

CS 101

Project Report on

SUDOKU

(Sudoku-Game Cum Auto-Solver)

Team Members:

Roll numbers:

Amey Gupta (Team leader).....	140050026
Sumith.....	140050081
Pramod V.....	140050076
Lohith R.....	140050041

Instructors: Prof. Deepak B Phatak
Prof. Supratik Chakraborty
Autumn 2014
CLTA: Sadagopan N S

Abstract:

In this report, we present the detailed development and implementation of simple Sudoku game. The project is to create a program using C++ that can be used to play the game sudoku and also solve any sudoku given by the user. The sudoku game consists of Graphic User Interface, auto-solver, and sudoku-game.

The basic idea is to store all the inputs in the grid using multi-dimensional array and solve the sudoku using backtracking algorithm. It checks whether the input is a valid input as per the basic rules of sudoku and continues further.

History of Sudoku:

The name Sudoku comes from Japan and consists of the Japanese characters Su (meaning 'number') and Doku (meaning 'single') but it was not invented in Japan. Sudoku originated in Switzerland and then traveled to Japan by way of America.

Sudoku has its deep roots in ancient number puzzles. For many centuries people have been interested in creating and solving puzzles. Puzzles of all kinds continue to be the basis of developing important new mathematics. Completed puzzles are always a type of Latin square with an additional constraint on the contents of individual regions. For example, the same single integer may not appear twice in the same row, column or in any of the nine 3×3 subregions of the 9×9 playing board.

As there are so many Sudokus printed these days, surely all the possible grids have now been solved? Well you may think so. After a little thought it is clear there are quite a few new puzzles left and we are unlikely to run out of Sudokus in the near future

Fortunately some clever people have used super sized calculators to do the maths and claim there are 6,670,903,752,021,072,936,960 unique Sudoku grids of size 9×9 .

But if you then start determining symmetries including rotations and swaps then the number of 'effectively different' puzzles goes down to 5,472,730,538.

This large number means that if you solved one puzzle every second you would not need to repeat the same one in over a hundred years. These puzzles would all require different strategies to be used for their solution.

Algorithm for Auto-Solver:

Part 1:

- 3D array sudoku[9][9][10] in which sudoku[i][j] is an array consisting {‘value’,1,2,3,4,5,6,7,8,9} where ‘value’ gives the value at the corresponding cell.
- Zero or one are possible assignments to each element indexing from 1 to 9 of every element of array sudoku[i][j]
- Initialize each element to 1 indicating that all 1 to 9 are possible.
- Take input from user(while checking its VALIDITY).
- If the cell is filled(i.e sudoku[i][j][0]!=0),change all elements in array sudoku[i][j] except the first element, to 0 except the index with value same as that contained in that cell.
- According to the inputs, the possible values for remaining cells are eliminated using its row,column,grid.
- Filling the elements having only one possible value over entire matrix.

Part 2:

- Iterate over elements of matrix whose int value is 1.
- Insert the 1st possible value into element and fill the possible value(updated) onto 2nd element and so on till the last box is reached.
- If a break occurs in middle i.e, if for any element there are no possible values then backtrack the assignment.

Part 3: (backtracking)

- Go to the previous element and insert the next possible value and check the feasibility of matrix again.
- If again there are no possible values then repeat the process with next possible value till all the possible values are exhausted.
- If so repeat the above 2 steps again till we reach the final element.

Algorithm for Sudoku-Game:

Part 1:(for the checking part)

- The puzzle given to the user is solved by the auto-solver algorithm.
- The answer submitted is compared with answers generated by the Auto-solver.
- If the number in any particular does not match with the answer the corresponding square is highlighted by changing its colour to red.
- If all the squares match then the time taken by the user is compared with the scores in the high scores file.
- If the score is more than any one of the scores in the high scores then the high scores are updated.

Part 2:(for the solve part)

- The Algorithm for the solve part of the Sudoku-game is same as the Auto-solver.
- The initial sudoku that is given to the user to solve taken from the input files is solved by the above algorithm and the result is given as output along with the part submitted by the user as two sudoku grids in the same window

SCOPE FOR FUTURE WORK:

- There is scope for making a Sudoku generator(using an algorithm), in which the digits in the Sudoku (depending on the level), will randomly be generated and placed accordingly in the grid.
- There is a scope of using algorithmX (dancing links algorithm) for the autosolver part which is more efficient in comparison to back tracking algorithm.
- In the game, we could also check if the user is playing according to the rules after every digit he inputs.
- Also, we could highlight the conflicting entries (same digit in a row or a column or a 3X3 box) if there are any.
- A pause button could be done to pause the timer if the user wants to take a break in between the game.
- Saving a current game and loading a previously saved game could also be done.
- We could also have better graphics, like changing the colour of the background.
- Input for digits can be provided by mouse by selecting the input choices provided by us at the bottom of the sudoku.
- Many types of sudoku puzzles such as diagonal sudoku,jigsaw sudoku,grid sudoku and many others may be provided

Code Description:

The cpp files used are:

- 1)[main.cpp](#) : creates a new window in gtkmm.
- 2)[mywindow.cpp](#) : creates the main menu window containing the options New game, Auto-solver and High scores.
- 3)[mywindow_mem.cpp](#): creates a new window on clicking each of the options in main menu.
- 4)[mywindow2.cpp](#): creates all the buttons in the window for Auto-solver.
- 5)[mywindow2_mem.cpp](#): contains the algorithm for solving the sudoku and opening the corresponding window.
- 6)[mywindow3.cpp](#): creates the buttons in the window for new game(easy, medium, and difficult).
- 7)[mywindow3_mem.cpp](#): It creates the windows for each of the difficulty levels.
- 8)[mywindow4.cpp](#): It reads and writes the high scores of each of the difficulty levels.
- 9)[mywindow4_mem.cpp](#):It switches the tabs in the high scores window.

10)[mywindow6.cpp](#):creates a window

11)[mywindow6_mem.cpp](#): It contains the algorithm to solve and check the sudoku game and creates the corresponding windows for each of the difficulty level.

12)[mywindow7.cpp](#):creates a window

13)[mywindow7_mem.cpp](#): takes the input into a square of the grid and validates the input.

14)[mywindow8.cpp](#): creates a window

15)[mywindow8_mem.cpp](#): creates a window to give both the submitted sudoku and the correct solved sudoku by the computer in a new window on clicking the Solve button.

The corresponding header files used are:

1)[mywindow.h](#)

2)[mywindow2.h](#)

3)[mywindow3.h](#)

4)[mywindow4.h](#)

5)[mywindow6.h](#)

6)[mywindow7.h](#)

7)[mywindow8.h](#)

Acknowledgements

We wish to express our sincere gratitude to CS 101 Course Instructors Prof.D.B.Phatak, Prof.S.Chakraborty for their guidance and encouragement in carrying out this project.

We also like to thank our CLTA: Sadagopan for rendering his help during the period of our project work.

We like to thank Prof.A.G.Ranade for his book
-”An Introduction To Programming Through C++”

We will also like to thank:

- www.google.com
- en.wikipedia.org
- stackoverflow.com
- cplusplus.com
- blog.mpshouse.com
- MS word
- Google Docs
- Gnome Developers
- Contributors of gtkmm package
- Rohan Kumar(Batchmate)
- Shubham Goel(Batchmate)

