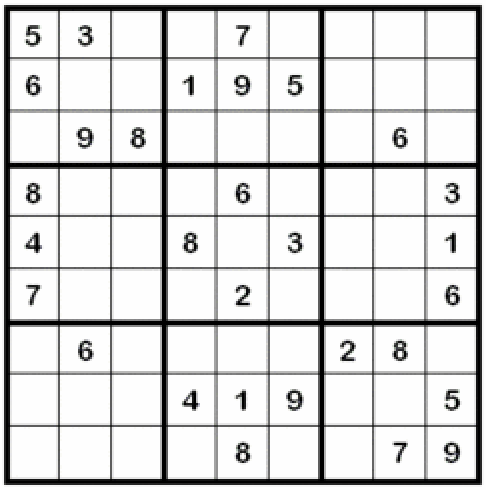
CS 342 – Software Design Project 3 Fall 2017

Project 3 - Sudoku Solver

Due Date: Thursday, October 26, 2017 at 11:59 pm

Group selection deadline: Tuesday, October 17, 2017 at 11:59 pm

Sudoku is a puzzle that often uses a 9x9 grid of 81 cells. The grid is divided into rows, columns and boxes. The boxes are 3x3 sub-grids of 9 cells. Thus each row, column and box contains 9 cells. The object is the fill in the numbers from 1 to 9 so that each row, column and box contain each number from 1 to 9 only once. For more information, check out the Wikipedia entry for Sudoku. Note that the Wikipedia uses the term "sub-grid" instead of "box". Other sources will also use different terms. The following is an example of such a puzzle.



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

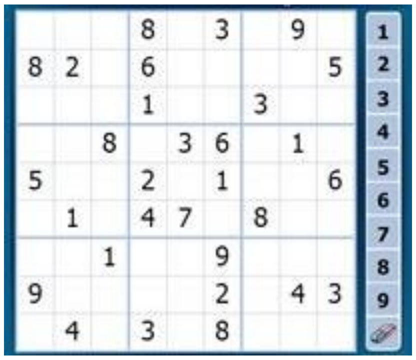
When solving a Sudoku puzzle, the solver attempts to find a situation that will resolve (or help resolve) a value in a cell. For this assignment, you are to write a GUI program in Java which will allow a user to "play" a Sudoku puzzle and that will provide help to the user for the following situations as defined in the Step-by-Step Guide to Solving Sudoku by Angus Johnson.

- Singles
- Hidden Singles (sometimes called Hidden Values)
- Locked Candidates
- Naked Pairs

Your program is not to employ any other algorithms. Since there are many Sudoku solving programs out on the internet, programs submitted that use additional algorithms will be assumed that the authors did not write their own code and thus will receive a grade of 0 for the project.

For this project, all CS undergrad students must work in teams of 2 or 3 students. All members of the each team must be currently taking CS 342. Students will be allowed to select team members themselves. Students in the class who are not CS undergrad students may work on their own or in a group of 2 or 3 students. Groups must be selected and reported in Piazza by Tuesday 10/17/17 before 11:59pm. The reporting in Piazza must be done as a REPLY to the instructor's original post for Program 3 Group Selection.

Our Sudoku program will display the 81 cells of the Sudoku grid with an additional 11 "buttons" to interact with the grid. Consider the following image. The image shows 10 buttons, 9 for digits and 1 with an eraser.



The 9 digit buttons allow the user to place digits into the cells. This is done by first clicking on one of the digit buttons (which will "activate" that digit) and then clicking on one of the cells in the grid. This places the active digit into cell clicked. The user should be able to place the digit into multiple cells without having to re-select the digit button every time. Also there needs to be some manner to inform the user of which digit is currently active.

The eraser button allow the user to clear a cell in the grid. Using the text of "X" or "C" is a valid replacement for an image of an eraser. The user should only be able to clear cells that were not part of the puzzle when loaded. The user should be able to clear multiple cells without having to re-select the clear button every time. Also there needs to be some manner to inform the user that the puzzle is on "clear mode".

Our puzzle will need one more button. This will be a help button (often a question mark is used to identify this button). When this button is clicked and then a cell is clicked, the program is to display information about the "candidate list" for that cell. One idea is to create a "status bar" at the bottom of the JFrame window (using a JLabel in the BorderLayout.SOUTH position). The user should be able to view the candidate list from multiple cells without having to re-select the clear button every time (only one candidate list is expected to be displayed at any time). Also there needs to be some manner to inform the user that the puzzle is in "candidate list display mode".

