ERHVERVSAKADEMI
AARHUS

# Compulsory Assignment C# Fall 2019

Exercises Lecture_10

**Hand in no later than 10. October 2019**

## 1. Introduction

This assignment is about solving Sudoku puzzles. You must make a graphical desktop application that can load new puzzles, assist in playing the game, give hints, and if required solve the puzzle if the user can't.

You must show a working knowledge of how to write desktop GUI applications in C#. To make sure you have shown "working knowledge" you must write the game using either WinForms (Lecture 05) or WPF (Lecture 06 and Lecture 07), handle events, use external libraries, use a simple database for puzzles, and decouple your solver from the GUI application to prevent blocking.

Solving the Sudoku can be done using a solver provided to you as part of this assignment. But if you have time, you could try to write your own. It's fun! You don't have to generate puzzles yourself, you can load puzzles from a file. A good puzzle file is part of the assignment.

You can work on this assignment in groups of two students.

## 2. Detailed Requirements

Your program must contain at least the following features (REQUIRED)

- Load a puzzle from file
- Save puzzle to file
- Show puzzle in GUI
- User input: Create keyboard input of numbers
- Calculate solution if user can't finish on his own
- You must test that each number the user inputs is set correctly (This means that he must not set a number that makes the Sudoku impossible to solve!)

The report (REQUIRED)

- You must hand in a small 5-10-page report explaining your software.

You should implement this if you can (OPTIONAL)

- Hints for user
- Scalable Sudoku square (In WPF ViewBox is a great tool)
- Make the GUI look as good as you can!
- Load puzzle from database (Use ADO)
- Save puzzle to database (Use ADO)

If you like a challenge (OPTIONAL)

- Create small video where you demonstrate you application. You need to record what happens on the screen. CamStudio (https://camstudio.org/) is a good Windows solution to this problem.
- Write a sudoku solver from scratch (A working algorithm is described below).

# 3. Helper code

To solve this assignment, you need a working Sudoku solver. You can use the one provided to you as part of this assignment, but you can write your own if you prefer (see next section). If you write your own, make sure that it has the same interface as the one provided with the assignment. Interface used by provided solver:

```
public delegate void SolverOutput(string s);
public interface ISudoku {
    // Horizontal positions in ISudoku are counted from 0 to 8
    // Vertival positions are counted in the same way.
    // Upper left  corner : (0,0)
    // Lower left  corner : (8,0)
    // Upper right corner : (0,8)
    // Lower right corner : (8,8)

    // Get/set the current value at position (i,j)
    // The return value is number 1-9 or 0 if unset cell
    byte this[int i, int j] { get; set; }
    // Alternative set/get interface
    byte GetNumberAt(int i, int j);
    void SetNumberAt(int i, int j, byte number);

    // Conflicts are dublicate numbers in Rows, Columns, and Blocks.
    bool HasConflicts();
    // Get list of Non-conflicting numbers at position (i,j)
    List<byte> PossibleNumbers(int i, int j);

    bool IsSolved();
    bool IsSolvable();

    // Number of cells not containing numbers 1-9 yet
    int NumberOfOpenCells();

    // 81 - NumberOfOpenCells
    int NumberOfFilledCells();

    // Each unfilled position has a number of
    int OptimalPosition(out int I, out int J); // returns number of possible,
                                               // and the best place I,J
    // Make a copy of the Sudoku
    ISudoku Clone();

    // Returns a solution
```

```
    ISudoku Solve(SolverOutput log = null);
    string ToString();
}
```

# 4. Extra Assignment

If you are up for the challenge, you should write your own Sudoku solver. You can use the algorithm below. Before writing the solver, I suggest you create a helper class to take care of representing your Sudoku square data.

Algorithm:

```
Load puzzle
While (not solved):
    Find the number of possible numbers in each un-filled cell
    Find the un-filled cell with the fewest possible numbers
    (If the fewest possible numbers is zero the Sudoku can't be solved)

    if (only one possible number in cell with fewest numbers):
        Insert number and continue
    else if (more than one number possible):
        Make a copy of the Sudoku. One for each possible number.
        Try to solve each of these recursively.
        (Skip the ones that can't be solved)

If you are left with a Sudoku with all cells filled and no conflicts, you
have solved the problem.
```

# 5. Hints on how to solve this problem

This section clarifies how you should approach this assignment if you are in doubt.

You can solve these assignments to get started:

**Startup exercise 1:**
Find the file "top1465.txt" in the assignment file and make sure you understand how each line in this file translates into a Sudoku puzzle. You could read the file one line at a time and write the 9x9 sudoku puzzle to the Console. The file uses a dot (a period) to indicate that a number is not set.
(The file can also be found here : https://github.com/vlvovch/sudoku-solver/blob/master/sets/top1465.txt)

(Nice console printouts can be seen here: http://forum.enjoysudoku.com/question-about-top1465-t4066.html)

**Startup exercise 2:**
Create a small Console project and add a reference to the DLL SudokuLib.dll from the assignment files (copy the DLL to somewhere inside your console project). You must add reference by browsing for the assembly.

In your project references in the solution explorer right click on SodokuLib and "view in object browser". Look at the methods the library offers.

