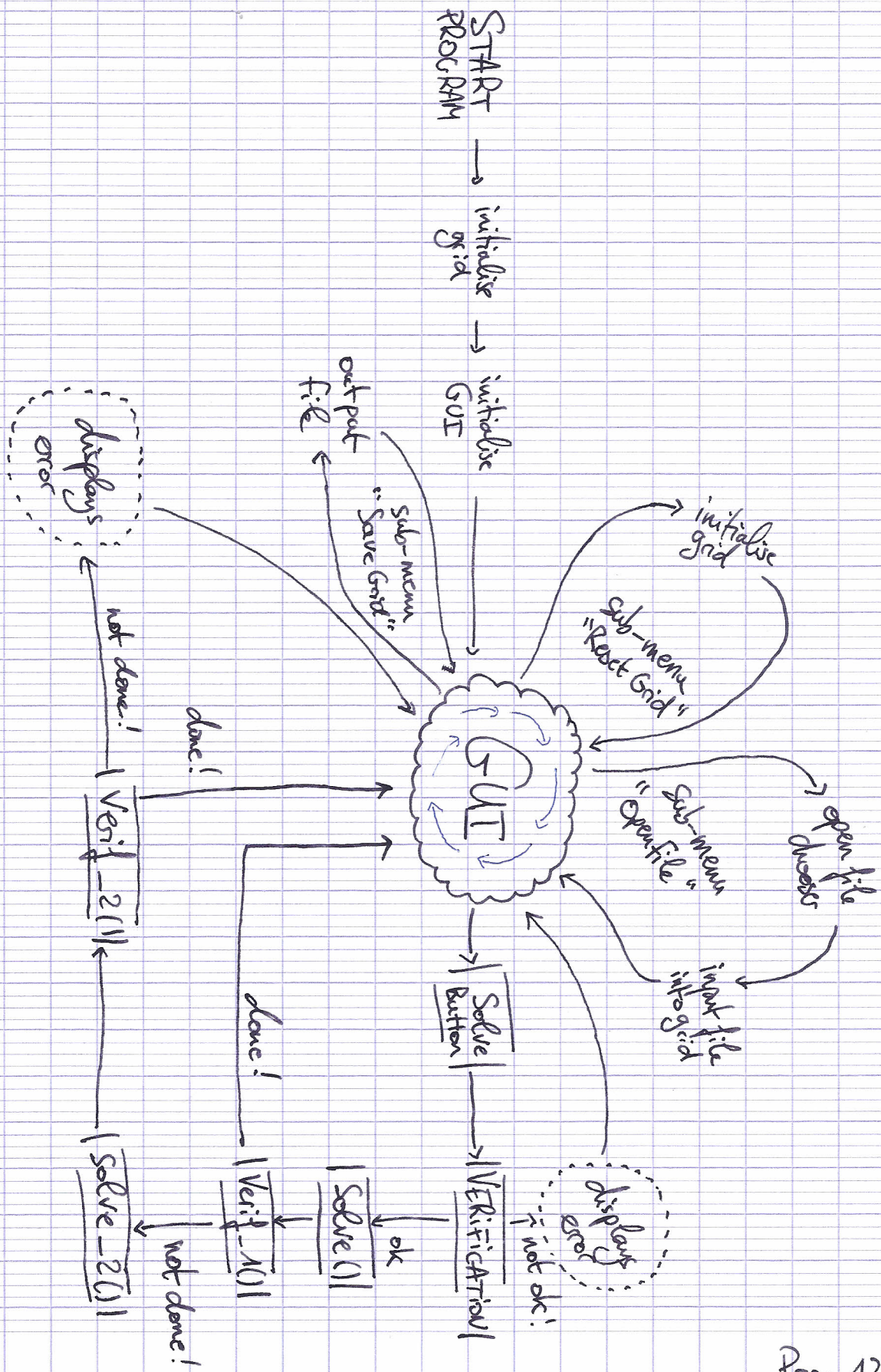
INSANE

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

logic solve → N/A

final version → 6,003s

5,995s → hard solve

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

VIII°/ Schematic Diagram of Sudoku Solver.