The general algorithm that most solvers use (and that will be expected for this program) is to begin with a list of all possible values (i.e. 1 through 9) in each cell of the grid. This list is often called the **candidate list**. As each cell is resolved (i.e. the digit is placed into that cell), remove that value from the candidate lists for the cells that share a row, column or box with the cell that is resolved.

JMenu Items Required for the JFrame

Your program is to have the following JMenu items. The JMenu needs to be organized into sub-menus or ease of use.

Under a File Menu, your program is to allow the user to:

- Load a Puzzle from a File – This is to allow the user to search the file system for a puzzle stored in a text file. Your program is expected to use the JFileChooser class for this. The format for the puzzle is given below.

- Store a Puzzle into a File – This is to allow the user to store the current puzzle into a text file named by the user. The format for the puzzle is give below. Your program is expected to use the JFileChooser class.

- Exit the program

Under a Help Menu, your program is to allow the user to:

- Display a dialog box on how to play Sudoku.

- Display a dialog box on how to use your interface.

- Display an About box listing the authors of the program. Include name and Net-IDs of all team members.

Under a Hints Menu, your program is to allow the user to:

- Toggle a "Check on Fill" mode – When this mode is on, if the user attempts to place a digit into a cell that is NOT in the cells current candidate list, an error message will be given and the digit is not placed into the cell. This error message could be given in a status window. The use of a JCheckBoxMenuItem is suggested for this.

- Fill in a single blank cell using the **Single Algorithm** – Your program will need to search through the blank cells and see if the Single Algorithm can be applied to that cell to resolve it. Once one cell has been resolved, stop processing.

- Fill in a single blank cell using the **Hidden Single Algorithm** – Your program will need to search through the blank cells and see if the Hidden Single Algorithm can be applied to that cell to resolve it. Once one cell has been resolved, stop processing.

- Fill in a single blank cell using the **Locked Candidate Algorithm** – Your program will need to search through the blank cells and see if the Locked Candicate Algorithm can be applied to that cell to resolve it. Once one cell has been resolved, stop processing

- Fill in a single blank cell using the **Naked Pairs Algorithm** – Your program will need to search through the blank cells and see if the Naked Pairs Algorithm can be applied to that cell to resolve it. Once one cell has been resolved, stop processing

- Fill in a many blank cells as possible using all four algorithms – Your program is to repeatedly process all of the blank cells until none of the algorithms can resolve any blank cell. Note that once a single blank cell has been resolved, your program should restart this process with the first blank cell.

When all 81 cells have been filled in, display a message congratulating the user on solving the puzzle.

The four situations

- Single

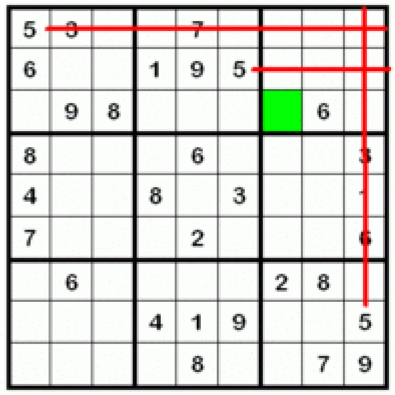
  A single is when a cell, X, only has one possible value, V, because the other values are already resolved in a cell that share a row, column or box with X. In the puzzle at the top of the page, the cell in the vary center of the puzzle (row 5, column 5) must have the value of 5 since the values 1,2,3,4,6,7,8 and 9 are already known in cells that share the row, column or box with the cell at row 5, column 5.

  - There is 1 at row 5, column 9 and row 8, column 5.
  - There is 2 at row 6, column 5.
  - There is 3 at row 5, column 6.
  - There is 4 at row 5, column 1.
  - There is 6 at row 4, column 5.
  - There is 7 at row 1, column 5.
  - There is 8 at row 5, column 4 and row 9, column 5.
  - There is 9 at row 2, column 5.

  This situation is easy to recognize using candidate lists. When an unresolved cell only has a single value in its candidate list, you have a single.

- Hidden Single (or Hidden Value)

  A hidden single is when a cell, X, must have the value V because no other cell in a row, column or box could have that value. Consider the following puzzle.



  The green cell at row 3, column 5 must contain the value of 5 because no other cell in the upper right box could contain the value of 5. The top row of the box is prevented from having a 5 because of the 5 at row 1, column 1. The second row of the box is prevented from having a 5 because of the 5 at row 2, column 6. The right column of the box is prevented from having a 5 because of the 5 at row 8, column 9. The box can't have a 5 at row 3, column 8 since it is known to have a 6.

