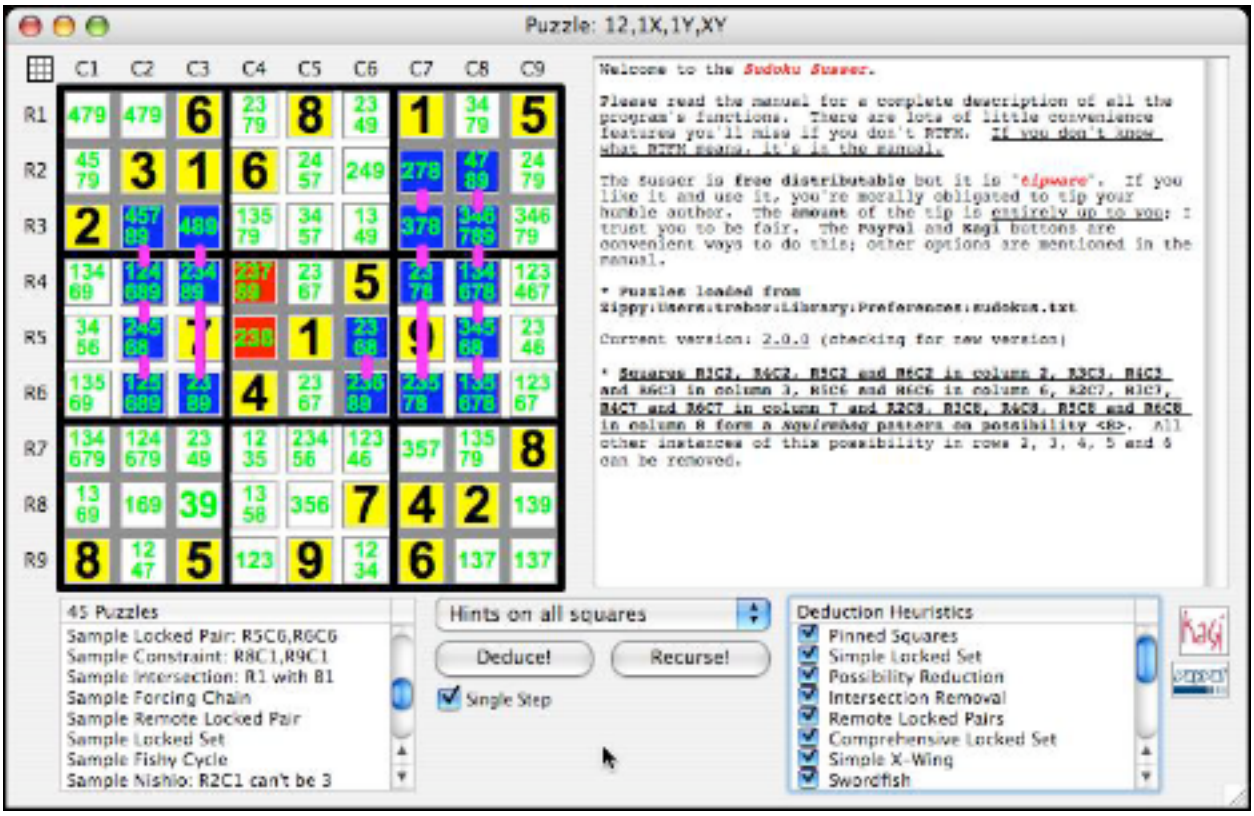
The Sudoku Susser



The Sudoku Susser may be freely redistributed but not sold or included as part of a package or collection without prior written permission.

The latest version of the program and documentation can always be found at:

http://www.madoverlord.com/projects/sudoku.t

Note: If you replace your current sudokus.txt file with the new version in the distribution, you'll lose any changes you've made to that file; it stores your prefs and sudokus.

Note: there is a bug in keyboard menu handling in the current version of RealBasic, at least on the Macintosh. If you try and type a cmd-key while one of the yellow "help tags" is visible, it won't be processed. Please be aware of this glitch for now.

The Point of the Program

The Sudoku Susser helps you solve sudoku puzzles. A sudoku puzzle grid has 9 rows, 9 columns, and 9 boxes of 3x3 squares each.

To solve the puzzle, you must fill in the grid so that every row, every column, and every 3x3 box contains the digits 1 through 9. A properly constructed puzzle has only one solution.

For a more detailed introduction to sudoku puzzles, I recommend visiting the Daily Mail Sudoku pages:

http://www.dailymail.co.uk/sudoku

This documentation assumes a basic familiarity with sudokus.

An Important Note for Windows Users

I am a "Mac Guy," and Sudoku Susser was written on a Macintosh in RealBasic. I've coughed up $300 so that I can cross-compile Susser for Windows, but since the only Windows box I have is an old Win98 machine, I can't extensively test it.

This documentation is mostly Mac-oriented. When you see references to the **command** (sometimes **cmd**) key, the equivalent key on Windows is **control** (aka **CTRL**); in other words, the menu key. Also, the Macintosh **Option** key is the Windows **Alt** key.

See the Unanswered Windows Questions section at the end of this document for a list of things that I can't currently test on Windows -- perhaps you can help me out.

I am also able to produce a version that runs on Linux, which I cannot test at all. If you would like to play with the Linux Susser, drop me an email and I'll send it to you.

Installing the Program

Just create a folder wherever you keep your applications (ie: on the Mac, in the Applications folder) and drag all the included files into it. One of these files is the ***sudokus.txt*** file. It contains all your puzzles and preferences.

If you're the only one using your computer, then the sudokus.txt file located in the same folder as the application will be used. But if your computer is used by multiple sudokuholics, each can have their own sudokus.txt file, if it's placed in the right place:

Mac OS X : /Users/{username}/Library/Preferences folder

Mac Classic: {System Folder}:Preferences folder

Windows 98: Windows directory

Windows: Settings\{username}\Application Data or c:\Documents

In other words, the app looks in these places first, then looks in the same folder as the app. If you look in the log panel after the program launches, it'll tell you where it found the file.

Confused? Blame John Anderson, the guy who begged for this feature.

Important: If you download a newer version and then replace your current sudokus.txt file with the new copy, you'll lose any changes you've made to that file; it stores your prefs and sudokus.

The previous version of your sudokus.txt file is saved in ***oldsudokus.txt***, so you can usually (mostly) recover from this calamity.

Note that if you stick sudokus.txt in one of the places listed above, then when you update the program, even if you copy over the sudokus.txt file in the same folder as the Susser application, you won't trash your current sudokus.txt file, since it's safely stored elsewhere!

The Sudoku Susser Window

The large grid in the top-left of the window contains your current puzzle. The original solved squares in the puzzle are shaded yellow. Squares you have solved will contain a single digit with a white background. Depending on what hinting modes you have selected, the remaining possibile values for a square may appear in green. You will also be able to select a variety of hilight modes to help you identify puzzle features; more on them later.

Below the puzzle is your puzzles list; this contains all the puzzles you currently have stored in the ***sudokus.txt*** file. To the right of the puzzle is the log panel. As events occur, they are recorded in the log.

Below the log panel (and to the right of puzzles list) are some buttons and a series of checkboxes; these let you decide what techniques the program will use when you ask it to deduce some moves. You can also choose whether you want a complete solution, or just the next step; you can also ask for a "brute force" recursive solution to the puzzle.

Using the Puzzles List

The puzzles list shows you the current list of puzzles that Sudoku Susser knows about.

Things you can do with the puzzles list:

Load a puzzle	Double-click (fast!), or click once to select, then press RETURN or ENTER.
Delete a puzzle	Click once to select, then press DELETE.
Edit puzzle name	Click once to select, then click again to edit.
Reorder puzzles	Click on an unselected puzzle and drag it. You can also drag outside the app; see the section on dragging.
Sort puzzles	Click on the "number of puzzles" header.

File Menu Options:

File» Open... lets you load a text or graphic puzzle. The types of graphic puzzles that can be loaded will depend on your system configuration. Macs seem to be much better than Windows and Linux machines in this regard.

File » Save Puzzles and Preferences will save your current puzzle, the puzzles in the puzzle list and your current preference settings to the Sudokus.txt file. The previous version of the sudokus.txt will be stored in the same folder as oldsudokus.txt.

File» Save Current Puzzle as Text… lets you save your current puzzle as a text file.

File» Save Current Puzzle as Graphic… lets you save your current puzzle as a PICT (Mac) or BMP (Windows/Linux) graphic.

File » Update Current Puzzle in Puzzle List will replace the puzzle in the puzzle list with the same name as the current puzzle with the contents of the current puzzle.

File » Add Current Puzzle to Puzzle List will add your current puzzle to the puzzles list, choosing a new unique name for it. You can edit this name to taste.

File » Clear Current Puzzle will blank out the current puzzle board (it will not delete a puzzle from the puzzles list). Since this is equivalent to the New option you find in most File menus, it has Cmd-N as a shortcut.

File » Reset Current Puzzle will erase the changes made since you started playing with the puzzle (leaving the yellow squares unchanged)

About the "missing" Preferences menu

The preferences menu item is always disabled, because the program doesn't need one. It does a have a bunch of things you can set in the various menus and options in the window, and your current settings are saved every time you quit, which amounts to the same thing.

Entering Puzzles (the hard way)

You can enter a puzzle by hand by doing the following:

- **File** » **Clear Current Puzzle** to make the puzzle totally blank
- Use the mouse or the keyboard to move the red **current square** hilight to the square you want to change, and type a number to set it, or space to blank it out. The arrow keys, home, end and tab all work as you'd expect.
- Alternately, you can click on the square and select from a popup menu.

When the puzzle is complete:

- Select **Options** » **Set Initial Squares** to color the starting squares in yellow.
- Select **File** » **Add Current Puzzle to Puzzle List** to add the puzzle to your puzzles list.

Entering Puzzles (the easy way)

Just drag them into the program! You can drag puzzles in from webpages or files. Sudoku Susser can read graphics, text clippings, even some PDF documents. It has a built-in character recognizer that almost always gets the puzzle right (if you find a puzzle it can't load, let me know so I can improve the program!)

When dragging text puzzles into Sudoku Susser, it's best to select just the puzzle and drag that (or copy & paste, that works too). The one exception is puzzles from this site:

http://www.menneske.no/sudoku/eng/

At this site, just select the entire page and drag it. Sudoku Susser has special code to recognize this action.

