Mini-Project: Sudoku Solver

Summary:

The aim of my project was to make a GUI that could solve a Sudoku puzzle. All that you have to do is enter a Sudoku puzzle, in the grid, that has a unique solution and press the “Solve!” button. Press the “DEL” button to clear the whole program at any time.

The method that I used to solve this was backtracking. This is an approach that checks if it is possible to place a number in the first empty square if so then the number is placed, and the method is called recursively. When no new number can be placed, the function goes back and increments a number and starts from there.

A modern computer is capable of solving a Sudoku efficiently despite each extra cell meaning 9 times the possible solution.

Note: this project led me to enjoy sudoku more and due to that I attempt Sudoku problems every now and then.

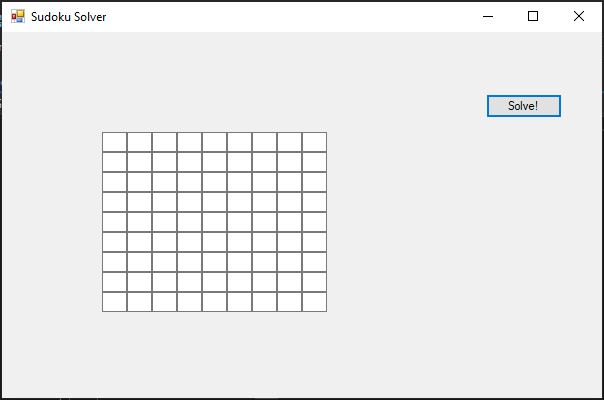
UI Design:

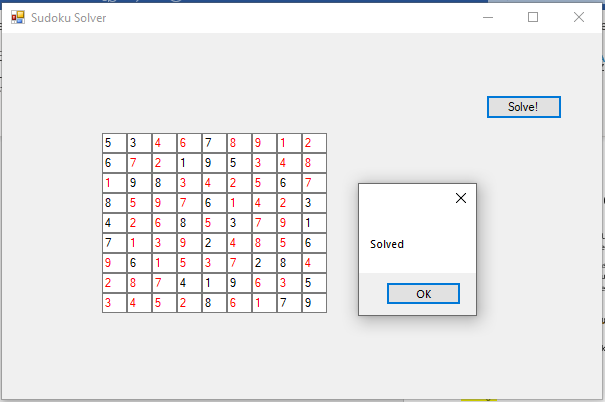
A grid of TextBoxes programmatically created at runtime.

The numbers that were inputted are in black and the solved numbers are in red.

The “Solve!” button that is clicked when you want to solve the Sudoku puzzle.

A MessageBox that appears once the Sudoku puzzle is solved.





Event Handlers:

* Form1\_Load

This Event Handler is run once the form is loaded and starts the game and adds the KeyDown event to solveButton.

* solveButton\_Click1

This Event Handler calls the AddToGrid and Solve methods. This means that all the inputs in the cells are added to the numberGrid and numberGrid2 arrays then the Sudoku puzzle is solved.

* Form1\_KeyDown

If the “DEL” key was pressed down, all the cells in the grid are emptied and so are the arrays the numberGrid and numberGrid2. This clears the whole program

* grid[y, x].KeyDown

This adds the KeyDown event to each TextBox in grid.

* solveButton.KeyDown

This adds the KeyDown event to solveButton.

Code:

Sudoku Solver.cs:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Sudoku\_Solver

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

TextBox[,] grid = new TextBox[9,9];

int[,] numberGrid = new int[9, 9];

int[,] numberGrid2 = new int[9, 9];

/// <summary>

/// Once the form is loaded, the grid is created and the KeyDown event is added to solveButton

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void Form1\_Load(object sender, EventArgs e)

{

CreateGrid();

solveButton.KeyDown += new KeyEventHandler(Form1\_KeyDown);

}

/// <summary>

/// Creates a grid of TextBoxes programmatically

/// </summary>

public void CreateGrid()

{

for(int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

{

grid[i, j] = new TextBox();

grid[i, j].Location = new Point(25 \* i + 100, 20 \* j + 100);

grid[i, j].Height = 50;

grid[i, j].Width = 25;

grid[i, j].KeyDown += new KeyEventHandler(Form1\_KeyDown);

this.Controls.Add(grid[i, j]);