  This situation is recognized by checking the candidate lists for all cells in a group (a group can be a row, a column or a box). If a value only appears in only one candidate list for a group, that value must be the value for the cell that corresponds to this candidate list.

- Locked Candidate

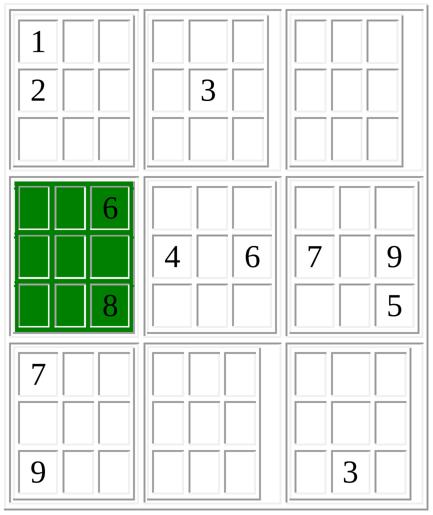
  This situation will not resolve the value in a cell but will reduce the number of possible candidate values for a cell.

  This situation involves two groups (where a group is a row, column or box) that have three intersecting cells. The possible groups could be:

  - a box and a row
  - a box and a column
  - a row and a box
  - a column and a box

  When it is determined that a value must be contained in the cells in the first group and these cells are also part of a second group, this value can be removed from the candidate lists for the other cells in the second group.

  Consider the following example. The value of 3 in the fifth row must occur in the first 3 cells (the middle sub-row in the green box). Thus the value of 3 can be removed from the candidate lists of all the cells in the top sub-row and the bottom sub-row of the green box. Once this has been done, the cell at position (6,1) should only have the value of 4 in its candidate list and can be resolved. After that is resolved, the cell at position (4,1) should only have the value of 5 in its candidate list and can be resolved. The data for this puzzle is available in proj1data5.txt.



- Naked Pairs

  This situation will not resolve the value in a cell but will reduce the number of possible candidate values for a cell.

When a group (row, column or box) has two cells that have the same two value candidate list those two values must exist in those two cells. Those two values can be removed from any other candidate lists in the group.

Consider the following example. The candidate lists for the two green cells are both (4,6). The candidate list for the red cell is (2,4,6,8). The candidate list for the blue cell is (4,6,8). The green cells have a naked pairs situation. Since the green cells have the values of 4 or 6 we can remove those values from the candidate list for all cells in the same group (the fifth row in this case). By removing the value of 4 and 6 from the blue cell, its candidate list will only have the single value of 8 and can be resolved. Now, the red cell will have had the values of 4 and 6 removed by the naked pairs and the value of 8 removed by the single value, so it will have only a single value of 2 left in its candidate list and can be resolved. The data for this puzzle is available in proj1data4.txt.

			2					
			8					
1	🟥	3	🟩	5	🟩	7	🟦	9
					2			
					8			
							2	

Description of Input and Output to the Text Files

The input for your program will be an ASCII text file that will have 3 values per line:

1. a row position (listed first)
2. a column position (listed second)
3. a value (listed last)

This information will give the beginning layout of the puzzle. The filename is to be given to the program using a command line argument. If any of the row, columns or values is outside of the range from 1 to 9, print an appropriate error message and ignore that line of input for the puzzle. If the input attempts to place a value in a cell that is invalid (the value is not one of the values contained in the cell's candidate list), print an appropriate error message and ignore that line of input for the puzzle. Below are some example data files:

- proj1data1.txt - A Gentle Puzzle
- proj1data2.txt - A Gentle Puzzle
- proj1data3.txt - A Moderate Puzzle
- proj1data4.txt - An Unsolvable Puzzle using Naked Pairs

- proj1data5.txt - An Unsolvable Puzzle using Locked Candidates

**** NOTE: THE PARAGRAPH THAT WAS IN THIS LOCTION WAS TO HAVE BEEN REMOVED FROM THE ORIGINAL WRITE_UP FOR THE PROGRAM***************

When saving the puzzle information to a file, the same format must be used:

- 3 integer values per line for every filled cell in the puzzle
- the row position of the cell listed first
- the column position of the cell listed second
- the value of the cell listed third

Submission of the Program

You are to submit your program via GitHub following the directions provided by the TA. Thus you are required to make sure your program is accessible via the GitHub interface in Eclipse.