Now try to use the DLL like this:

```
static void Main(string[] args) {
    // Lots of Sudoku here:
    // https://github.com/ralli/sudoku/blob/master/data/top1465.txt
    // https://www.telegraph.co.uk/news/science/science-news/9359579/Worlds-hardest-sudoku-can-you-crack-it.html
    // http://lipas.uwasa.fi/~timan/sudoku/
    // 17-cell Sudoku (No valid Sudoku has fewer cells filled)
    // http://theconversation.com/good-at-sudoku-heres-some-youll-never-complete-5234

    // Loading a Sudoku. This Sudoku uses '0' as indication of unfilled cell
    // You must translate to this format your self
    var s = SudokuFactory.CreateSudoku("042000005000632080080040200000000000715068340908350761091006000000020190006100050");
    Console.WriteLine(s);

    // Print how many cells are filled/unfilled
    Console.WriteLine("Filled Cells : " + s.NumberOfFilledCells());
    Console.WriteLine("Open   Cells : " + s.NumberOfOpenCells());
    Console.WriteLine();

    // Do we have any obvious conflicts?
    // column, row, and local block conflicts?
    Console.WriteLine("s.HasConflicts() = " + s.HasConflicts());

    // What numbers can be placed at position 0,0 (remember we count from 0 and not from 1)
    Console.Write("s.PossibleNumbers(0, 0) : ");
    foreach (var n in s.PossibleNumbers(0, 0))
        Console.Write(n + ", ");
    Console.WriteLine();

    // At what position are the fewest possible numbers?
    // (Easiest place to start filling in)
    int I = 0;
    int J = 0;
    int numberCount = s.OptimalPosition(out I, out J);
    Console.WriteLine($"{numberCount} numbers possible at optimal position {I},{J}");
    if (numberCount == 1) {
        int n = s.PossibleNumbers(I, J)[0];
        Console.WriteLine("Only one number possible, so you must fill this cell with the number " + n);

       // Lets do this, by the way
       s[I, J] = n;
    }
    Console.WriteLine();

    // Solve the Sudoku (if possible) and dump the result
    var s_clone = s.Clone();
    var s_solved = s_clone.Solve(); // You could try to give this delegate to Solve : (txt) => Console.WriteLine(txt)

    if (s_solved != null) {
        Console.WriteLine("SOLVED");
        Console.WriteLine(s_solved);
    } else {
        Console.WriteLine("No solution exists");
    }

    Console.WriteLine("The original Sudoku is still here:");
    Console.WriteLine(s);

    Console.ReadLine();
}
```

**Startup exercise 3:**

Try do print the numbers in a Sudoku like this:

```
Console.WriteLine("Manual printing of sudoku numbers:");
for (int r=0; r<9; ++r) {
    for (int c = 0; c < 9; ++c) {
        if(s[r, c] == 0) // or s.GetNumberAt(r, c) == 0
            Console.Write(".");
        else
            Console.Write(s[r, c]);
    }
    Console.WriteLine();
}
```

Try to set one of the open cells like this:

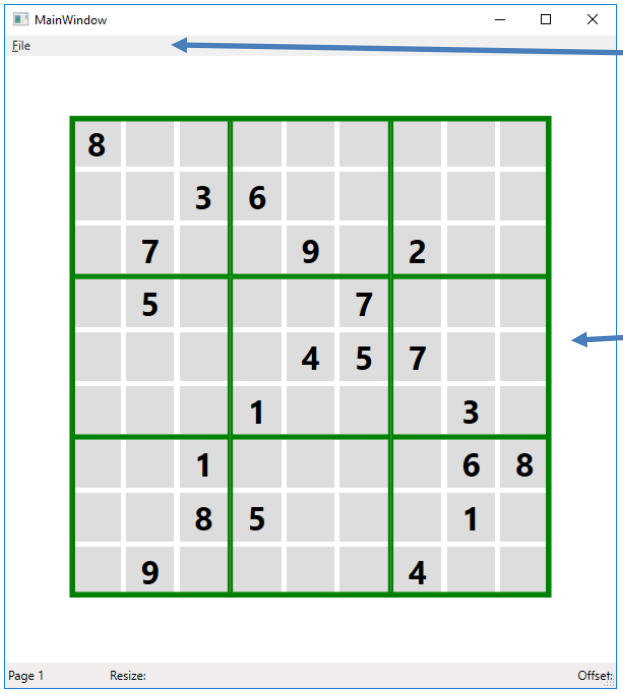
```
s[I, J] = n;
// or
s.SetNumberAt(I, J, n);
```

Before doing this, you must make sure that it does not break anything. The Sudoku must be solvable after you set the number. Otherwise you should flag an error!

**Startup exercise 4:**

You are now ready to create a WPF or WinForms project and start building your Sudoku square. Give your application a private Sudoku and start using this as data structure for your numbers.

My GUI looks something like this (only an illustrative solution):




Maybe a menu for selecting a new puzzle from a puzzle file, saving solutions, ect.

Nice big Sudoku window for playing. Remember to check if the user enters correct numbers.

A StatusBar could give valuable info to the user: Time spent, number of open cells … ect.

Happy coding 😊