}

}

}

/// <summary>

/// Adds the contents of the grid into the 2D arrays: numberGrid and numberGrid2

/// </summary>

public void AddToGrid()

{

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

{

if(grid[i, j].Text != "")

{

numberGrid[i, j] = Convert.ToInt32(grid[i, j].Text);

numberGrid2[i, j] = Convert.ToInt32(grid[i, j].Text);

}

}

}

}

/// <summary>

/// Checks whether n is a valid move in position [y, x]

/// The method checks the row, the column and the 9 x 9 grid

/// </summary>

/// <param name="y"></param>

/// <param name="x"></param>

/// <param name="n"></param>

/// <returns></returns>

public bool CheckValidPosition(int y, int x, int n)

{

//Check row

for (int i = 0; i < 9; i++)

{

if (numberGrid[y, i] == n)

{

return false;

}

}

//Check column

for (int i = 0; i < 9; i++)

{

if (numberGrid[i, x] == n)

{

return false;

}

}

//Check minigrid

int y0 = (y / 3) \* 3;

int x0 = (x / 3) \* 3;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (numberGrid[y0 + i, x0 + j] == n)

{

return false;

}

}

}

return true;

}

/// <summary>

///

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void solveButton\_Click(object sender, EventArgs e)

{

AddToGrid();

Solve();

}

/// <summary>

/// Uses backtracking to solve the Sudoku in numberGrid

/// If when "Solve!" was clicked there was a 0 inside a cell in numbeGrid2...

/// that means that it was not an inputted number and it should be red.

/// Otherwise, the number should be black.

/// These numbers are then put back into the grid of TextBoxes

/// A message shows that the Sudoku is solved and all the cells are set to black for the next puzzle

/// </summary>

public void Solve()

{

for (int y = 0; y < 9; y++)

{

for (int x = 0; x < 9; x++)

{

if (numberGrid[y, x] == 0)

{

for (int n = 1; n < 10; n++)

{

if (CheckValidPosition(y, x, n))

{

numberGrid[y, x] = n;

Solve();

numberGrid[y, x] = 0;

}

}

return;

}

}

}

for (int y = 0; y < 9; y++)

{

for (int x = 0; x < 9; x++)

{

if(numberGrid2[y, x] == 0)

{

grid[y, x].ForeColor = Color.Red;

}

else

{

grid[y, x].ForeColor = Color.Black;

}

grid[y, x].Text = numberGrid[y, x].ToString();

this.Controls.Add(grid[y, x]);

}

}

MessageBox.Show("Solved");

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

{

grid[i, j].ForeColor = Color.Black;

}

}

return;

}

/// <summary>

/// If the "DEL" key is pressed, the whole program is cleared

/// This Event Handler operates for the form, the cells in the grid and the "Solve!" button

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

if(e.KeyCode == Keys.Delete)

{

for (int i = 0; i < 9; i++)

{

for (int j = 0; j < 9; j++)

{

numberGrid[i, j] = 0;

numberGrid2[i, j] = 0;

grid[i,j].Text = "";

}

}

}

}

}

}

Sudoku Solver.Designer.cs:

namespace Sudoku\_Solver

{

partial class Form1

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.solveButton = new System.Windows.Forms.Button();

this.SuspendLayout();

//

// solveButton

//

this.solveButton.Location = new System.Drawing.Point(484, 62);

this.solveButton.Margin = new System.Windows.Forms.Padding(2);

this.solveButton.Name = "solveButton";

this.solveButton.Size = new System.Drawing.Size(76, 24);

this.solveButton.TabIndex = 0;

this.solveButton.Text = "Solve!";

this.solveButton.UseVisualStyleBackColor = true;

this.solveButton.Click += new System.EventHandler(this.solveButton\_Click);

//

// Form1

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(600, 366);

this.Controls.Add(this.solveButton);

this.Margin = new System.Windows.Forms.Padding(2);

this.Name = "Form1";

this.Text = "Sudoku Solver";

this.Load += new System.EventHandler(this.Form1\_Load);

this.KeyDown += new System.Windows.Forms.KeyEventHandler(this.Form1\_KeyDown);

this.ResumeLayout(false);