Text Puzzle Tips

Selecting text puzzles on web-pages works better if you start selecting just after the bottom-right character in the puzzle and extend the selection to the line just above the start of the puzzle. This ensures that if there are any hidden leading spaces in the selection (often the case on web-pages), each row in the puzzle has the same number of them, and the text recognizer can compensate and line things up. The Susser is pretty good about throwing away trash before and after the puzzle, so it's always better to select too much than too little -- as long as you don't select extra text with digits in it...

Context is everything

It's always better to drag puzzles from web-pages into the program, as opposed to saving them onto your hard disk and dragging the files. When you drag a graphic from the browser, Sudoku Susser also gets the URL, and this provides extra information that it can use to extract the puzzle and strip off the ornaments and text surrounding it.

When all else fails, try saving the puzzle as a file, then opening it in your favorite graphic application and use the marquee tool to grab just the puzzle, and paste it into Sudoku Susser. This also works with PDF documents in Adobe Reader. Sometimes you can drag the selection, sometimes you need to copy & paste.

Consider SnapNDrag

An excellent app for grabbing a puzzle graphic off any part of your screen is a freeware app called SnapNDrag. It's like Apple's Grab app, only much easier to use. You just click the selection button, select an area of the screen using a big and easy to use marquee, then drag the resulting thumbnail from SnapNDrag into Susser. Highly recommended. Get it at

http://www.yellowmug.com/snapndrag/

Secret Scanning Features

Hold down **Option** (**Alt**) when dragging in a graphic puzzle to disable the "find the puzzle in the middle of the graphic" feature. This lets you drag in borderless puzzles as long as you've clipped them right to the edges of the puzzle.

Similarly, holding down the **Shift** key will disable the "make sure the puzzle is valid and has only one solution" sanity checks.

If you want to cancel an import, hold down **cmd-.** [**period**], the standard "stop it already" key, until the Susser notices and stops.

If the puzzle doesnt load correctly…

Select **Options** » **Clear Initial Squares** to unhilight the squares, then edit them (see "the hard way", above), then select **Options** » **Set Initial Squares** to hilight them again.

Entering Puzzles (the really easy way)

Select **Options** » **Fetch Puzzle** » **(Type of Puzzle)** to fetch a puzzle directly from the ***www.menneske.no*** sudoku archives. You can choose from various types and difficulties. **Options** » **Fetch Puzzle** » **Repeat Last Fetch** (**cmd-F**) will let you get puzzle after puzzle of the same type.

Note: Sometimes when you try and fetch a puzzle from the site, it will return a puzzle of a slightly different difficulty; this is an issue with the archives program on menneske.no, not the Susser.

Saving and Sharing Puzzles is a Drag

Whenever you quit the program, you will be given the option to save your puzzles if you've made any changes to them. You can also save the puzzles list at any time using the **File » Save Puzzles** option. Saving your puzzles will also save any changes you've made to the program's settings, window position, size, and so on.

In addition, you can usually **drag** the current sudoku from the application into any other application, in either graphic or text form. Where you can drag will depend on your platform and the apps you use. Simply drag the tiny sudoku grid widget in the top left corner of the window into the other application, and drop it where you want it. Most email programs and word processors will accept drags from the Susser; on the Mac, you can drag puzzles just about anywhere, including the Finder!

- A normal drag will drag a small text version of the puzzle.
- For a larger text puzzle, hold down option and drag.
- If you want a graphic puzzle, hold down ctrl and drag.

  Large text and graphic puzzles will have hints in them if you have chosen to show hints.
- You can also hold down both ctrl and option to drag a text version in a format suitable for import into Andrew Gregory's Sudoku for the Palm application.

You can also do all these drag operations directly from the Puzzles list!

Solving Sudoku Puzzles

The current puzzle is displayed as follows:

Yellow squares are the initial squares of the puzzle.

White squares with a single number in black are the squares you have solved.

If you have hints turned on, then the remaining possibilities for each unsolved square will appear in it in green.

Note that if you see a square with a single green number in it, that's a square that has reduced to a single possibility but hasn't yet been formally declared to be solved.

The **current square** is surrounded by a red hilight. You can change the current square by moving the mouse into another square, or by using the arrow keys. Also, **Tab**/**Shift-Tab** move you to the next/previous square, wrapping to the next/previous row, and **Home** and **End** do what you expect.

Note: if you're editing the name of a puzzle in the Puzzles list, the arrow keys will move the cursor around in the text. Click outside the Puzzles list to release the "imprisoned" arrow keys.

All you have to do to solve the puzzle is figure out what number goes in each square, make it the current square, and type the number.

If a move you make results in a square having no possible values (in other words, you screwed up!), you'll be rudely informed of this fact by the appearance of a large black **X** on a red background.

If you don't like this, uncheck **Options » Hints » Emphasize Invalid Squares** *to make invalid squares display as blank.*

If you make a mistake:

- You can blank out a square by typing space or cmd-space.
- Undo and Redo are fully functional, so you can backtrack.

As when setting the puzzle, you can also click on a square to pop up a menu of choices.

Basic Hints

If you select **Hints on all squares** from the pulldown menu in the window under the puzzle (also **Options** » **Hints** » **Hints on all squares**), each of the empty squares will display the remaining possibilities for that square. Knowing what possibilities are left is the key to solving puzzles, so most people play with hints on. However, you can also choose to display hints just when you hilight squares (see below), or be a stud (or more likely, a masochist) and play with no hints.

If you have hints on, then the popup menus on each square will list the remaining possibilities first, followed by other options, including the ability to remove a possibility from the square.

You can also remove possibilities by typing **shift + the number** you want to remove.

Note: different keyboard layouts have different shift+number layouts, and it is impossible to support them all. Currently, I am supporting the US keyboard layout and [I hope, let me know if it's broken] the UK layout.

An alternate way to remove possibilities is to type the **minus (-)** key, then type the number. The mouse pointer will change to a pointing finger when you type the minus to give you a clue that its expecting another key. Macintosh users can also type **cmd + the number** to remove possibilities.

Select **Solve» Check puzzle validity** (or simply press **V**) to have the program check that the puzzle is in a consistent state, that starting from the intial squares, it has a single solution, that your current partial solution is correct, and whether the possibilities listed in each unsolved square are correct. If a problem is found, you'll be given a clue as to how to proceed.

This feature is very handy if you're starting from scratch and filling in the possibilities yourself. See Masochismo for more details on how to do that.

When you make changes to the puzzle, either solving a square or reducing its possibilities, possibilities for other squares in the puzzle will be updated.

Note: If you blank out a square (space or cmd-space), the possibilities for every square in the puzzle will be recomputed to maintain a consistent puzzle. This may mean that possibilities you've removed from other squares will reappear. This doesn't happen when you solve a square; in such cases the puzzle can be updated without undoing possibility removals.

Hilights aren't just for hair anymore

Sudoku Susser provides a variety of hilighting options that make it easier for you to recognize patterns in the puzzle.

Pressing the **H** key and then a number key will hilight in blue all the squares in the puzzle that have been set to that number, as well as all of the squares that can still possibly be that number. **Hilight » Squares that can be...** does the same thing.

Mac users can also tap the **option** key while the mouse (or hilight cursor) is over a solved square to hilight using that square's number.

This is very handy for finding pinned squares; if a group has only one hilighted square, it's pinned.

You can turn off hilighting by pressing **ESC**.

Pressing **=**, then a number, will hilight all the instances of that number (both solved squares and possibilities) in pink (thus the option is found in the menus under **Hilight » Pink Possibilities...**

Pressing **B**, then a number, will hilight all the squares in the same row, column or block as the current square that can still possibly be that number (the "buddies"). **Hilight » Buddies that can be...** does this also, but only if you are using the keyboard to activate the menu, since it depends on the mouse position. Mac users can also type **option + number**.

To turn off hilighting, press **Esc**. **Hilights » Clear Hilights** does the same thing.

There are many other hilighting options in the **Hilight** menu, and they all have single-key equivalents that you can just type at any time (no need to hold down the cmd key). Each will give you different information about the puzzle.

Important: please see the Deduction Methods section for a complete explanation of the terms and methods mentioned below!

Pinned Squares (**P**) will hilight all the squares in the puzzle that are pinned to a single value.

You can either figure out for yourself what the value must be, or, if you have **Options** » **Help Tags on Squares** checked, you can mouse over the hilighted square and get the inside scoop. This works for all of the hilighting features where it's helpful.

You can also use **cmd-?**, or just plain **?**, when mousing over a square, to print the contents of its help tag into the log. This is probably most helpful to Windows users, because Help tags seem to be flaky on that platform.

Locked Sets (**L** or **S**) will hilight all the locked sets in the puzzle.

Locked Sets will display all the locked sets that are useful (because they help you progress), as well as all the locked pairs (because even if they don't help you directly, they are used in more complex techniques. Each set is hilighted in a different color; if a square belongs to more than one locked set, it will may have multiple colors.

Even cooler, if you check **Hilight** » **Use Line Connectors**, instead of hilighting the squares, Sudoku Susser will draw nice vectors between them! This sometimes makes it much easier to see the relationships.

Note: Line Connectors don't work right on Linux; they display behind the squares. This is a bug in RealBasic and I have no workaround.

Reductions (**R**) shows you combinations of squares that have extra possibilities that can be removed. Performing these reductions often creates locked sets.

Intersections (**I**) looks at the 3-square intersections of rows and columns with blocks in order to find ways to progress.

Conjugate Pairs (**C**) is an advanced hilighting feature, useful mostly for finding features like **Fishy Cycles**. A group has a conjugate pair in <1> if it has exactly two and only two squares that can be a 1 (and are not yet solved). Puzzles often have a lot of conjugate pairs.

Forcing Chains (**O**) will display the shortest forcing chain in the puzzle. It will display additional chains if they do not share squares with earlier chains (this is to prevent the display from getting cluttered). The first square in each chain is hilighted in blue.

Fishy Cycles (**F**) will display the first fishy cycle in the puzzle that permits progress to a solution. The vertex groups are colored in, and the linking squares that link them are connected by line connectors. Squares that can be modified are hilighted in red. It will also display the sub-types of Fishy Cycles, such as Simple X-Wings, Swordfish, Jellyfish and Squirmbag patterns.

Similarly, **XY-Wings** (**Y**) and **XYZ-Wings** (**Z**) display those patterns in a similar manner.

Do Nishio Cycle (**N**). Please read the explanation of Nishio in the Deduction methods section to get an idea of how this works. To do a Nishio reduction on a particular possibility, tap **option** over a solved square of the same number or use **Hilight** » **Squares that can be...** to hilight all the squares that can be that number. Then mouse over the square you wish to Nishio (it'll be hilighted, since it contains that possibility) and press **N**.

The square's hilight will change to a green circle, and the hilights on any intersecting squares will be removed.

Now repeat the process on another hilighted square, mousing over it and pressing **N**. You can only continue to Nishio on squares that are the only remaining square that can contain the possibility in a group; these are hilighted in solid blue. Some squares will not be eligible for Nishio because their groups all have more than one square that can contain the possibility; these will be marked with a blue slash.

When you nishio a square, its hilight will change to a blue circle, and the hilights on the intersecting squares will be removed. This may also cause blue-slashed squares to go solid blue. Continue to do this until either you run out of squares to Nishio (the remaining squares will either be solved, circles or slashed) or you generate a contradiction (a row, column or block will have no squares that can be the possibility; it will be hilighted in red).

If you find a contradiction, you can remove the original possibility from the original square.

Trebor's Tables (**T** and **A**)

To understand tabling, please refer to the deduction methods section of the manual. There are also several options that control what tabling does that can be changed in the **Solve** » **Table Settings** menu.

Pressing **shift-A** will generate and expand the assertion tables for the entire puzzle, looping until they are fully expanded.

Pressing **a** (no shift key) will do one loop of generation and expanding and will display the current assertion tables in the log.

Pressing **shift-T** will do one loop of generating and expanding of assertions for the current square.

Pressing **t** (no shift key) will do one loop of generating and expanding of assertions for the current square, and display the assertion tables for the square in the log.

Pressing **x** or **X** will clear all the assertions, allowing you to start over.

Assertions are automatically cleared if you do a deduction or make a change to the puzzle.

Puzzle Validity (**V**) will check the puzzle for various problems and give you hints as to how to solve them.

Get a Hint (**g**) or **Get a Big Hint** (**G**) will give you a hint about what techinque you need to apply next, and where to look.

You can also hold down cmd or shift+cmd and click the Deduce! button to get hints. And they're available as options in the Solve menu, not to mention cmd-G and cmd-shift-G.

Only the rules that are currently selected are checked when the program tries to give you a hint.

Masochismo

When a puzzle is loaded, the Susser automatically computes the possibilities for the unsolved squares. For most people, this is the initial step in solving a puzzle, and because it's a mechanical step that requires no thinking, I don't think it's cheating to have the program do it.

Some of my faithful users beg to differ with me in this regard, and have basically said "I'm not going to send you any money until you let me torture myself figuring out the initial state of the puzzle."

Thus, while I was tempted to be truly sadistic and not add any such features to the program, my kinder nature prevailed. If you really must start from scratch, do this:

- Turn hints off.
- Load a puzzle.
- Select either **Options» Set Unsolved Squares to have all possibilities** or **Options» Set Unsolved Squares to have no possibilities**, depending on whether you like to set the initial possibilities by setting stuff or removing stuff.
- Turn hints on again.
- Set or remove possibilities in the normal way (ie: shift-# when over a square).
- When you think you've got the puzzle to a correct state, then use **Solve» Check puzzle validity** to check your efforts before proceeding.

Note: if you don't have all the possibilities for the squares set correctly, the deductive solver will almost certainly eventually get to an invalid puzzle state. However, it seems to be pretty good about solving puzzles if you have extra possibilities in the squares.

Summary of keyboard commands:

Key	Action
# (1-9)	Set current square (the one the mouse is over) to number.
Shift + #	Remove possibility from current square; also "-", then #.
V	Checks puzzle for validity.
H, then #	Hilights squares that can contain #.
=, then #	Hilights instances of # in the puzzle (solved and possibilities) in pink.
B, then #	Hilights squares that can contain # in the row/col/block of current square.
?	Display help tag for current square in log.
P	Display pinned squares.
L or S	Display locked sets.
R	Display reductions.
I	Display intersections.
C	Display conjugate pairs.
O	Display forcing chains.
F	Display Fishy Cycles; looks for and displays simple X-Wing, Swordfish, Jellyfish and Squirmbag patterns before checking for more general Fishy Cycles.
Y,Z	Display X**Y**-Wing and XY**Z**-Wing patterns.
N	Do Nishio cycle.
T	Do one loop of tabling for the current square without displaying details.
t	Do one loop of tabling for the current square, displaying details.
A	Do multiple loops of tabling for the entire puzzle, without details.
a	Do one loop of tabling for the entire puzzle, displaying details.
X	Clear all the tables.
g / G	Get a hint (hold down shift for a big hint)
ESC	Remove current hilighting
Arrows	Moves current square around puzzle.
Home/End	Moves current square to top-left or bottom-right corner
Tab	Moves to next square, wraps down to next row if need be. **Shift-Tab** goes backwards.
	Note: except when specifically noted (T and A), an unshifted letter performs the same function as the shifted one

If You're Stuck, or Lazy

If you want help with a puzzle, the **Deduce**! button will apply 10 increasingly difficult techniques in order to progress. There are currently no known sudoku puzzles that the Susser cannot solve by logical deduction methods that a human could perform (assuming they had a lot of patience and a box of pencils)! Each step in the solving process is explained in detail.

If you want Sudoku Susser to just perform the next step in the solution, check the **Single Step** checkbox located just below the Deduce! button. You can also select which rules it will apply by checking and unchecking the rules checkboxes in the list in the lower-right section of the window.

Hint: Shift or Option-click on a checkbox to select it and deselect all the others; Cmd or Control-click on a checkbox, or double-click any rule name, to select all the rules.

If you just want to see the answer, the **Recurse**! button will brute-force solve the puzzle.

These features are also available as menu items in the Solve menu, as is **Make All Forced Moves**, which sets all squares that have only one possible value for you.

If you just want a hint, try **Solve » Get a Hint** to learn what rule you need to apply next, and **Solve » Get a Big Hint** to learn where you ought to be looking.

Experts may want to use **Solve » Concise Output** to make the deduction engine's messages shorter (particularly when it tries to Nishio and Bowman Bingo)

There are also several options in **Solve » Table Settings** that change the way Tabling works. In general, you should have **Exhaustive Table Generation** and **Negative Assertion Expansion** off, and **Aggressive Forces and Pins** and **Propagate Contradictions** on. See the explanation of the Tables algorithm for more details.

Printing Puzzles

You can print your current puzzle by using the normal **Page Setup...** and **Print** commands in the **File** menu. The puzzle will be printed nice and large with a caption on the page. If you're currently displaying hints, the printout will have the hints in small letters in the blank squares. The **File » Half-Sized Puzzles** option lets you print the puzzle in a smaller size so you have more space for notes.

Miscellaneous Options

Undo and **Redo** work as you might expect them to. You can also select, cut, copy and clear text out of the log panel.

You can change the font and size of the text in the status area using the **Font** menu.

Options » XY Cells (Top-left origin), **XY Cells (Bottom-left origin)**, and **RC Cells** change the display of the cell labels and log messages. Some people like Row&Column format, others like a XY grid format.

Options » Help Tags lets you enable and disable the popup help tags that appear below the various elements of the user interface if you hover the mouse over them for a few seconds.

Options » Help Tags on Squares enables helpful help tags on the puzzle squares, listing their contents and possibilities. If you've done any hilighting, extra information is stashed in the help tags.

Options » Help Tags on Connectors adds help tags to the vector connectors, if you're using them.

Options » Colorful Blocks... and **Colorful Lines...** let you customize the appearance of the puzzle grid. Colorful Blocks only actually sets the color and saturation; the program will adjust the brightness automatically to contrast the blocks.

Options » Online Resources lists Sudoku-related websites and puzzle sources. Select an item and your web-browser will open the site.

Options » Check for updated version on launch enables and disables the automatic check for a new version that Susser performs each time you run it. There's also an option to **Check for updated version now** that will, as a side effect, print the latest version's new features.

Options » Visit Website brings you to my always entertaining website.

The PayPal and Kagi buttons

You do not have to pay for Sudoku Susser... unless you want to. If you think that the program (and the humble programmer) is worthy of your support, then you're morally obligated to send what you think the program is worth to you. It's the digital equivalent of tipping.

If you do send me a tip, you'll get... absolute nothing extra, except that warm feeling you get when you do the right thing. And you'll encourage me to continue improving the program. Also, rumor has it that generous people tend to get the features they want implemented in future versions. Which is a polite way of saying I accept bribes.

If you click one of the buttons, it'll open a new window in your web-browser and send you to the appropriate site. If for some reason you want to PayPal by hand, my PayPal email address is trebor@animeigo.com.

If you cannot (or prefer not to) use PayPal or Kagi, you can send a cheque, money order, or carefully-concealed cash to:

Robert Woodhead, 6810 Finian Drive, Wilmington, NC 28409

As US Banks are rather backward when it comes to changing foreign currencies, cheques and money orders should be in US$ only. On the other hand, while US Greenbacks are preferred, if you want to send some of your local cash, it's always interesting and I do tend to travel, so I'll probably get to use it.

The Dastardly Online Version Check

Each time Sudoku Susser launches, it connects to the internet to see if a new version is available, using the standard MacPAD protocol. Basically, it just loads a webpage that contains an XML document that describes the current version. No personal information about you is transmitted to my server.

Those of you who are convinced that this check is part of my **Diabolical Master Plan for World Domination**™ can disable this check by holding down any of the Shift, Option, Control or Command keys during launch; it can also be disabled by unchecking **Options » Check for updated version on launch**. Of course, those of you who do disable this check will be in trouble when the Nations of the Earth bow down before me!

The file Sudoku Susser requests can be found here:

http://www.madoverlord.com/MacPAD/sudoku%20susser.plist

Reporting Bugs / Making Suggestions

I welcome bug reports and suggestions on how to improve the program. Please email me at **trebor@animeigo.com**. If you find a new newspaper with Sudoku puzzles, let me know so I can update the character recognizer. If you find a puzzle that Deduce! can't handle AND you come up with a method of solving it that doesn't involve guessing or sacrificing virgins, I'd appreciate knowing about that too.

The Deduction Methods

Here is a short explanation of the deduction methods the Sudoku Susser uses.

It can't be anything else

Finds squares that are already constrained to a single value only. This will find and set the squares hilighted in green. You can't turn this method on and off, it's always on. This is equivalent to **Solve » Make All Forced Moves**.

In this simple puzzle, you can immediately see that there are many squares that can have only a single value.

Pinned Squares

Finds a group (row, column or block) in which only a single square can be a particular value; this is called a "pinned" square.

Example: if a square might be <123>, but no other square in its row can be a <1>, then it must be a <1>.

In the "Sample Pin" puzzle (shown above), R1C6 must be a <5>, because it is the only square in column 6 that can be a 5.

Simple Locked Set

If a set of N squares in a group all have the same set of N possibilities, then we can eliminate those possibilities from the other squares in the group.

Example: if there are 3 squares with possibilities <123>, then one must be 1, one 2 and the other 3; so none of the other 6 squares in the group can be 1, 2 or 3.

A set of two squares with the same two possibilities is referred to as a "locked pair"; a set of three squares with the same three possibilities would be a "locked triplet", and so on.

In the "Sample Locked Pair" puzzle (shown above), R5C6 and R6C6 are both <38>. This means that no other square in their block can be 3 or 8, and so R5C4 can be reduced to <14>. R7C1 and R8C3 are also a locked pair in the bottom-right block, but since all the other squares are solved, knowing this does not advance the solution.

Possibility Reduction

Removes extra possibilities on groups of squares that share the same subset of unique possibilities.

Example: if there are two squares with possibilities <12> and <123>, and no other squares have possibilities 1 or 2, then the second square cannot be a 3 (because one square must be a 1, and the other must be the 2).

This sometimes generates a simple locked set, which permits further reductions.

In the "Sample Constraint" puzzle (shown above), R8C1 and R9C1 are the only squares in column 1 that can be <14>. So <36> can be removed from the possibilities in R8C1, forming a locked pair on <14>.

Intersection Removal

If the squares in a row that have a given possibility only intersect a single block, then that possibility must occur in the 3-square intersection of the row and the block, and thus cannot appear in the other 6 squares in the block.

Example: consider the first row and the top-left block. If the possibility <1> only appears in the first 3 squares of the row (and so is not a possible solution for the other 6 squares), then it must be in those first 3 squares, and therefore cannot be in the other 6 squares of the top-left block.

The same removal can be done with columns vs. blocks, blocks vs. rows, and blocks vs. columns.

In the "Sample Intersection" puzzle (shown above), the value <6> can only appear in squares R1C1, R1C2 and R1C3 of row 1. Thus, the non-intersecting squares of block 1 cannot contain this value, so <6> can be removed from R3C2. Note also the other intersections in the puzzle.

Remote Locked Pairs

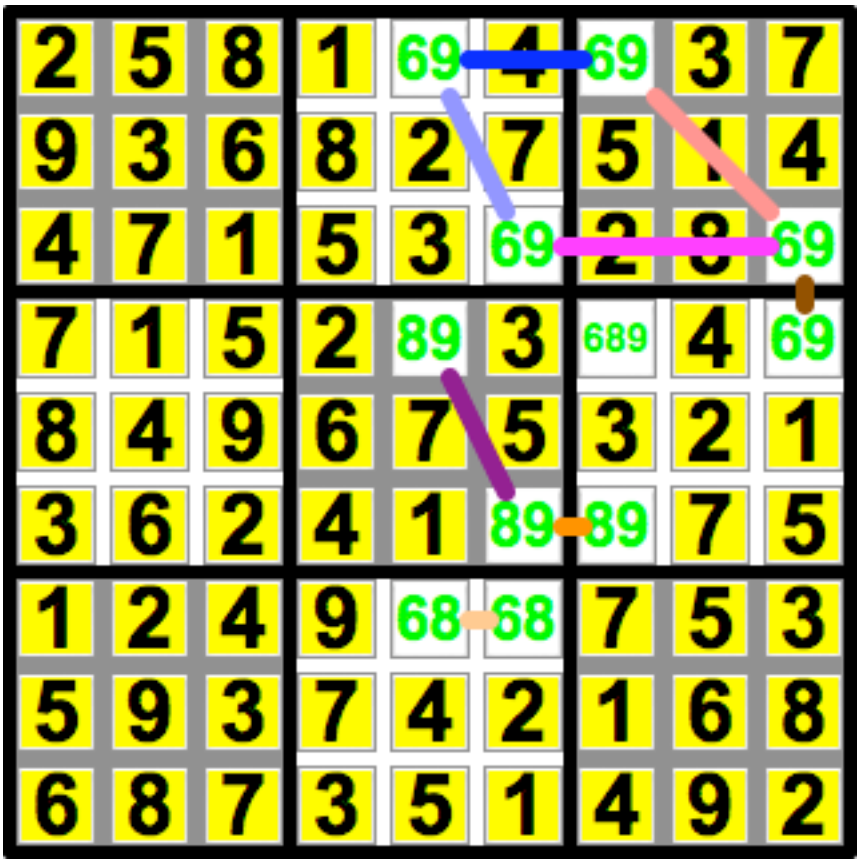
Consider two squares A and B in a row, column or block that have the same two possibilities; these form a "locked pair", and if you find out what A is, you'll know what B is.

Now consider two locked pairs B and C in an intersecting row, column or block. Now A and C form a "complimentary pair"; if you know A, you know C must be same.

Finally, consider two locked pairs C and D in yet another intersecting row, column or block. Now A and D form a "remote locked pair"; they will be in different rows, columns and blocks.

If you can find A and D, they will form two corners of a box, and you can eliminate their possibilities from the squares at the other two corners of the box.

This can be extended further; if you can find locked pairs DE and EF, you have found a remote pair AF, and so on.



In the "Sample Remote Locked Pair" puzzle (shown here), squares R1C5 and R4C9 form a remote locked pair. The easiest way to see it is turn line connectors on, and hilight locked sets.

Comprehensive Locked Set

For each row, column and block, look for a set of N squares that have exactly N possible solutions. Then those solutions must be in the N squares, and can be removed from the other squares of the group.

For example, if you have 3 squares with values <12>, <23> and <13>, they form a locked set on <123>.

Similarly, <123>,<14>,<234> and <23> form a locked set on <1234>.

Comprehensive Locked Sets are a superset of Simple Locked Sets, and like them, they are often generated by Possibility Reduction.

In the "Sample Locked Set" puzzle (shown here), you can finda locked triplet consisting of squares R2C1, R2C3 and R3C2 in Block 1.

Note that there are other locked pairs in the puzzle at this point, but none of them can be used to advance the solution since they do not permit reductions. The locked triplet lets you remove possibilities from 2 squares in Block 1.

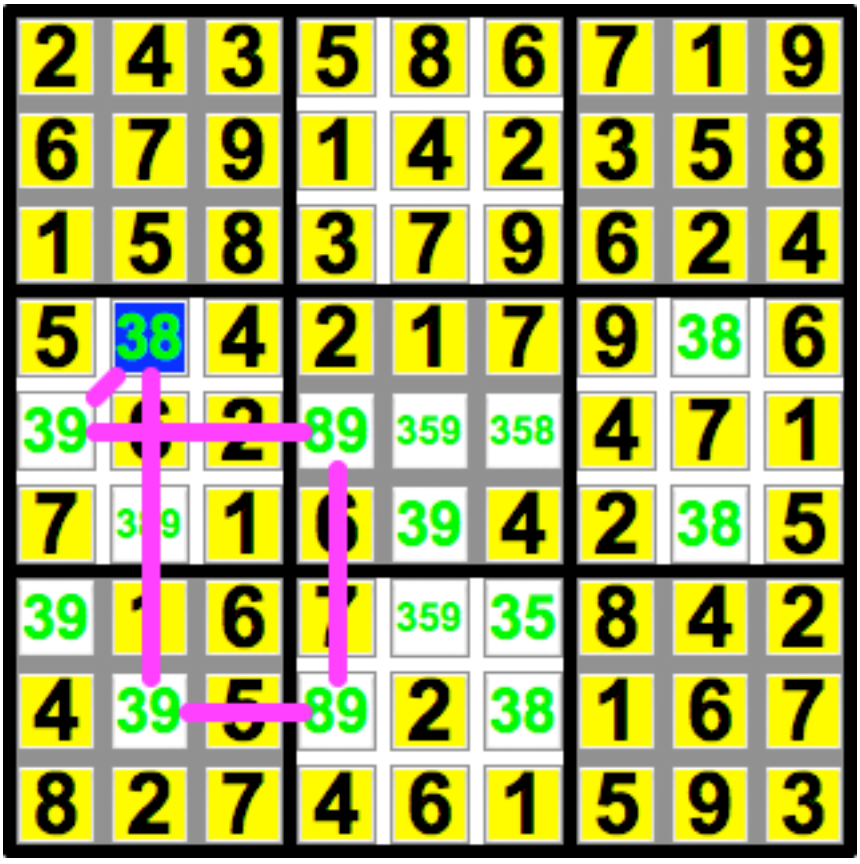
Forcing Chains

This method also comes close to guessing, but does not, I think, cross the line.

For each square that is unsolved, assume that it is one of the possible values. Then that would force other squares to have particular values, which in turn would force other squares to take on definite values.

If a chain of forced values loops back to the original starting square with a different result than our original choice, then we know that our original value cannot be correct, and it can be eliminated.

In order to emulate typical human reasoning ability, this method only follows chains of squares that have two possible values (though the starting square can have more than two). It is thus equivalent to a human moving the pencil's tip from square to square as the chain is constructed.



In the "Sample Forcing Chain" puzzle (shown here), you can find a forcing chain by starting at R4C2 and assuming it is 3. Follow the chain of forces through R8C2, R8C4, R5C4, R5C1 and back to R4C2 to find a contradiction.

Nishio

Nishio is an advanced technique, named after the fabled Nishio-san, a Japanese Sudoku Sensei. It is a form of structured guessing.

I shall try to explain the algorithm in a way that a human being might actually perform it, in particular a human being like me with a horrible memory. You will need 9 coins, a supply of counters (say, dried peas), and a pencil.

Start with a square (usually one with only two possible values). Pick a value to test.

Let's assume the square has possibilities <12> and you decide to test <1>.

Place coins on all the squares that currently must be <1> (no other possible values).

Pull the little rubber eraser off the end of your pencil (if you are solving Sudoku, you've got one!), and put it on the starting square.

Put peas on every square that has <1> as one of its possible values.

So now every square that MUST be an <1> has a coin on it, every square that MIGHT be an <1> has a pea on it, every square that can't be an <1> is empty, the starting square has an eraser tip on it, and everyone around you thinks you're insane.

Congratulations, you're ready to start Nishio'ing! Simply repeat the following loop:

- Remove the peas from every cell in the same row, column, and block as the eraser tip square.
- Replace the eraser tip with a new coin.
- If you now can find a row, column or block that has no coins and no peas, then there's no possible place for <1> to go in that group. That's a contradiction, which means you now know that your original square cannot be <1> (so, in this example, it must be <2>).
- Otherwise, if you can find a row, column or block that has exactly one pea in it, replace the pea with the eraser tip, and repeat the loop.

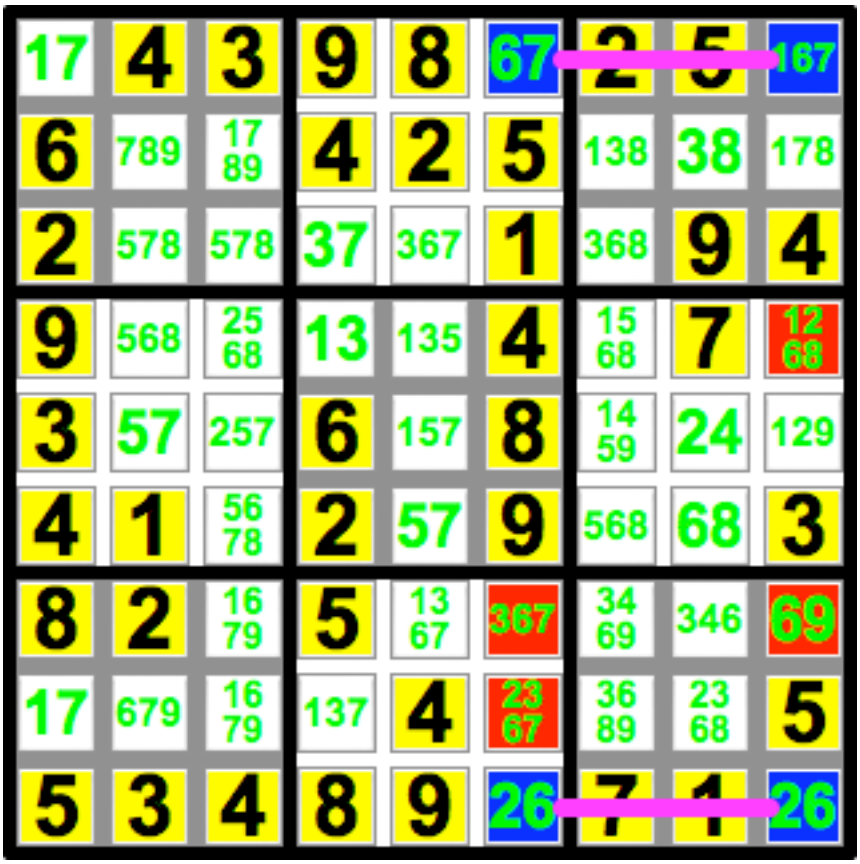
If you can't find a row, column or block with one and only one pea in it, then you can stop and say " <1> is still a possible value for the original square". So you would repeat the test with <2> as your choice.

It is important to recognize that Nishio cannot tell you what the value of a square is, it can only tell you what it isn't. So if Nishio fails to eliminate your first choice, you have to test the other possibilities in the square; you can't assume they are invalid. It is quite possible for all the possibilities to still be valid.

In the "Sample Nishio" puzzle (shown above), you can use Nishio to prove that R2C1 cannot be a 3. If you want the deducer to show you this, however, you must turn off the Forcing Chains rule.

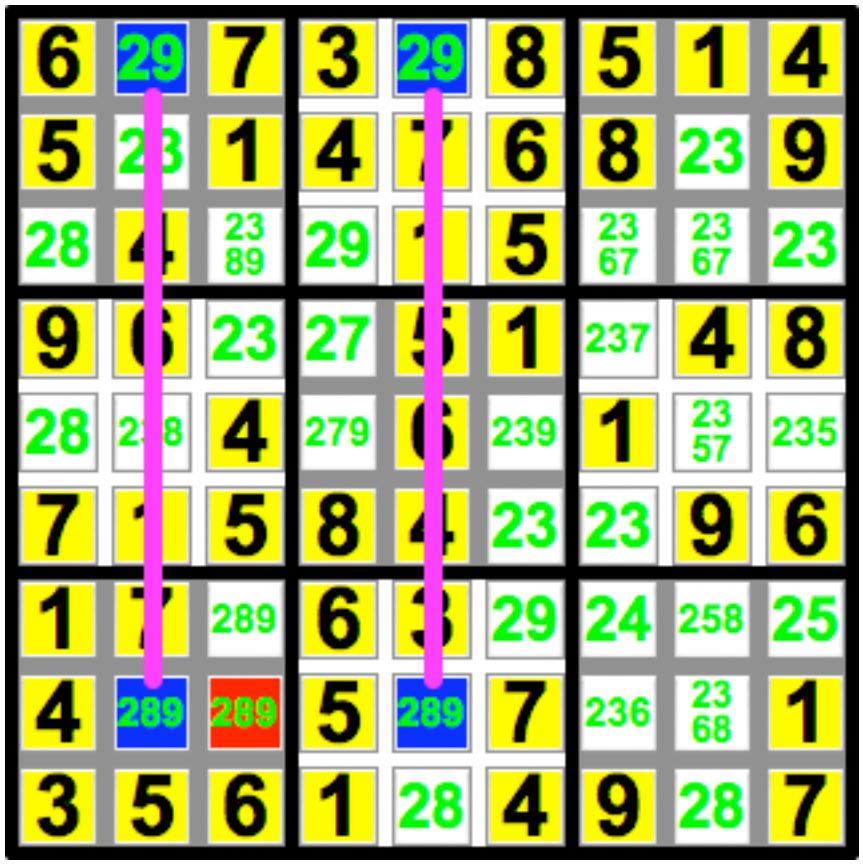
Simple X-Wings

A simple X-Wing is a common pattern that is a subset of the more complex "Fishy Cycle" pattern. To find a simple X-Wing, look for a row in the puzzle that has only two squares that can contain a particular number, then find another row that has only two squares that can contain that number. If the squares line up into the same two **columns**, forming a box shape, then you've found an X-Wing, and the number can be removed from all the other squares in the two columns. You can also look for this pattern in columns that share rows.



In the "Sample X-Wing" Puzzle (shown here), squares R1C6 and R1C9 in row 1 are the only squares in the row that can be a 6; the same goes for squares R9C6 and R9C9 in row 9. They form an X-Wing and permit 6 to be removed from 4 squares in column 6 and column 9.

It's easy to understand how X-Wings work once you think about it. In this case, if R1C6 is the 6, then R1C9 and R9C6 can't be a 6, which means R9C9 must be a 6. And if R1C6 isn't the 6, R1C9 must be, which means R9C9 can't be and R9C6 must be. Either way, columns 6 and 9 have a 6 in them in either row 1 or row 9, so rows 2-7 in those columns can't be a 6!



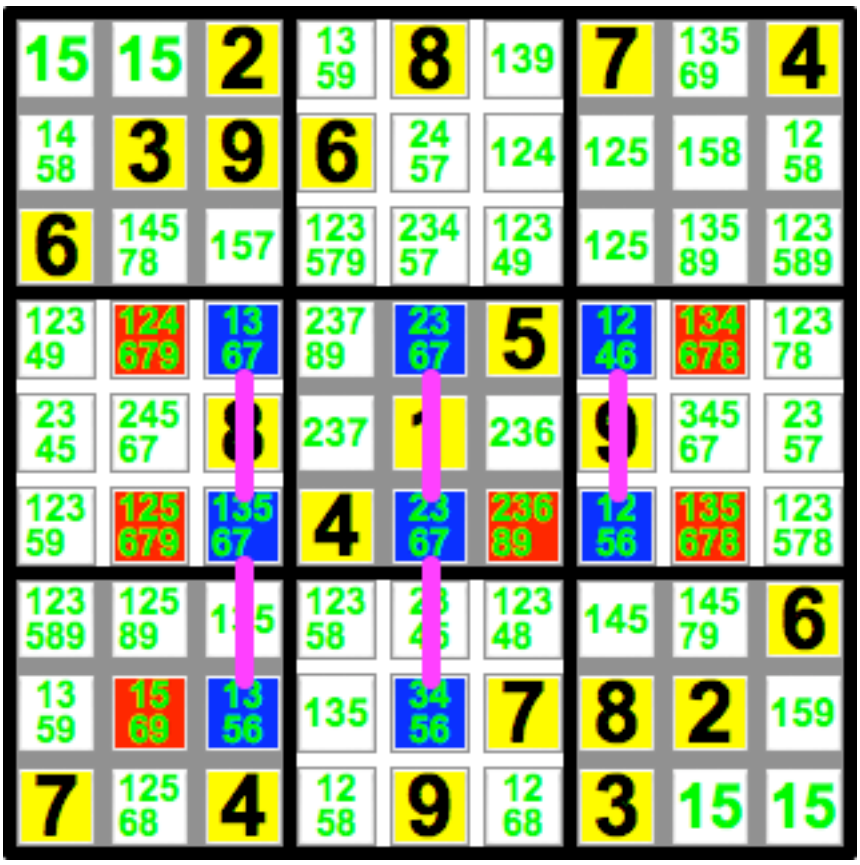
In the "Sample X-Wing #2" puzzle (shown here), you can see an example of an X-Wing that uses two columns that share two rows.

No matter what, the 9 in row 1 and row 8 has to be in one of the blue squares, so it can't be in the red square.

Swordfish

The next step up in complexity from an X-Wing is a Swordfish pattern. Instead of looking for a 2x2 set of rows and columns, Swordfish uses a 3x3 set.

To find a Swordfish, look for 3 rows each containing either 2 or 3 possible squares for a given number. If these fall on exactly 3 common columns, each containing at least 2 squares, then you've found a Swordfish, and any other instances of the number in the columns can be removed. Similarly, you can look for 3 columns sharing 3 common rows.



In the "Sample Swordfish" puzzle (shown here), if you immediately hilight (f) you'll see a Swordfish, although you don't need to use it to solve the puzzle.

Note that two of the columns in this Swordfish have 3 squares that can be a 6, while the third has only two. It is quite possible to find Swordfish with 1, 2 or all 3 columns/rows having 2 possibilities.

The "each (common column/row) containing at least 2 squares" restriction above isn't really needed, but a Swordfish that has a common column or row with only one square will first show up as an X-Wing plus some other simple reductions.

Jellyfish

Instead of looking for a 2x2 set of rows and columns (X-Wing), or a 3x3 set (Swordfish), finding a Jellyfish requires using a 4x4 set!

To find a Jellyfish, look for 4 rows each containing from 2 to 4 possible squares for a given number. If these fall on exactly 4 common columns, each containing at least 2 squares, then you've found a Jellyfish, and any other instances of the number in the columns can be removed. Similarly, you can look for 4 columns sharing 4 common rows.

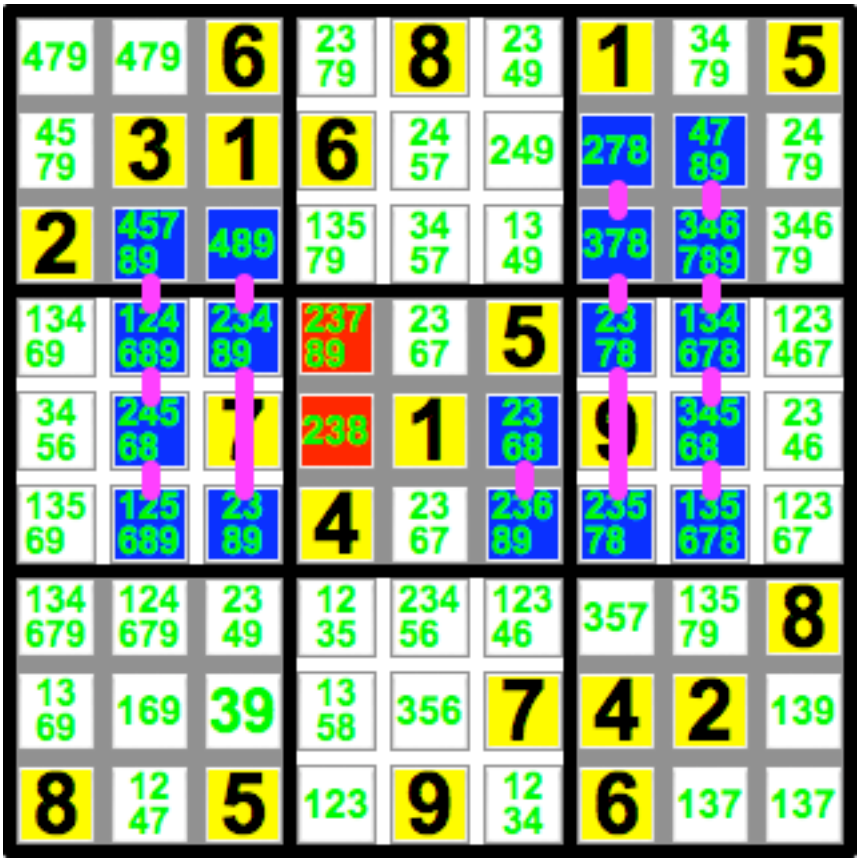
In the "Sample Jellyfish" puzzle (shown here), after a single intersection is performed, a JellyFish will appear.

Jellyfish are exceedingly rare creatures!

Squirmbag

If you thought Jellyfish were insane, consider Squirmbags. As you've probably guessed, they are the 5x5 pattern.

To find a Jellyfish, look for 5 rows each containing from 2 to 5 possible squares for a given number. If these fall on exactly 5 common columns, each containing at least 2 squares, then you've found a Squirmbag, and any other instances of the number in the columns can be removed. Similarly, you can look for 5 columns sharing 5 common rows.



In the "Sample Squirmbag" puzzle (shown here), you can hilight a Squirmbag on possibility 8, but the deducer does not need to use it to solve the puzzle.

Squirmbags (and higher-order patterns of the same type) are not required to solve sudoku because simpler methods (like pins and intersections) or lower-order patterns embedded in them will always be available that permit reductions. Still, I couldn't resist adding a Squirmbag rule!

XY-Wings

XY-Wings are interesting patterns formed by 3 2-possibility squares.

To find an XY-Wing, look for a square with two possibilities; this is the **XY** square, and its two possibilities are **X** and **Y**. Now look in that square's "buddies" (the other squares in the XY square's row, column and block; every square in the puzzle has 20 buddies) for two other 2-possibility squares, one with possibilities **XZ** and the other with possibilities **YZ**, where **Z** is not **X** or **Y**.

If you find an XY-Wing, then all the squares that are *buddies to both XZ and YZ* cannot contain Z.

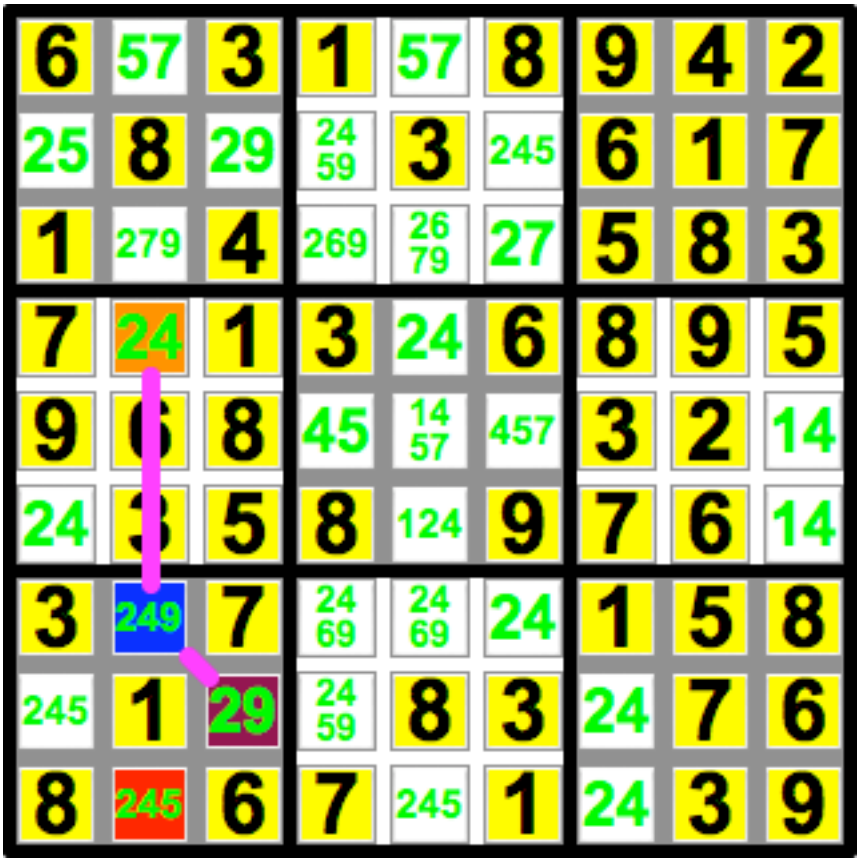
In the "Sample XY-Wing" puzzle (show here), XY is R6C1, XZ is R4C2, and YZ is R6C9. X is 8, Y is 9 and Z is 1.

It's easy to see how XY-Wings work: if R6C1 is 8, then R4C2 must be 1, so R4C7 cannot be 1. And if R6C1 is 9, then R6C9 must be 1, so R4C7 cannot be 1. Either way, R4C7 can't contain a 1!

In a "Degenerate XY-Wing," XY, XZ, YZ and the square(s) that can't contain Z are all in the same row, column or block; this is the same as a comprehensive locked set.

XYZ-Wings

XYZ-Wings are a twist on XY-Wings. If you can find a square with 3 possibilities **XYZ** that has buddies with possibilities **XZ** and **YZ**, then squares that are buddies with *all three of the squares* cannot contain Z.



In the "Sample XYZ-Wing" puzzle (shown here), the XYZ square is R7C2, and Z is 2. R4C2 and R8C3 are the XZ and YZ squares, and R9C2 can't contain a 2.

If R7C2 is 2, then R9C2 can't be 2; if R7C2 is 4, then R4C2 is 2, and R9C2 can't be 2; finally, if R7C2 is 9, then R8C3 is 2, so R9C2 can't be 2.

In a real XYZ-Wing, XYZ will always be in the same block as either XZ or YZ, and the third square will be in a row or column that intersects the block.

In a "Degenerate XYZ-Wing," XYZ, XZ, YZ and the square(s) that can't contain Z are all in the same row, column or block; this is the same as a comprehensive locked set.

Fishy Cycles

Fishy Cycles are the general case of the X-Wing, Swordfish, Jellyfish and Squirmbag patterns. However, Fishy Cycles are not restricted to rows and columns; they can include blocks as well, which makes them much more powerful (and hard to see).

I'm still trying to write up a good explanation of how Fishy Cycles work that will help you find them; in the meantime, here is the discussion that explains how they work:

http://www.setbb.com/phpbb/viewtopic.php?t=35&mforum=sudoku

In the "Sample Fishy Cycle" puzzle (shown here), a fishy cycle with link number 2 exists that permits 3 cells to be reduced. However, the puzzle also contains a Forcing Chain at this point. Load the puzzle, turn off the Forcing Chains method, and single step to progress using the fishy cycle (or press F to just see it)

One type of Fishy Cycle that needs particular mention is the "Generalized X-Wing". In this pattern, instead of two rows and two columns, one or both of the rows or columns can be replaced by a block.

In the "Sample Generalized X-Wing" puzzle (shown here), the X-Wing on number 9 is formed by row 2 and block 9, along with columns 8 and 9.

Bowman Bingo

Bowman Bingo, named in honor of Douglas Bowman (who described the Forcing Chains and Fishy Cycles heuristics), is a human-executable recursion method. By itself, Bowman Bingo can solve almost all Sudoku puzzles, but it's a sufficient pain in the butt to use that you typically only want to try it when all else has failed.

Like Forcing Chains and Nishio, Bowman Bingo makes a guess at the value of a square and then follows the consequences of that choice, looking for a contradiction; if one is found, then the original choice cannot be correct.

In order to do Bowman Bingo by hand, you will need a supply of translucent chips, such as the ones used by Bingo players (hence the name). Mark them with the numbers 1-9 so you have at least 9 of each number.

Pick a square that you wish to test, and pick a possible number for that square. Take a chip of that number and put it on the square upside-down. Now repeat the following loop:

- Find an upside-down chip and turn it rightside-up. This is the current chip.
- Find squares in the same row, column and block as the current chip that are forced to a single value (after eliminating the values used by rightside-up chips), and place an upside-down chip with that value written on it on the new square.

Example: you just flipped a <1> chip on R1C1. R3C3 (in the same block) has possible values <123>, and there's a previously-flipped chip on R3C8 that has value <3>. So R3C3 is forced to <2>, and you put an upside-down <2> on it.

You can also place chips on squares that get pinned to single values (eg: there are the only square in a group that has a particular possibility). This usually greatly reduces the number of cycles.

- If any row, column or block now has 2 chips with the same number (either rightside-up or upside-down), then you've got a contradiction, and your original choice can be eliminated. Bingo, you're a winner.
- If you have any unflipped chips, repeat the loop.

If you end up with all your chips flipped, then you're not a winner (unless, of course, you've got chips on all the unsolved squares, in which case, you've lucked into the solution of the puzzle!). Pick another square or number, and try again.

It is entirely unclear at this point whether there are ways of picking squares to test that increase your chances of a Bingo. My guess would be, pick squares whose solution will tell you a lot about the rest of the puzzle. It will also be interesting to see if there are more complicated patterns than the "2 chips in a group with the same number" that indicate a contradiction.

In the "Sample Bowman Bingo" puzzle, after a possibility reduction, a 4-way locked set(!), a fishy cycle, a nishio reduction, and an intersection, you'll find a bingo that unlocks the solution to the puzzle. However, you must turn off Forcing Chains and Trebor's Tables to see it.

No graphic illustration of Bowman Bingo is presented because I haven't done a hilighting feature for them yet!

Trebor's Tables (the simple introduction)

Your humble author is experimenting with a new technique called tabling. Tabling seems to be the key to cracking the toughest puzzles, and in fact is needed to crack the toughest known puzzle ("Toughest Known Puzzle" in the puzzles list).

Tabling works by asking questions like: "If R1C1=1, what does that imply for the rest of the puzzle?" It builds a list of pins and forces such a move would cause (direct **implications**), as well as other, indirect implications.

If R1C1 had possible values <12>, then tabling can generate 4 **assertions** for the square; R1C1=1, R1C1=2, R1C1<>1 and R1C1<>2. Each assertion can have implications, and once you've created assertions for each square in the puzzle, you can check what the implications of their implications are, over and over, and build a complete list of the implications of each assertion.

Once this list is complete (and in fact, during the process), one can find **verities** and **veracities**. A verity is any implication that is true for all the positive assertions in a square.

Continuing the example above, if the implication R9C9=4 is true for both assertion R1C1=1 and assertion R1C1=2, then it must be true no matter what, since R1C1 must be either 1 or 2. Similarly, if R8C8=3 is true for both assertion R1C1=1 and R1C1<>1, it must be true no matter what.

Veracities are a similar idea, but they look at all the squares in a group that can have a particular value.

For example, if there are 3 squares in row 1 that can contain the value 1, R1C1, R1C2 and R1C3, then any implications that are contained in all of R1C1=1,R1C2=1 and R1C3=1 must be true, since on of them has to be correct. You can also look at the <> assertions for those squares; any implication that appears in at least two of them is also true.

Thus, tables can tell you things that are definitely true about the puzzle. But they can also tell you thinks that are definitely false. For example, if you've found that R9C9=4 is definitely true (because it's a verity or veracity), then any assertion that has the implication R9C9<>4 must be invalid.

Similarly, if you can find contradictory implications in an assertion (such as R1C1=1 and R1C1<>1), you know the assertion must be invalid. A more subtle version of this is when R1C1 = 1 or 2 and you find assertions that R1C1<>1 and R1C1<>2.

Invalid assertions can trigger cascades of invalidation, and eventually you get to the point where you can make definite statements about the puzzle.

There are a lot of subtleties to tabling, and it is still under development. However, at this point, tabling is sufficiently mature that it can solve all known puzzles! It is thus a superset of all other methods.

It goes without saying that if you find a puzzle that Tables can't crack with all its options on, I'd like to see it!

Note: while tabling is more powerful than Bowman Bingo, it is run before it. This is because when some tabling options are disabled (see next section), tabling can fail to generate a solution, but Bowman can.

The "Toughest Known Puzzle" puzzle must be tabled to solve.

Trebor's Tables (the complete explanation)

This section will be of interest to those who wish to play around with the various Table Settings options. Some of what follows duplicates the simple explanation given above.

As an aid to explanation, I will give examples from the tabling of the **Sample Remote Locked Pair** example puzzle provided with the Susser; this puzzle is simple and thus generates small tables. The sample tables below assume that all tabling options are off, except for **Propagate contradictions**.

Tabling works by generating a list of **Assertions** for each square, and figuring out their **Implications**. If a square has *n* possible values, it will have *2n* assertions.

For example, square R1C5 has possible values <69>, so there will be 4 assertions; R1C5=6, R1C5=9, R1C5<>6 and R1C5<>9.

The first step is to compute the **initial implications** of each assertion. This is done by simply making the move, and recording what changes this makes in the puzzle. We also must **propagate forces and pins**; this means that if the move causes another square to be forced to a single value, or pinned to a single value (it's the only square in a row, column or block that can have that value), then we consider what the effects of this forced move will be.

Normally, the tables algorithm only looks at the effects of forces and pins in the row, column and block of the initial square, because these are easier for human beings to recognize. If **Aggressive Forces and Pins** is enabled, then it considers and propagates the effects of forces and pins in all rows, columns and blocks. This is required to solve some very tough puzzles, and finding out if there is a middle ground that is easier for humans to perform is a subject of investigation.

So for example, the initial implications of **R1C5=6** are:

```
R1C5<>9 (R1C5=6 implies R1C5<>9)
R1C7=9 (R1C5=6 forces R1C7=9)
R1C7<>6 (R1C5=6 indirectly implies R1C7<>6)
R3C6=9 (R1C5=6 forces R3C6=9)
R3C6<>6 (R1C5=6 indirectly implies R3C6<>6)
R4C5=9 (R1C5=6 column pins R4C5=9)
R4C5<>8 (R1C5=6 indirectly implies R4C5<>8)
R7C5=8 (R1C5=6 forces R7C5=8)
R7C5<>6 (R1C5=6 indirectly implies R7C5<>6)
```

(To have the susser generate this list, load the Sample Remote Locked Pair puzzle, make R1C5 the current square, and press "t")

Some of these (such as the first) may appear to be stating the obvious, but the more implications you can find for an assertion, the better! It should be obvious that R1C5=6 means that R1C7=9, since the two squares are locked pairs, and that this indirectly implies that R1C7<>6. The other implications are similarly derived, except for R4C5=9. This is clearly true because if R1C5=6, then R7C5=8 is forced, which leaves only 9 as a possible value for R4C5. This is a classic pin.

If you were doing this table by hand, you'd write down a table line for this assertion as follows:

```
R1C5=6      R1C5<>9, R1C7=9, R1C7<>6, R3C6=9, R3C6<>6, R4C5=9, R4C5<>8,
            R7C5=8, R7C5<>6
```

Once we have all the initial assertions defined, we now proceed to **expand assertions**. This is simple: since each implication in an assertion is also an assertion somewhere in the table, we simply look at it and add it to our assertion.

So, to expand R1C5=6, we first look at the implication R1C5<>9, which is:

```
R1C5=6 (R1C5<>9 forces R1C5=6)
R1C5<>9 (R1C5<>9 implies R1C5<>9)
R1C7=9 (R1C5<>9 forces R1C7=9)
R1C7<>6 (R1C5<>9 indirectly implies R1C7<>6)
R3C6=9 (R1C5<>9 forces R3C6=9)
R3C6<>6 (R1C5<>9 indirectly implies R3C6<>6)
R4C5=9 (R1C5<>9 column pins R4C5=9)
R4C5<>8 (R1C5<>9 indirectly implies R4C5<>8)
R7C5=8 (R1C5<>9 forces R7C5=8)
R7C5<>6 (R1C5<>9 indirectly implies R7C5<>6)
```

In this particular case, since R1C5=9 is really the same as saying R1C5<>9, we don't learn anything new at this point. But let's look at R1C7=9, which has implications:

```
R1C5=6 (R1C7=9 forces R1C5=6)
R1C5<>9 (R1C7=9 indirectly implies R1C5<>9)
R1C7<>6 (R1C7=9 implies R1C7<>6)
R3C9=6 (R1C7=9 forces R3C9=6)
R3C9<>9 (R1C7=9 indirectly implies R3C9<>9)
R4C7=6 (R1C7=9 column pins R4C7=6)
R4C7<>8 (R1C7=9 indirectly implies R4C7<>8)
R4C7<>9 (R1C7=9 indirectly implies R4C7<>9)
R6C7=8 (R1C7=9 forces R6C7=8)
R6C7<>9 (R1C7=9 indirectly implies R6C7<>9)
```

(To have the susser generate this list, type x to clear the assertions, then make R1C7 the current square and press "t". If you don't clear the assertions, R1C7 will automatically get expanded to include implications it can learn from R1C5=9)

The first three implications we already know about, but the rest can be added to R1C5=6. If we were doing the table by hand, we would now have:

```
R1C5=6      R1C5<>9, R1C7=9, R1C7<>6, R3C6=9, R3C6<>6, R4C5=9, R4C5<>8,
            R7C5=8, R7C5<>6, R3C9=6, R3C9<>9, R4C7=6, R4C7<>8, R4C7<>9,
            R6C7=8, R6C7<>9
```

This process is continued for every implication in R1C5=6, including the new ones we just added. Once we've expanded all the assertions, we've completed one **cycle of expansion**.

At this point, if we want, we can **check the assertions for contradictions**. Some people don't like to prove puzzles by contradiction, which is why one of the Table Settings is **Propagate Contradictions**. However, doing so greatly reduces the amount of work you need to do in order to solve the puzzle, so I think it's a great idea.

This is easy: look at the assertion and its implications and see if any of the implications contradict each other (or the original assertion). There are several types of these:

Assertion contradictions: You find an implication that directly contradicts the assertion (ie: Assertion R1C1<>1 contains implication R1C1=1)

Direct contradictions: You see implications that directly contradict each other (ie: Assertion R2C2=2 contains implications R1C1=5 and R1C1<>5)

Exclusions: You see a set of implications that exclude all possible values for a particular square (not necessarily the square that is the subject of the assertion!). So for example, if R1C1 has possible values <123>, and the assertion R3C3=5 has implications R1C1<>1, R1C1<>2 and R1C1<>3, then R3C3=5 must be false.

As a useful-side effect of looking for exclusions, you can also easily find **implied forces and pins**. Using the above example, if you found that R1C1<>1 and R1C1<>3, then you can add an implication to assertion R3C3=5 that R1C1=2. You can do similar deductions for pins, though these are only needed to solve the toughest puzzles. Basically, when doing by hand, you add them if you notice them.

If a contradiction is found, then it follows that the assertion is invalid, and that furthermore, any assertion that contains it as an implication must in turn be invalid.

When doing a table by hand, you'd simply cross out any assertion you found to be invalid, which would make it easy to see them when you look them up during expansion.

Even better, *any positive implication (ie: R1C1=1) that becomes invalid is a change you can make to the puzzle*! You've made progress.

Another useful technique that can be applied during expansion is **expanding negative assertions**. Consider a square like R4C7 that has 3 possible values; it thus has 3 negative assertions (such as R4C7<>6). It turns out that any implications that are true for all of the non-matching positive assertions of the square must also be true for the negative assertion. In this case, this means that any implications shared by R4C7=8 and R4C7=9 must also be true for R4C7<>6, and can thus be added to that assertion.

Expansion can continue until you do a complete expansion pass and don't find any new things to write down. At that point, the puzzle is said to be **completely expanded**.

However, at any point during expansion (or indeed, just at the end), you can take some time to look for two special types of inference about the puzzle, which we've dubbed **verities** and **veracities**.

A **verity** is any implication that is true *for all valid positive implications of a square!* For example, if R1C1 has possible values <123>, then any implication that is true for R1C1=1, R1C1=2 *and* R1C1=3 (the *intersection* of the three assertions) must be true, because no matter what R1C1 ends up being, that implication must be true, *and can thus be applied to the solution of the puzzle*. Not only that, any assertion contradict, or contains an implication that contradicts, the verity must be invalid!

Another way to find verities is to find the intersection of a positive assertion and its associated negative assertion; for example, R1C1=1 and R1C1<>1.

A **veracity** is very similar, but looks at the implications of all the *valid positive assertions for a number in a block*. For example, if in row 1 only R1C1, R1C2 and R1C3 can contain the number 1, then the intersection of R1C1=1, R1C2=1 *and* R1C3=1 must be true, and, like verities, can be applied to solving the puzzle and contradicting assertions.

Another way to find veracities is to look for implications that appear in *2 or more of the valid negative assertions for a number in a block*. This is because for each pair of negative assertions, at least one must be true (because both of them cannot be false, which would mean two squares had the same value), so anything they have in common must be true.

If verities or veracities are found, then they can be used to contradict assertions, which in turn may reveal more verities and veracities (because there are fewer valid implications and assertions to consider, so more chances for intersections). When there are no more possible expansions, and no more verities and veracities to be found, the table is said to be **exhausted**. You know know everything about the puzzle, and it can be completely solved.

In practice, a reasonable approach is to do one or more rounds of tabling, then look for verities and veracities. Once you find some, you can immediately apply them to the puzzle, and if they result in simpler methods being useful, you can proceed using the easier techniques. **But here's a nice side-effect of tables**; if later on you get stuck and need to table again, *you can re-use the tables you have created since they are still valid!*

In fact, any progress you made using other methods simply become verities that you can add to the table and use to invalidate assertions! These will often immediately generate new results.

The Susser uses this approach. It does a few rounds of tabling until the table isn't growing much, and then it looks for verities and veracities. It stops when it has completely expanded the puzzle, has 5 or more verities or veracities, or has gone 8 rounds without finding a new verity or veracity. It then uses simpler techniques to advance the solution before resorting to tabling once more. If you want the Susser to continue tabling until it has exhausted the puzzle and found everything that can be known, select **Exhaustive Table Generation** (and be patient in some cases!)

One final note about invalidation and contradiction. There are some in the Sudoku research community that believe that solving by contradiction is somehow "bad." I do not share in this belief, and wish to point out that tabling with **Propagate Contradictions** disabled still solves all known puzzles. However, it takes longer generates larger tables. Thus, contradiction is shown to be a useful tool that permits one to concentrate on the important features of the puzzle.

Places To Find Sudoku Online

Major Newspapers

http://www.timesonline.co.uk/section/0,,18209,00.html

http://www.dailymail.co.uk/sudoku

http://www.mirror.co.uk/funandgames/sudoku/

http://www.guardian.co.uk/sudoku

http://puzzles.usatoday.com/sudoku/

http://www.thesun.co.uk/section/0,,6,00.html

USA Today uses a flash applet to deliver puzzles. Either click the options button then print the puzzle to a PDF and drag it in (or preview the print and marquee the puzzle), or use the freeware SnapNDrag app to grab it directly off the webpage.

Mennekse.no Sudoku Archive

http://www.menneske.no/sudoku/eng/

Tons of puzzles can be found here, categorized by difficulty. Sudoku Susser has special code to recognize puzzles dragged from this website and scan them properly. Simply select the entire page (CMD-A) and drag it or cut and paste it into Sudoku Susser.

Jaap Scherphuis' Sudoku Generator

http://www.geocities.com/jaapsch/sudoku.htm

Java application that can generate sudokus of various difficult levels, including nice symmetric sudokus. The 1-line encoded sudokus can be cut and pasted into the Susser.

Minimum Length Sudokus

http://www.csse.uwa.edu.au/~gordon/sudokumin.php

A site that is trying to build a list of all the known minimum-length (17 digit) sudoku puzzles.

Web Sudokus (you'll need something like SnapNDrag to grab them)

http://www.websudoku.com/

BrainBashers (a new puzzle every day, draggable into the Susser)

http://www.brainbashers.com/sudoku.asp

Michael Mepham's Sudoku.org.uk

http://www.sudoku.org.uk/daily.asp

Michael sets the puzzles for the Daily Telegraph and the LA Times, so his daily puzzle can be expected to be of high quality. He also has published some Dead Tree Sudokus for use when your laptop batteries fail.

Acknowledgements and Resources

I must acknowledge my debt to **Douglas Bowman**, Associate Professor of Mathematics at Northern Illinois University. Douglas was the person who developed the Fishy Cycles and Forcing Chains methods, and described in sufficient detail to permit me to program them without too much blood leakage from my ears. He has also contributed much to the creation of the Bowman Bingo and Trebor's Tables algorithms.

Richard A. Fowell must also be singled out for his contributions to the program. Richard extensively tested the OCR graphic puzzle recognition system, and was diabolical in his ability to find "problem" puzzles that it didn't read -- but should have. Finding ways to deal with them in a reasonably elegant manner greatly improved the recognizer.

If you have a PalmOS PDA, check out **Andrew Gregory's Sudoku for PalmOS**.

http://www.scss.com.au/family/andrew/pdas/palm/myprogs/sudoku/

You'll also need the Sudoku Importer; see **http://sudoku.latte.ca/**

The **Sudoku Programmers Forum** is a great place to learn about Sudoku solving methods:

http://www.setbb.com/phpbb/viewforum.php?f=1&mforum=sudoku

I was inspired to develop the deductive solver by **Peter Wake's Sudoku Solver by Logic** project:

http://www.sudokusolver.co.uk/

The Remote Locked Pairs method is better explained (well enough for me to implement it), by **Andrew Stewart** at:

http://www.scanraid.com/remotepairs.htm

Steve Blott's Sudoku Solver and explanations of some of the heuristics were extremely helpful. See:

http://www.blott-online.com/sudoku/index.html

Vegard Hanssen for his Sudoku Archive and permission to fetch puzzles from it.

The following are just some of the people have made bug reports or suggestions that I have incorporated:

Ian Orchard, Douglas Bowman, Chris Clark, James Belot, David Wentroble, Graham Harrison, Andrew Jobbings, and Vindaloo.

Sudoku Susser uses the Aqua About Box code, created by Steve Forbes and released in the public domain:

http://www.ravenna.com/~forbes/yonk/source/

An excellent article on the history of Sudoku can be found here:

http://www.maa.org/editorial/mathgames/mathgames_09_05_05.html

And a wonderful illustrated tutorial on solving techniques (written by a fellow Sudoku application author), can be found at:

http://www.angusj.com/sudoku/hints.php

And finally, I would like to thank my son Alex, who wanted a math puzzle to do one day, which made me remember about this silly puzzle I'd heard about that everyone in the UK was going nuts over...

Unanswered Windows Questions

My ability to test the Windows version is limited to what I can try on an old Windows-98 machine. So there are several features of the app that I can't really test, but which I'd like to know more about.

If you can play with these features on other Windows platforms and let me know how they work, much appreciated:

Dragging Puzzles into the Susser from your hard disc

Windows98 won't let me drag entire folders into the application. It also won't recognize PNG, JPG and PDF files as graphic files, only GIFs.

Dragging Puzzles from the Susser to somewhere else

I don't have any applications that can accept drags from the Susser. Notepad won't accept text drags, for example. If you know of freeware apps that can accept text and/or graphic drags and run on Win98, let me know.

Also, the current drag modifiers (option/alt) and control may not work properly on Windows. In particular, I may have to remap control on Windows machines to another key. Let me know.

Internet Features

The test machine doesn't have an internet connection, so all these features are untestable by your humble author. Rumor has it that they work.

General Fit and Finish

The app looks slick on the Mac, less so on Win98. If there are any rough edges you notice, send me a screenshot and perhaps I'll be able to tweak things.

PS: RTFM means "Read The Fantastic Manual". Other people use a different F-word, though. And "suss" is a wonderful Britishism. Look it up!