}

#endregion

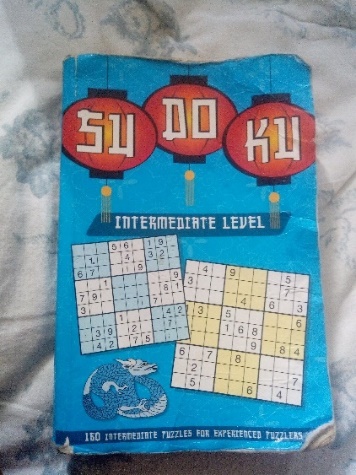
private System.Windows.Forms.Button solveButton;

}

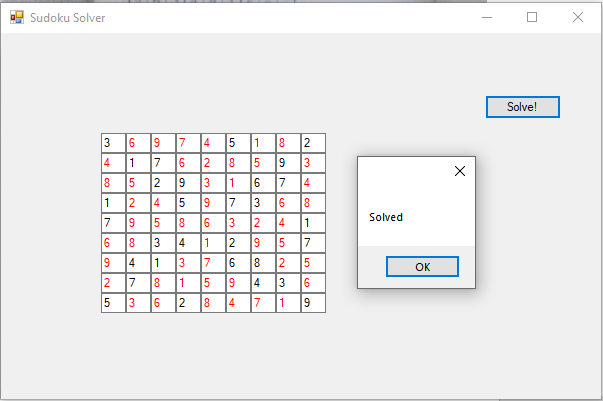
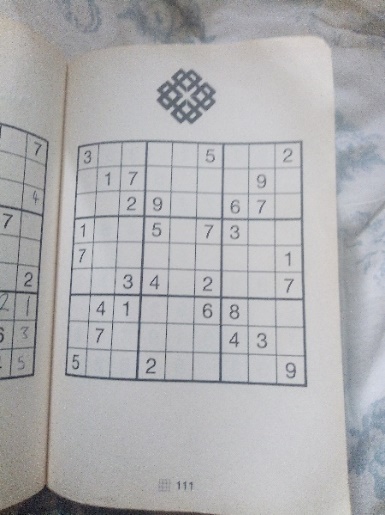
}

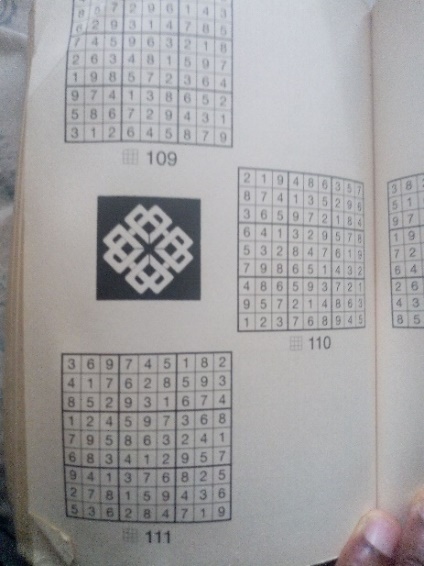
Test Plan:

I have an “intermediate” sudoku book and I used examples from it as the test cases. The book also provides solutions.

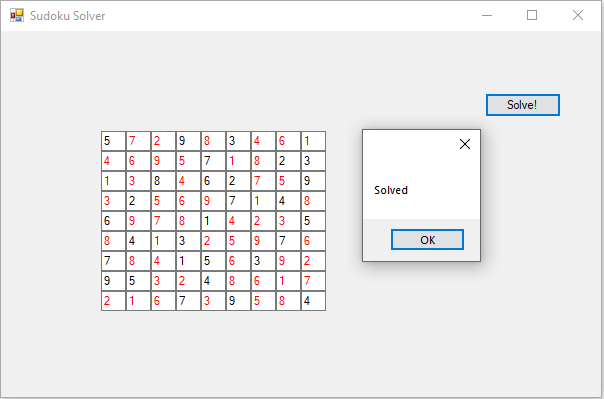
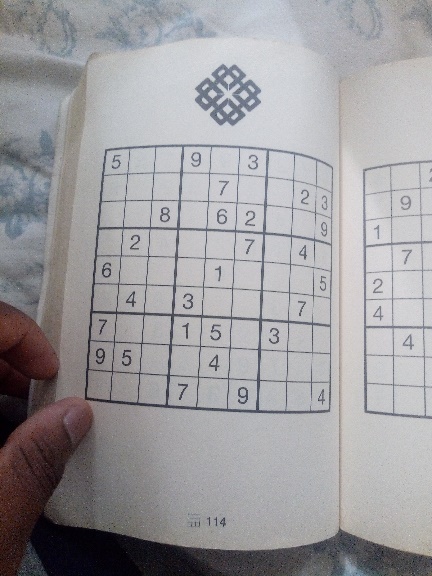
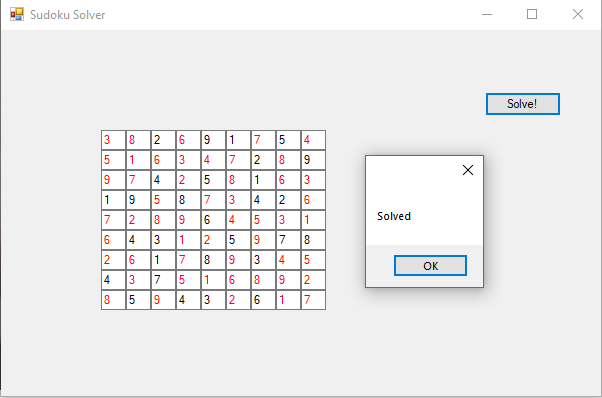
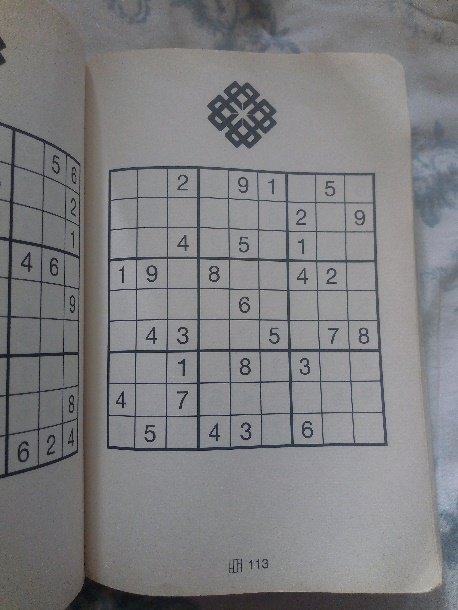
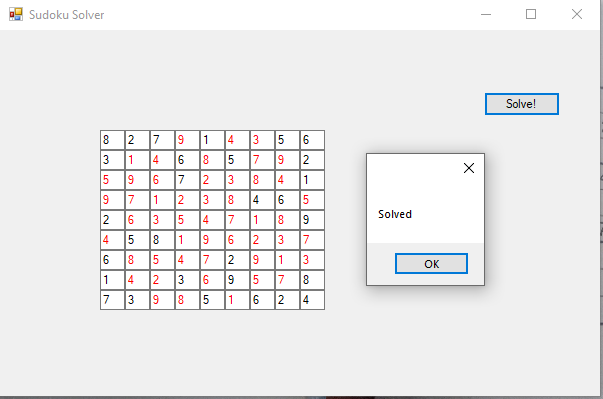
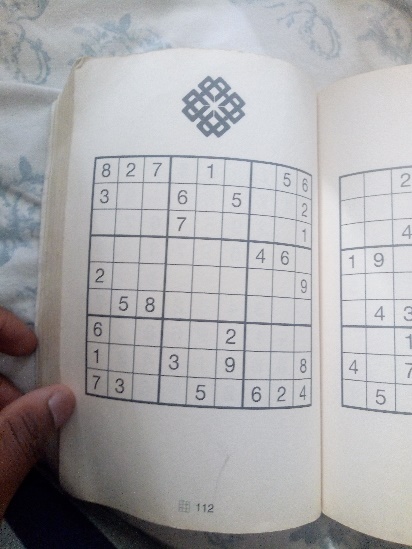


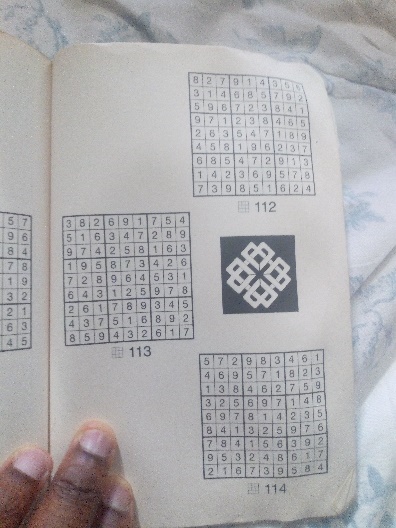
Test case 1:





Test case 2:





Test case 3:

