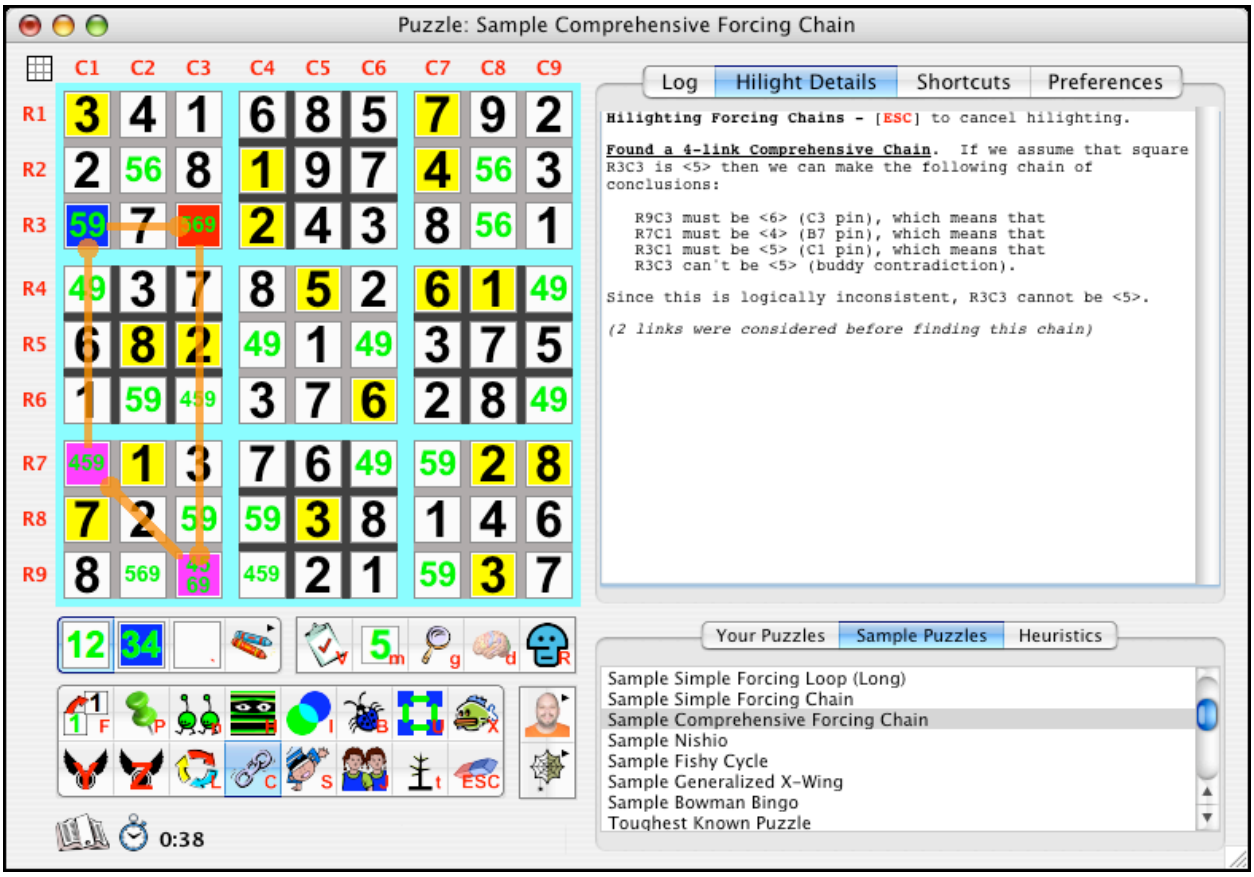
The Sudoku Susser



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 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. If you are updating to a newer version of the Susser, don't copy over the file!

If you find a bug in the Susser, please report it! But check the Known Bugs section at the end of the manual first; it has details on how to contact me, what I'll need from you, and a list of the current outstanding bugs. Note that some of the images in the manual may be from earlier versions of the application.

The Sudoku Susser is Tipware

The Susser is provided to you at no charge, totally uncrippled. Should you choose to do so, you can "tip" your humble author, paying him what you think the program is worth -- to you!

You decide whether you want to tip, and how much you want to tip.

The Susser represents over 9 months of work, several hours a day. It's a labor of love, but still quite a bit of labor.

I trust you will do the right thing. The program will not nag you if you don't -- your conscience will!

Clicking on the "humble author" icon will pop up a menu that lets you email me, tip via PayPal or Kagi, and visit the website.

If you prefer to contribute by mail, my address is:

Robert Woodhead
6810 Finian Drive
Wilmington, NC 28409

My email is trebor@animeigo.com. I welcome your comments, suggestions and bug reports.

I hope you enjoy using the Susser as much as I enjoyed creating it.

Note: if you have bought either of the Sudoku books I have co-authored, "Brainiac Sudoku" (for kids) and "Sudoku to Go" (for big kids), you may use the Susser without feeling any obligation to tip. It's my way of thanking you for helping me kill a few trees...

About the Screenshots

Some of the screenshots in this manual (okay, most of them!) are from recent versions of the app, so they might not quite match what you see on the screen -- but all the details important to the feature being discussed should be correct.

Also, of course, they show the Mac version.

What are Sudoku?

A sudoku puzzle has 9 **rows**, 9 **columns**, and 9 **blocks** of 3x3 squares each. Collectively, these are called **groups**.

Each of the 81 **squares** has 20 **buddies**, the other squares in its row, column and block.

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

For a more detailed introduction to sudoku puzzles, I recommend either the provided *Sudoku4kids.pdf* document, or visiting the Daily Mail Sudoku pages:

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

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 I only have a basic XP Windows machine, so my Windows testing is a bit limited.

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.

Important: If you ever get a "running out of system resources" message while using the Susser, quit the app and restart it. RealBasic apparently has a memory leak that affects certain operating systems variants (Win95 and ME for sure).

I am also able to produce a version that runs on **Linux**, which I cannot test much at all. I'm told it works reasonably well, but has some rough edges.

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. If you are updating to a newer version of the Susser, don't copy over the file!

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***. Depending on what ***hint*** and ***hilight*** modes you've selected, the puzzle will be colored in different ways, and may even have lines drawn on top of it.

Below the puzzle are the ***buttonbars***; these let you access almost all of the Susser's features. Also found here are a ***timer*** and an ***undo slider***.

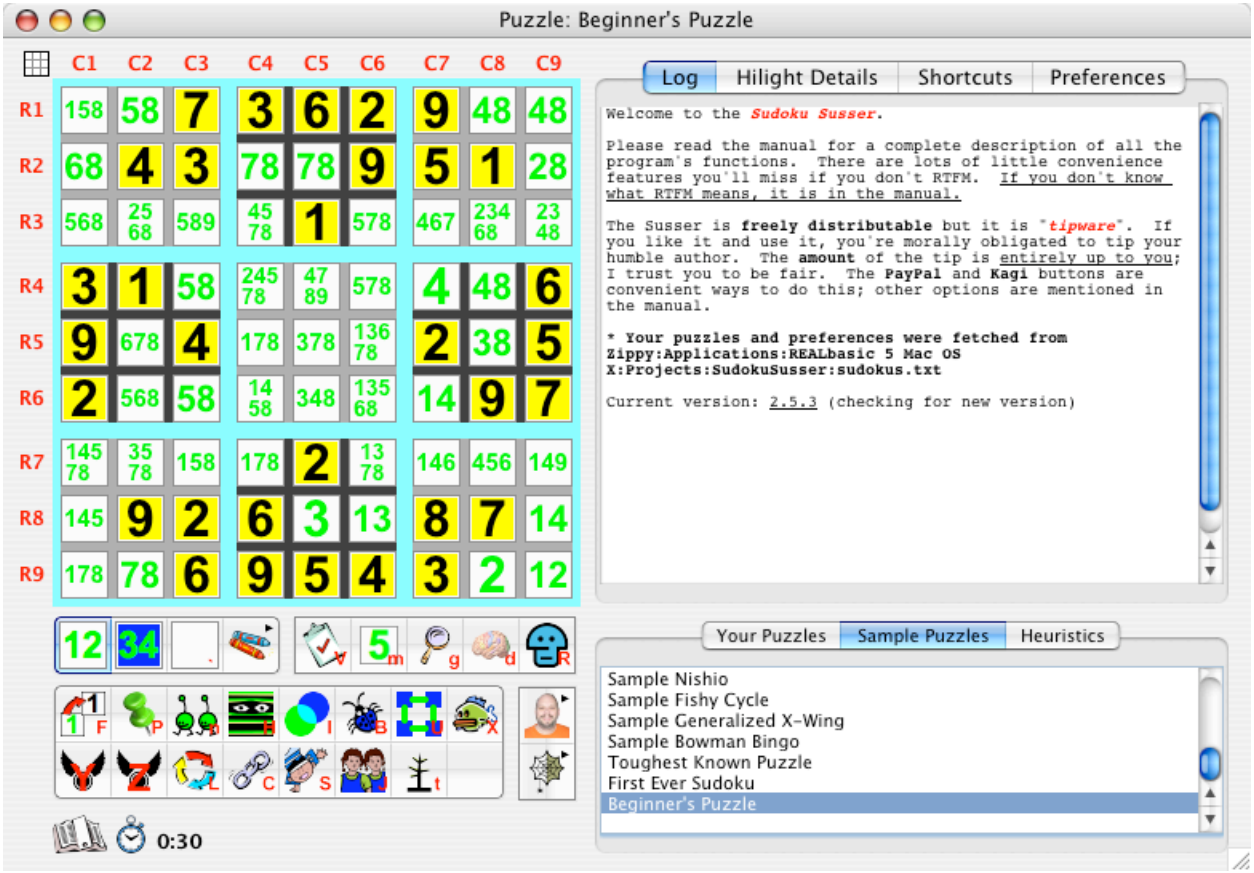
To the right of the puzzle is the ***info panel***; there are several tabs that let you access various displays and preferences.

Below the info panel is the ***puzzles list***; it has three tabs associated with it; ***Your Puzzles*** displays all the puzzles you currently have stored in your ***sudokus.txt*** file, ***Sample Puzzles*** shows you all the sample puzzles provided with the program, and ***Heuristics*** displays a list of deduction rules with checkboxes; these let you decide what techniques the program will use when you ask it to deduce some moves for you.

The program tries to be as helpful as possible: hover the mouse over a feature and you'll get a popup ***helptag*** that explains what it does (these can be turned off in the preferences).

How to Suss Sudoku

If you've just launched the Susser for the first time, you'll see something like this.



Since you don't have any puzzles of your own yet, the Susser will load and display the "Beginner's Puzzle" sample puzzle, which we'll use in this tutorial.

The original solved squares in the puzzle are shaded yellow with a black digit. Squares you have solved will be white with a black digit (none of those yet).

Unsolved squares are white with green digits. The digits in them represent the possible values that the square can still have, which I usually refer to as the "possibilities".

You can change how the Susser displays unsolved squares by using the ***hints button bar***, which looks like this:

The options, left-to-right, are: **hints on all squares**, **hints only on hilighted squares**, **no hints at all** and the **crayon popup menu**.

Your current selection is hilighted in pale blue. Note that the "no hints" button has a small red ` character in it. This indicates that if you type ` (the key above the tab key and to the left of the 1 key), that button will be selected. ***Anytime a button has a keyboard shortcut, the key will be in the button in red.***

If you type ` or click on the "no hints" button, you'll see this:

I also clicked on the ***Shortcuts*** tab of the log panel to display a list of the most common keyboard shortcuts. You'll find it handy to have this available while you're learning the program.

Note that the red ` has moved to the "hints on all squares" button! You can type ` again to move back to showing all the hints. Please do so now.

Moving around the puzzle

Looking at the puzzle, you'll see that several squares have only one possibility. This means they are ***forced*** to be that value, and can immediately be solved.

To solve a square, you must first select it. You can do this by either moving the mouse over it, or using the **keyboard arrow** keys. As you do so, a red selection box will appear around the selected square.

In addition to the keyboard keys, you can also type **Tab** to move to the next square "typewriter-style", scrolling down to the next row when you go off the right edge, **Shift-Tab** to do the opposite, **Home** to go to the top-left square, and **End** to go to the bottom-right square.

Play around with moving around the puzzle for a bit, and stop with the selection box on Row 4, Column 7 (R4C7), which only contains one possibility, 4. You should see something like this:

Solving squares

To solve a square, once it's selected, just type the number you want it to be. So in this case, you'd type 4, and see:

Note that not only did the 4 in R4C7 turn from green (unsolved) to black (solved), but the 4 possibility was removed from all the squares in R4C7's row, column and block. So R4C8 changed from 48 to 8, and is now itself forced.

When you solved the square, a message appeared in the log (one of the panes of the info panel) to that effect. Whenever you do something, the Susser tells you what just happened in the log.

Toggling Possibilities

When solving more advanced puzzles, you can often make deductions that reduce the number of possibilities in a square but don't yet solve it. You can toggle a possibility in a square by selecting the square and typing **Shift-#**, where # is the possibility you want to toggle.

For example, R1C8 and R1C9 are both 48, which means they form a "naked pair". This means that one of them must be the 4, and the other the 8, so no other square in row 1 can be 4 or 8; this permits you to remove the 8 from R1C1 and R1C2.

If you select R1C1 and type **Shift-8** (* on US keyboards), you'll see that the 8 disappears from that square...

Press **Shift-8** a few times to see the 8 reappear and disappear. Notice that no other squares change; toggling a possibility does not change any of the other squares.

Note: Some international keyboards have different **Shift-#** *combinations. The Susser tries to understand this, but it can't always get it right. If* **Shift-#** *does not work for you, you can use the two character sequence - then # to toggle possibilities.*

Also, on the Mac, you can use **Shift-Numeric Keypad #** *to toggle possibilities, but on Windows, this doesn't work. Thank your Uncle Bill for this restriction.*

Fixing Your Mistakes

Now remove the 8 from R1C2, and you'll see it has only a single possibility, so you can solve it. But instead of typing 5 to solve it, type **Shift-5** to remove possibility 5. As this leaves the square with no possibilities, the Susser complains that something's wrong!

You can recover from a mistake by selecting Undo from the Edit menu, or typing **Cmd-z**. But there's also a cooler way, using the ***undo slider***:

Just grab it and wiggle it back and forth and you can back up as many steps as you'd like, and the puzzle will update as you do so!

Back up until R1C2 contains possibilities 5 and 8 again.

Bookmarks

 The Bookmark icon in the bottom-left corner of the screen lets you mark a point in the puzzle so you can later return to it. Click on the icon to set the bookmark, and it will change color to indicate a bookmark has been saved.

 Once the bookmark has been set, if you click on the icon you'll get a popup menu that lets you either return to the bookmark or change it to your current position.

The bookmark not only saves the state of the puzzle, but also the state of the undo buffer, so you can bookmark, back up in the puzzle, make changes, then decide to return, and everything will be restored.

Other Hints Options

Returning to the hints buttonbar, there are a couple of features that I haven't explained yet:

The **hints only on hilighted squares** button will only become useful when you start using hilighters. It lets you display the possibility hint digits only on squares that have been specially hilighted.

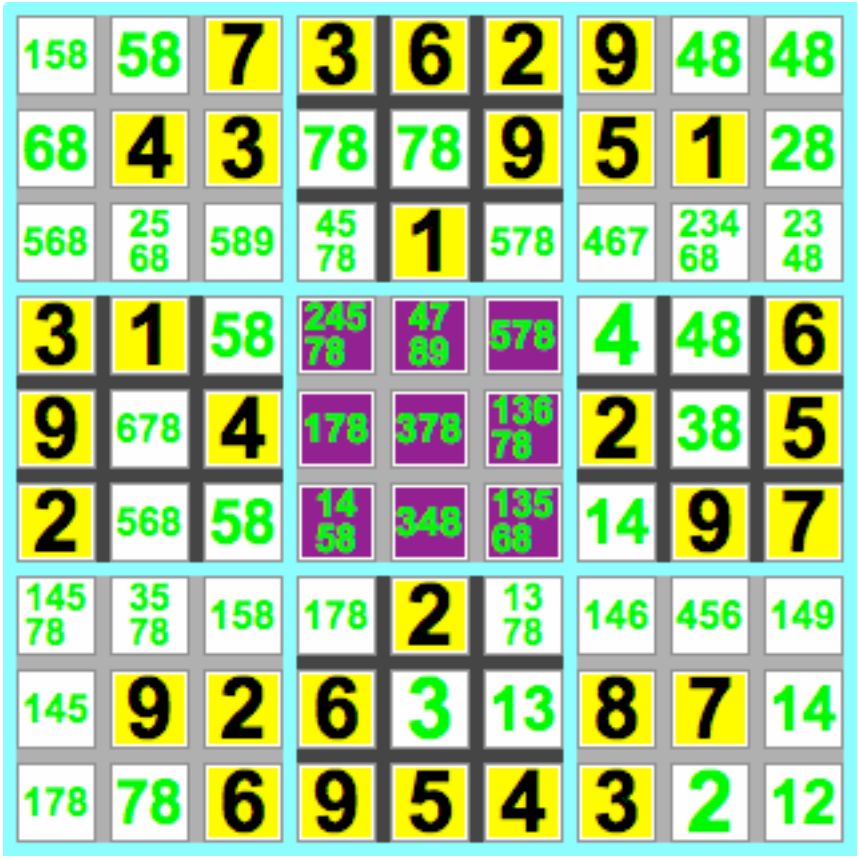
The **crayon popup menu** gives you some extra ways to color squares; these may be helpful to some users, but most people won't use them. There are three suboptions:

Pink Possibility (=)... lets you change the color of a particular possibility digit from the normal green to a pinkish hue. For example, if you choose to "pink" 5's, you'll see this:

As you have probably guessed, you can select pink possibilities via the keyboard by typing =, then the # you want to pink. To turn off pink possibilities, you can type =, then **ESC**.

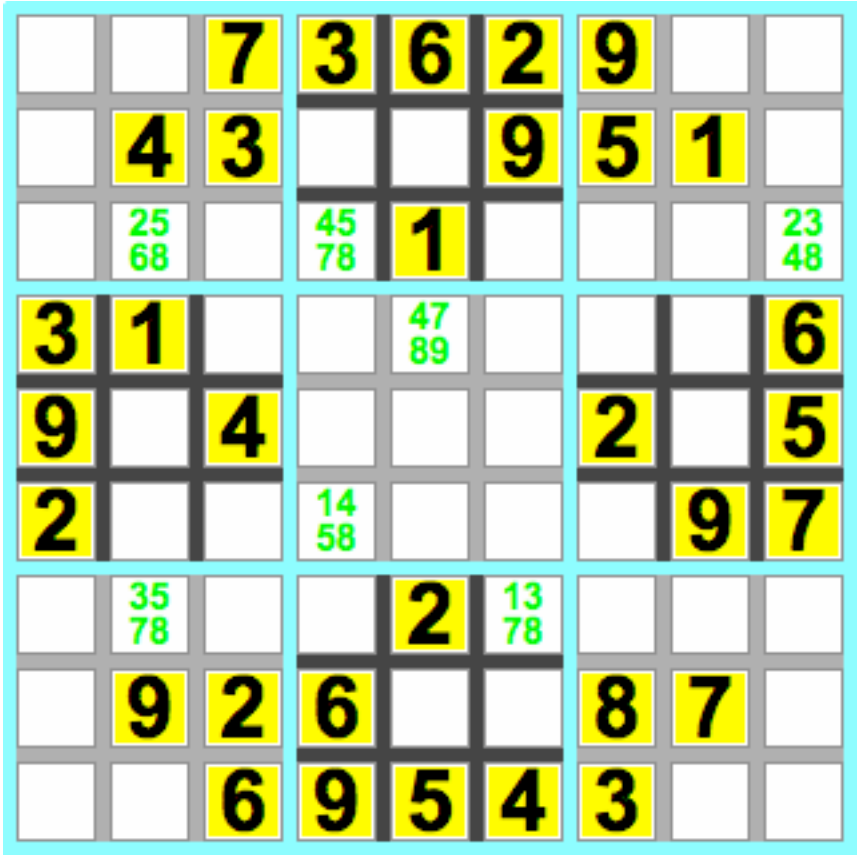
Note that when you type =, the mouse cursor turns to a pointing finger; this is a hint that the Susser is expecting another key.

Manual Hilight Color (<) lets you manually color in squares. You can select one of 9 colors, and color them in by selecting the square and using the set option, or typing , (comma). For example, after setting manual hilight color 8, and coloring in all the squares in the central block, you'd see something like this:



As you'll learn when we get more into hilighting, typing **ESC** will clear the hilighting.

Finally, you can use the **Possibility Filter (;)** option to show hints only on squares that have a particular number of possibilities. For example, if you decide to filter on 4 possibilities (you could type **;**, then **4**), you'd see:



The most common use for the possibiliity filter is to show only the squares that contain two possibilities, as such squares are often part of interesting features.

To turn off the possibility filter, you can type **;**, then **ESC**.

Using Commands

The Commands ButtonBar gives you access to a lot of useful features features:

From left to right, these are **check puzzle validity**, **make forced moves**, **get a clue**, **deduce a move**, and **solve by recursion**.

If you click **check puzzle validity** (or type **v**) the Susser will tell you that everything's OK, but that you can add possibilities to R1C1. This just means that you've removed the 8, but the Susser doesn't want to let it slip that this was correct, so it waffles a bit.

Make forced moves is a very useful command. Click it, or type **m**, and it'll make all the currently available forced moves in the puzzle. Do so now, and 4 moves will be made for you...

15	58	7	3	6	2	9	4	48
68	4	3	78	78	9	5	1	28
568	25 68	589	45 78	1	578	67	346	23 48
3	1	5	257	79	57	4	8	6
9	678	4	178	78	136 78	2	3	5
2	568	58	458	48	35 68	1	9	7
145 78	35 78	158	178	2	178	6	456	149
145	9	2	6	3	1	8	7	14
178	78	6	9	5	4	3	2	1

Making the 4 forced moves caused 6 more forces to become visible!

If you look carefully at the make forced moves button, you'll notice that the keyboard shortcut hint is a lowercase **m**. This means that you can hold down shift to get additional functions. When you hold down **Shift**, you'll see that the icon gets bigger and the **m** becomes and **M**.

Shift-clicking the make forced moves button, or typing **Shift-m**, means "keep on making forced moves until there aren't any more forced moves to make".

Try it now! In a flurry of forces, the entire puzzle will solve (it's a simple one!). Grab the undo slider and you'll be able to back up through the groups of forces.

Before going on, use the undo slider to go all the way back to the start (all the way left), then press Shift-m to solve the puzzle by making multiple groups of forced moves, then move the undo slider to about the middle. This makes sure you'll be able to find some of the things I explain later on by wiggling the undo slider a bit.

Get a clue (g) will give you a clue as to how to proceed. If you click it now, you'll be told:

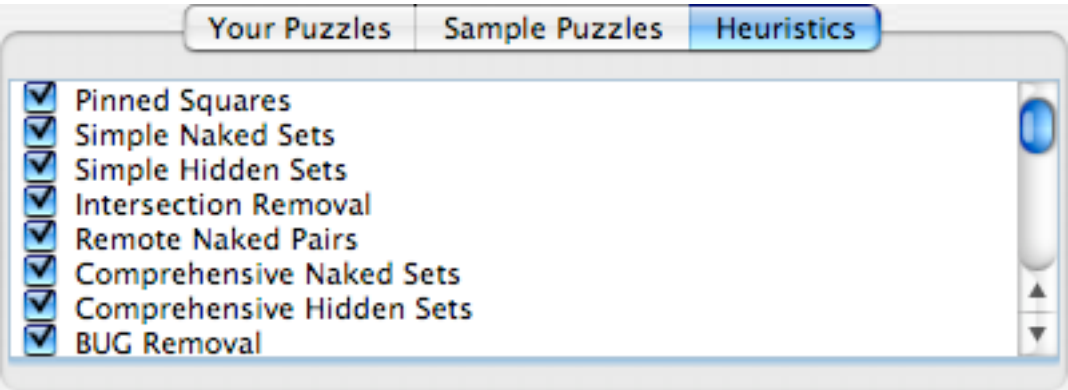
```
You'll be forced to agree that one of the squares can have but a single
solution.
```

As you can probably guess by looking at the icon, if you shift-click the icon, or type **Shift-g**, you'll get a bigger hint, perhaps something like this:

```
You'll be forced to agree that one of the squares can have but a single
solution. I won't say where it is, but it has to be a <4>.
```

Deduce (d) will make the next move in the solution by a process of logical deduction. The Susser has an arsenal of deductive techniques or "heuristics" that it can apply; these are described in detail later in the manual.

If you click on the ***Heuristics*** tab in the puzzles list, you can see and change what heuristics the deducer and clue-giver will use.



You can simply check or uncheck the heuristics you want (or don't want), or shift- or option-click to select a single heuristic and turn off the others, or double-click to turn them all on.

As you have certainly guessed, shift-clicking **Deduce** or typing **Shift-d** will deduce as many moves as possible using the currently selected heuristics.

Trick: holding down* option *while deducing or getting a clue will override the heuristics list selections and use all the heuristics.

The final command button is **solve by recursion** (r). This solves the puzzle by robotic brute-force trial and error. Boring, but very fast.

Hilighting Puzzle Features

Hilighting is one of the Susser's most useful features. It helps you see features in the puzzle that help you solve it.

All Susser hilights are *dynamic*. As you make changes to the puzzle, the hilights automatically update for you. In some cases, they even update as you change the selected square.

The simplest hilights are possibility hilights. To hilight all the squares that can contain a possibility, on the Mac, type **option-#** (so, to hilight the 8's, type **option-8**). On Windows and Linux, type **WindowsKey-#**.

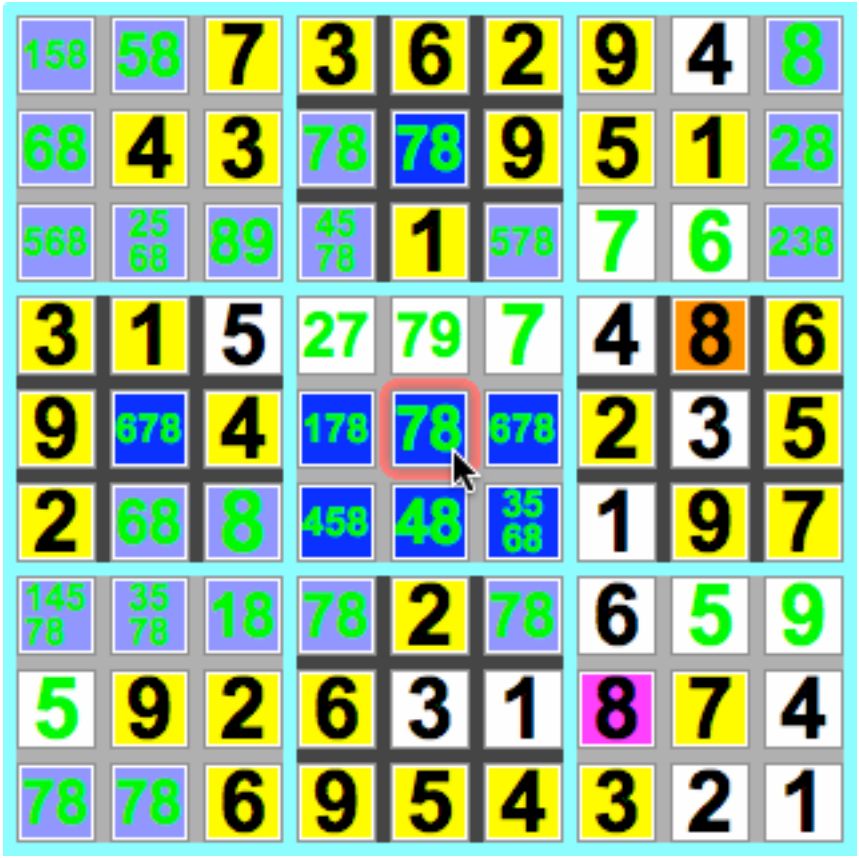
You can also click on one of the row or column labels to hilight the associated number, or type **+**, then the **number**.

If you hilighted the 8's, you might see:

All the unsolved squares that can contain 8 are hilighted in a light blue-purple. The initial squares that were 8 are hilighted in orange, and the solved squares that were 8 are in purple.

Also, when you selected the hilighter, the info panel switched to the ***Hilight Details*** pane, giving you extra info about the hilight you are using.

Now use the mouse or the keyboard to change the square selection and you'll see something interesting.



Whenever there is a selected square, all of its "**buddies**" (the other squares in its row, column or block) that can be the hilighted number are hilighted in dark blue!

Now for extra fun, grab the undo slider and zip it back and forth. As the puzzle changes, the hilights update. Finally, use the keyboard to select a square in the puzzle, and the mouse to wiggle the undo slider.

Secret: if you hold down* Control *while hilighting a possibility, the row and column labels will change to the number of the hilighted possibility in the row/column.

To cancel hilighting, press the **ESC** key in the top-left corner of your keyboard, or reclick the label you clicked to activate hilighting.

If you think this is cool, you ain't seen nothing yet!

Pattern Hilighters

In addition to simply hilighting possibilities, the Susser can hilight patterns. The ***patterns buttonbar*** contains 15 different patterns you can use:

From top-left to bottom-right, they are: **forces, pins, naked sets, hidden sets, intersections, BUGs, unique rectangles and loops, fishy patterns, xy-wings, xyz-wings, forcing loops, forcing chains, nishio, conjugate pairs** and **consequence trees**. Each has a keyboard shortcut. The final button reminds you that you can press **ESC** (or click the button) to cancel hilighting; you can also reclick a hilighter button to toggle it off.

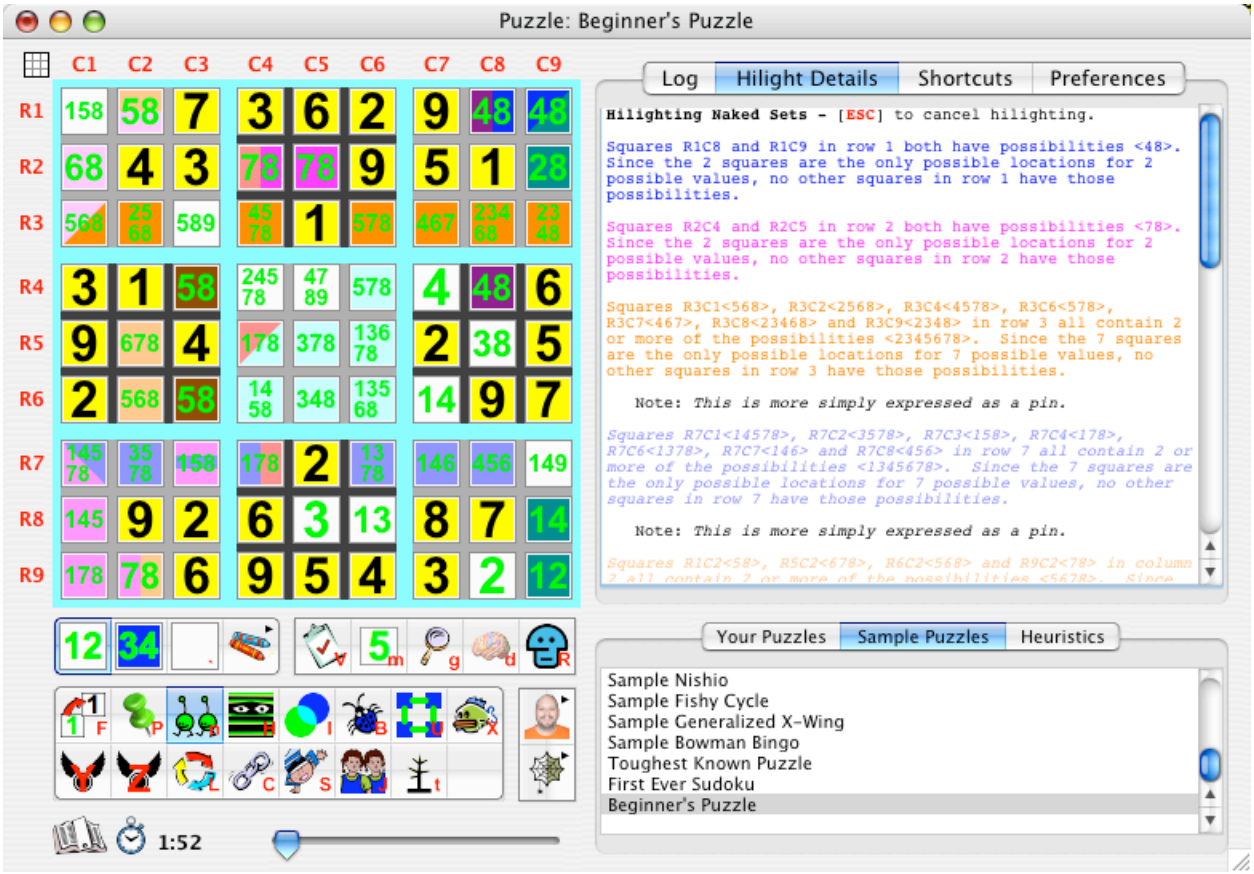
The latter part of the manual has detailed explanations of all the patterns and their implications. For now, let's look at some of the simpler hilighters.

Click on the top-left button (**forces**) or type **f** to activate it. All the current forces (unsolved squares with only one possible value) will be hilighted in blue. Wiggle the undo slider back and forth and watch the hilighting change.

Also note that the hilight details pane of the info panel updates as well, with text descriptions of the forces.

Next, click on the **pin** button, or type **p** to activate the pin hilighter. Pins are squares that are the only square in a row, column or block that can contain a value. Wiggle the undo slider and you'll see them change as well. Reading the text descriptions will help you understand the pin concept.

Move on to the **naked sets** hilighter (the little green men are a naked pair of aliens!). The simplest case of a naked set is a naked pair, two squares in the same **group** (row, column or block) that have the same two possibilities. As mentioned before, you can exclude these values from the other squares in the group. If you wiggle the undo slider towards the start of the solution (leftwards), you'll see a huge number of these, and you'll see that multiple hilight colors are used to keep track of them. Also, the hilight details display gives you the inside details of each naked set in matching colors! Note that the icon contains a lowercase **n**; you'll learn why later on when you find out about "remote naked pairs".



The reason there are so many naked sets at the start of the puzzle is that a pin can also be expressed as a naked set. If there are 5 unsolved squares in a group, and one of them is a pin, then the other 4 are a naked quad!

If you wiggle around a bit, you'll see a much more reasonable display with only a few naked pairs in it.

Some of the more advanced patterns can also be found in the beginners puzzle (though of course, you don't need them to solve it!). In particular, try **Unique rectangles and loops, fishy patterns (X), and XYZ-Wings.**

As you can see, some of the advanced hilighters not only hilight squares, but draw links connecting squares that form logical patterns!

For example, at a certain point in the puzzle, a "type-1 unique rectangle" can be found...

But that's not all. When you find the unique rectangle, move the selection around the puzzle. When it hits R7C6, the display will update and display a "unique loop".

Whenever appropriate, when trying to display a pattern, the Susser looks for an example that contains the currently selected square (if any). If no square is selected, or the square is not part of any instance of the feature, it looks elsewhere in the puzzle.

The feature that is displayed if no square is selected is the one that will be found and used by the deducer, if it is invoked.

Changing Puzzles

By now you are probably pretty bored with the "Beginner's Puzzle". To load a different puzzle, click on the **Sample Puzzles** tab and scroll to the topmost puzzle (Menneske.no Super Easy #878703).

To load the puzzle, double-click it, or select it (single-click) and press **Enter**. Since you've made some progress on your Beginner's Puzzle, you'll be asked if you want to throw away those changes.

Follow the advice in the dialog; click **Cancel** and use the **File » Add Current Puzzle to Puzzle List** (**cmd-L**) menu option. You will see that a copy of the Beginner's Puzzle now appears under the **Your Puzzles** tab. We'll get back to that later.

Click back to Sample Puzzles, and change to the Menneske.no Super Easy #878703 puzzle (double-click or click + enter). This time, because you just saved your work on the Beginner's Puzzle, the Susser will just switch to the new puzzle.

If you left the unique rectangles and loops hilighter running, you'll see there is one in this new puzzle!

Take some time and try and solve the puzzle. When you are ready to continue, keep on reading.

Managing Your Puzzles

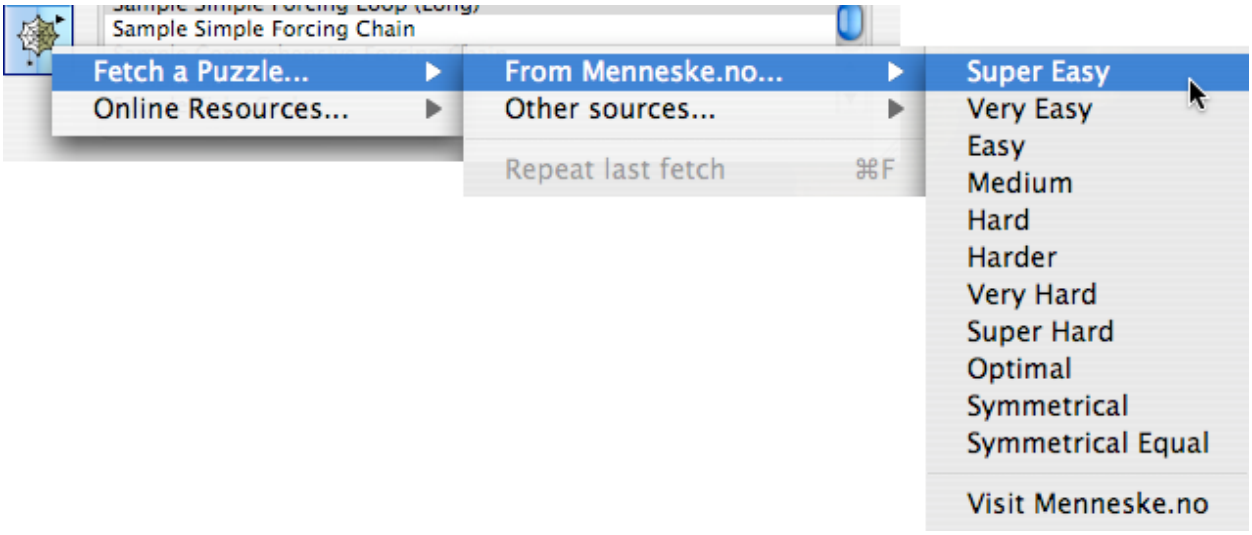
Your personal puzzles are stored under the **Your Puzzles** tab. At any time, when working on a puzzle, you can add it to your puzzles by using **File » Add Current Puzzle to Puzzle List**. If you try to add a puzzle that's already there, it'll be added as a copy. To update a puzzle that you have summoned from your puzzles list (via double-click or click+enter), use File » Update Current Puzzle in Puzzle List (**cmd-U**).

There are many other things you can do with the puzzles list:

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	**Options » Sort Puzzles in Puzzles list** will sort your puzzles. Hold down the shift key to sort in descending order.

Getting New Puzzles

The Susser can fetch new puzzles from a variety of sources on the internet. Click on the web button for a popup menu of selections...



The **Repeat Last Fetch** (**cmd-F**)option will repeat whatever puzzle fetch you last used, as long as it makes sense (fetching a daily sudoku won't activate it, for example). It's also available in the **File** menu. Also, if you hold down **Shift** while clicking the web icon, the Susser will download 10 puzzles at a time, if appropriate!

All of these puzzles will be placed in your puzzles list.

Also available in the web popup are online resources such as newspapers, other online puzzle sources, and information about Sudoku, including some great discussion forums. Selecting one of these will launch your web-browser.

Dragging Puzzles into the Susser

One of the Susser's cuter features is that you can drag (or cut&paste) any text or graphic puzzle from another application or a web-page and the Susser will try and recognize it!

No kidding, you can actually drag puzzles right off a newspaper's webpage and into the Susser. It will even usually recognize graphics containing multiple puzzles.

Same goes with files containing graphics or text puzzles, just drag them in.

What it usually can't do is handle javascript or flash-based puzzles, because web-browsers don't let you drag graphics off them. But if you have a screen-capture utility (such as SnapNDrag on the Mac), you can just clip an image of the puzzle and drag that into the Susser.

Graphic Puzzle Import Options

These rather geeky options can help when trying to load a difficult puzzle, but 99% of the time, you won't need them.

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 the standard "stop it already" key (**cmd-.** on the Mac), until the Susser notices and stops.

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...

Entering a Puzzle by hand

To enter a puzzle by hand, select **Options** » **Clear Current Puzzle** to blank out the puzzle, then fill in the initial squares as if you were solving them. Then select **Options** » **Set Initial Squares** to make the solved squares into yellow initial squares, and finally **File** » **Add Current Puzzle to Puzzle List** to save it.

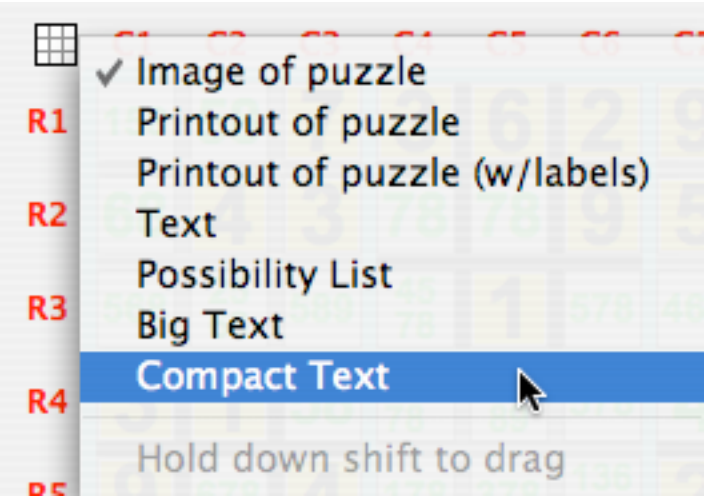
If you make a mistake, you can use **Options** » **Clear Initial Squares** to change the yellow squares back to white, make changes, and proceed as before.

Unlocking Book Puzzles

If you have bought either of my Sudoku books published by Peter Pauper Press, then **Options** » **Peter Pauper Puzzles...** will bring up a dialog that lets you enter the ISBN number of either book (above the bar code). This will add all the puzzles in the book(s) to the Sample Puzzles list.

Dragging Puzzles out of the Susser

You can drag puzzles out of the Susser into other applications. Clicking on the small **drag square** in the top-left corner of the Susser window will pop up a menu of drag options...



These let you select the format of the drag:

Image of puzzle drags an image of the puzzle as it currently appears in the Susser window (most of the puzzle examples in this manual were dragged this way).

Printout of puzzle drags a large image that looks just like the printout the Susser will give you. It is monochrome, with smaller hint digits.

Printout of puzzle (w/labels) adds puzzle name labels to the printout.

Text drags a simple but attractive text version of the puzzle, convenient for emailing.

Possibility List drags a text version that shows the possibilities for each square. This format is commonly used on Sudoku discussion forums.

Big Text is kind of geeky. Try it and you'll see.

Compact Text is a very simple text version that can often be loaded into other Sudoku programs, such as Andrew Gregory's Sudoku for the Palm app.

To actually drag the puzzle, hold down shift and drag the drag square. You can also drag puzzles right out of the puzzles list (though the image style isn't available there; you'll get a printout instead).

The File Menu

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. The Susser will use its text/graphic recognition system to try and figure out the puzzle.

Repeat Last Fetch will repeat the last internet puzzle fetch you did; hold down **Shift** to fetch a block of 10 puzzles.

Load and Rate Puzzles... is a true geek option; it is described towards the end of this document.

Save Puzzles and Preferences will save your current puzzle, the puzzles in your puzzles 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.

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

Save Current Puzzle as Graphic… lets you save your current puzzle as a PICT (Mac) or BMP (Windows/Linux) graphic. The graphic is a resolution-reduced version of the printed version of the puzzle (see below)

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.

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.

Page Setup and **File » Print** work as you would expect, and print the current puzzle in a format suitable for paper and pencil solving. If you want the puzzle printed smaller, so you have more space for notes, check **Half-sized Puzzle Printouts** before printing.

The Edit Menu

The Edit Menu contains all the standard menu items you expect, and they work as you expect them to. **Cut, Copy**, **Paste** and **Clear** work with text (such as in the Log pane, or when changing a puzzle name).

There are two extra options, **Bookmark** and **Return to Bookmark**. See the bookmark documentation earlier in the manual for detail on how to use them.

The Options Menu

In addition to the options mentioned a few pages ago, there are some other options that 99% of Susser users won't need to worry about.

Set Unsolved Squares to have all possibilities sets each square to have all the possibilities from 1 through 9, while **Set Unsolved Squares to have no possibilities** clears all the possibilities.

This lets people who like to do the puzzles with as little help as possible start with a blank slate and add or remove the possibilities by hand.

Once they get tired of this, they can use **Recompute possibilities on Unsolved Squares** to get the Susser to figure things out for them.

The truly masochistic will appreciate **Marquis de Sade Manual Masochism Mode**. When checked, the Susser will not even remove possibilities from a square's buddies when you solve a square. You will have total control of the puzzle - and your expensive computer will emulate a #2 pencil.

In MdSMM Mode, PgUp (or space) will set a square to have all possibilities, and PgDn will set it to have none.

The Trebor's Tables Menu

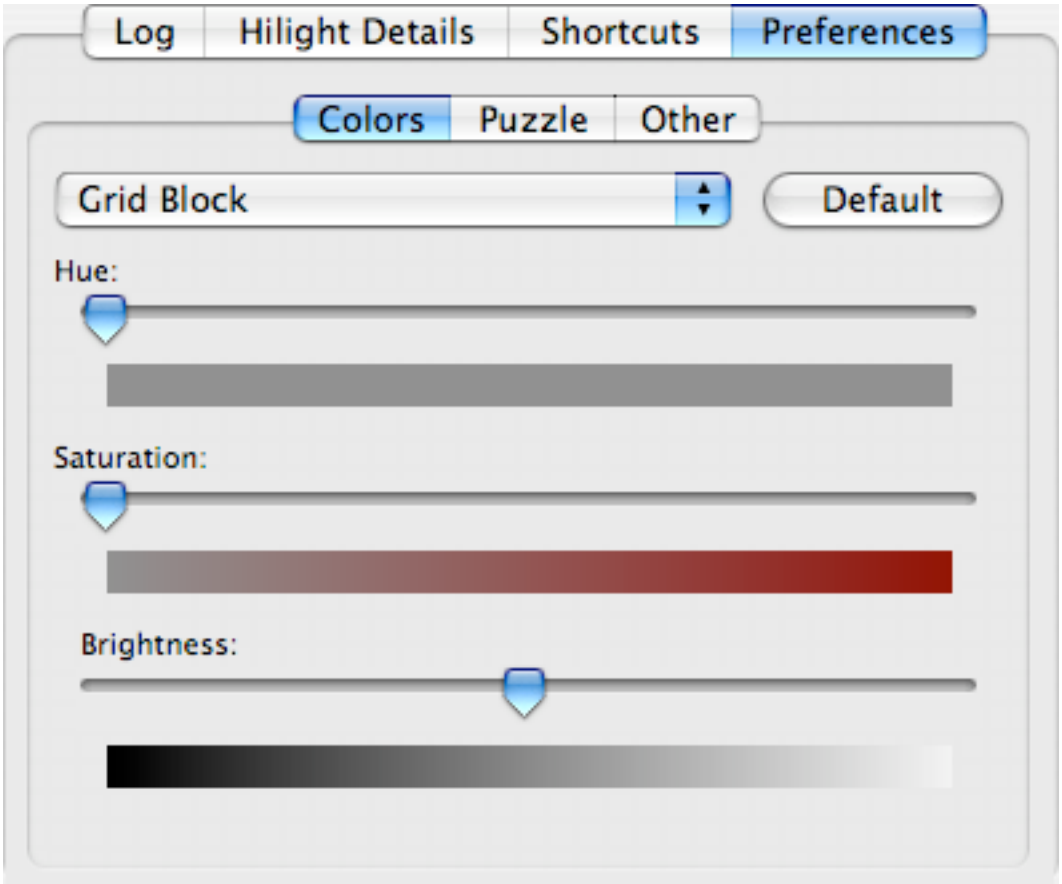
Trebor's Tables is an extremely advanced solving technique. See the section in the manual on tabling for more details.

The Font Menu

Lets you change the font and size of the text in the Log and Hilight Details panes of the info panel.

Preferences

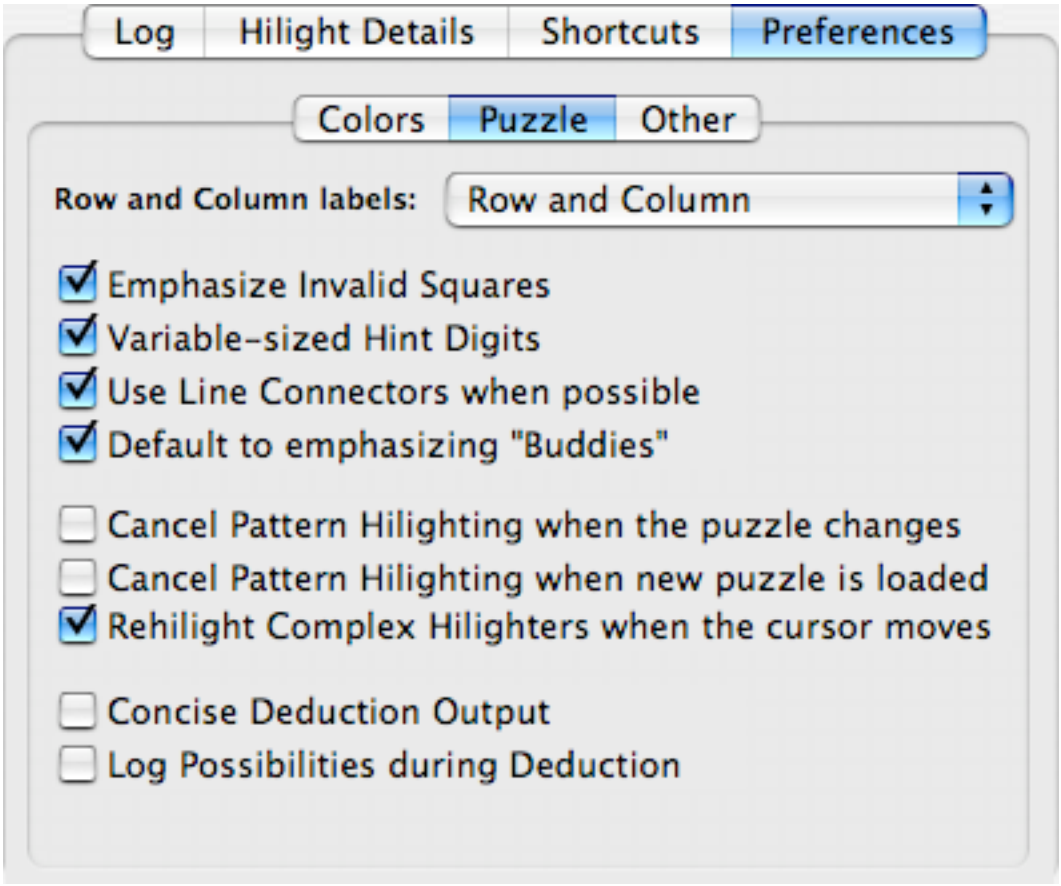
Select the preferences menu item, or click on the Preferences tab in the Info panel, to change your preferences. The Preferences tab has 3 subtabs, ***Colors***, ***Puzzle***, and ***Other***.



The **Colors** subtab lets you adjust the color of the puzzle background, lines, grid blocks, solved squares, etc., to fit your taste and the quirks of your perceptual system (in other words, if you're color blind, you'll want to play with it!). Use the popup menu to select the element you want to change, then wiggle the sliders to adjust the color. As you move the sliders, the color bars under them will change to reflect your change, and show you what other changes are possible. When you release a slider, the window will update to reflect your change. Click the **Default** button to return to my original garish color scheme.

Color options marked with (g) are also used in the popup hieroglyph, so you might want to have it active when you play with them.

The **Puzzle** subtab lets you customize the way the puzzle behaves.



Row and Column labels let you select how the rows and colums are labled, either **Row and Column** (R1C2), **XY (Top-left origin)** or **XY (Bottom-left origin)**.

Emphasize Invalid Squares displays invalid squares in the puzzle with a big black on red X; if unchecked, the square is just blank.

Variable-sized Hint Digits zooms the hint digits to fit the square (so a square with fewer possibilities has larger digits). If unchecked, the hint digits are all the same size, and each possibility is in the same place in each square.

Use Line Connectors when possible replaces colored hilight squares with line connectors whenever possible. Typing \ will toggle this option. This is particularly useful when looking for remote naked pairs.

Default to emphasizing "Buddies" affects the possibility hilighters. If checked, then buddies of the currently selected square that contain the hilighted possibility are colored differently from non-buddies. If unchecked, then all the unsolved squares are colored the same. Whatever you choose, you can select the other behavior by holding down **Shift** when activating a possibility hilighter (via the row/column labels or keyboard)

Cancel Pattern Hilighting when the puzzle changes / when new puzzle is loaded cancel hilighting in those circumstances. If you're the kind of person who only occasionally uses pattern hilights for a clue, then this prevents you from inadvertantly getting a clue after you make a move.

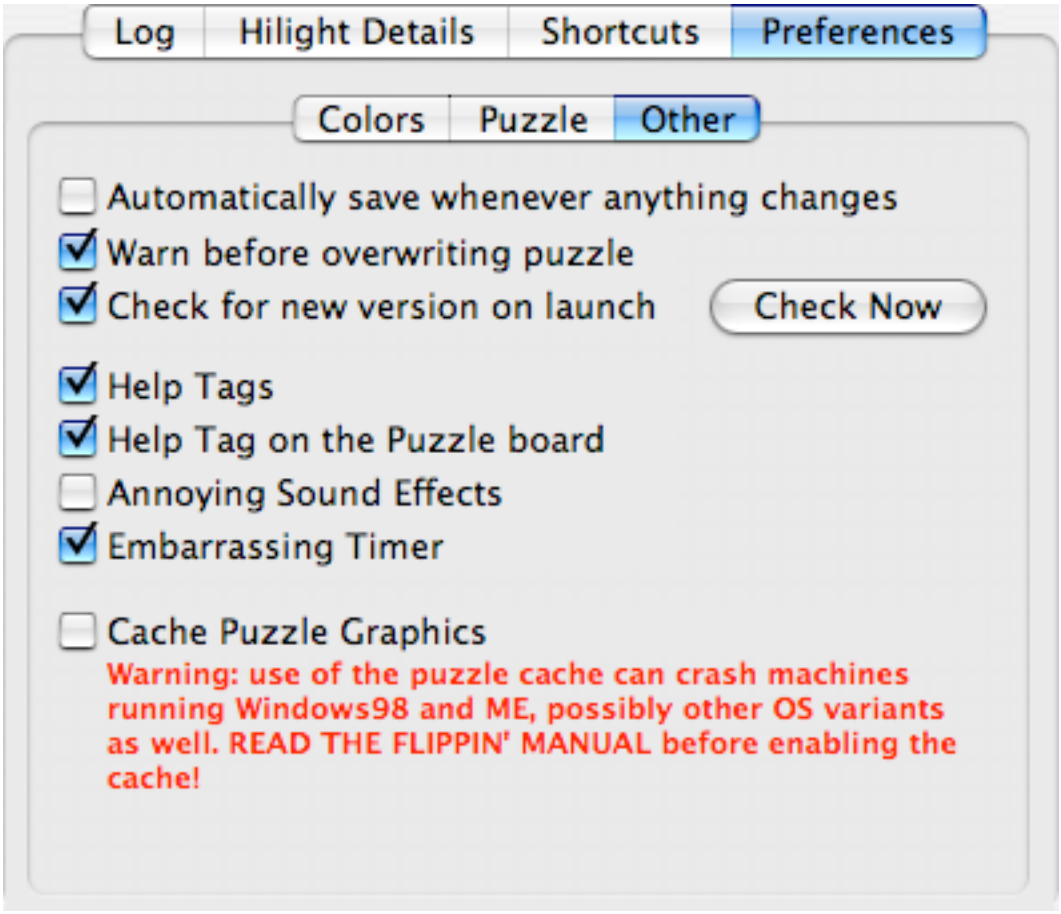
Rehilight complex hilighters when the cursor moves controls whether the complex (potentially time-consuming) hilighters recompute themselves when you move the cursor around the puzzle (trying to find a pattern example using the current square). On slower computers, you may wish to uncheck this option.

Concise Deduction Output reduces the amount of explanation provided by the deducer when it applies certain rules, such as Nishio and Bowman Bingo. Only experts will want to check it.

Log Possibilities during Deduction prefaces each deduction report with a "possibility list" representation of the puzzle. These are mostly useful for including in posts in Sudoku discussion forums.

You can also print the possibility-list of the puzzle to the log by pressing | (Shift-\) at any time.

The **Other** subtab lets you tweak miscellaneous program options.



Automatically save whenever anything changes updates your *sudokus.txt* file anytime it is needed. This option is particularly nice if you find a bug in the program that causes a crash. You can set this option, run the program until it crashes, and send me the sudokus.txt file along with info about the very last thing you did!

Warn before overwriting puzzle warns you, and gives you a chance to change your mind, anytime you do anything that will cause a loss of information (like loading a new puzzle when you've made unsaved progress on your current puzzle).

Check for new version on launch queries my webserver to find out if there is a new version of the Susser. No personal information is transmitted - all I ever know is that "ip address 1.2.3.4 launched the Susser". You can also click the **Check Now** button to check for a new version and display the current version's change notes.

Help Tags enables the yellow helptag "post-it notes" on all the various features in the Susser. Help tags are always enabled when you're in the preferences.

Help Tag on Puzzle board enables a helptag on the puzzle board that tells you information about the current square.

You can also print the current square's info to the log by pressing ?

Annoying Sound Effects allows you to disable my wonderful sound effects.

Embarrassing Timer lets you show and hide the puzzle timer. Clicking on the stopwatch icon to the left of the timer does the same thing.

You can also pause/restart the timer by clicking on the actual time, and reset it by shift-clicking.

Finally, **Cache Puzzle Graphics**, when enabled, computes a bunch of puzzle graphics in the background to speed up puzzle display a bit. Unfortunately, some people, notably those using Windows98/ME with relatively little memory, will run into problems with this feature - there seems to be a deep RealBasic bug that causes the app to use up all the system resources.

Here is how to test to see whether the option works for you.

1) Resize the window to the largest size you are likely to use (bigger window, bigger squares, so bigger cache requirements).

2) Use **File » Save Puzzles and Preferences** to save your state.

3) Go to **Preferences » Other** and check **Cache Puzzle Graphics**.

4) Wait a bit. Once the cache has finished computing, a message will appear in the log pane telling you everything is OK.

5) If the app crashes, then you can't use the cache; simply restart and, since you saved everything just before the test, everything will be OK. Just don't use the cache.

If you get a "running out of system resources" type of message but the app does not crash, then quit the app **but do not save your changes**. Restart and make sure that **Cache Puzzle Graphics** is unchecked.

The Popup Hieroglyphic

The Popup Hieroglyphic is an experiment feature of the app. As it is currently a bit inefficient in the way it displays itself, I currently recommend it only for people with faster computers.

Click on any square, and the glyph will appear, anchored to one of the corners of the square; non-selected squares will be slightly greyed.

To set a square, click one of the values in the horizontal bar of the glyph; the square will solve. To toggle possibilities in a square (off if on, on if off), click one or more of the values in the vertical bar. To reset the possibilities back to the values constrained by the current solved squares, select the R in the ball.

When the glyph is active, you will no longer be able to move the hilight around the puzzle (either by mouse or keyboard). But you will be able to do things like activate hilighters, and so on.

The glyph will dismiss itself if you solve the square, reduce it from two to one possibility (which will solve it) , or reset the possibilities. It will also go away if you deduce, recurse, or in general do anything that solves squares (you can use the keyboard to add-remove possibilities, and the glyph will update.

You can also dismiss it by clicking outside the glyph, or pressing ESC.

If possibility hints are being displayed for a square, then both the horizontal and vertical bars will display currently possible values in green, and invalid values in red. If hints are not being displayed, then the vertical bar will not be present, and all the values will be green (ie: you don't get hints!)

The hieroglyphic is my attempt at some clever user interface design. It takes up much less space than a regular popup menu, and obscures very little of the puzzle when displayed. Your comments are solicited.

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.

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. It is equivalent to **Make Forced Moves**.

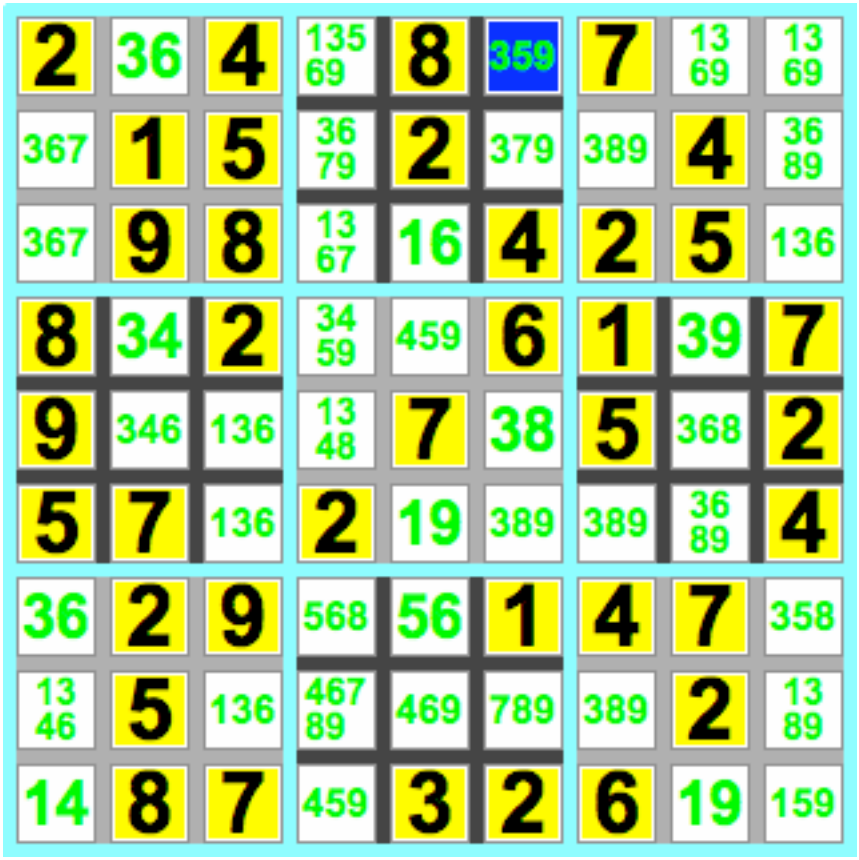
In this simple puzzle, you can immediately see that there are many squares that can have only a single value.

Type **f** to hilight all forced moves, **m** to make all currently available forced moves (**M** to keep on making them until there are no more left).

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>.



Type **p** to see the pins in a puzzle.

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

Important reminder:

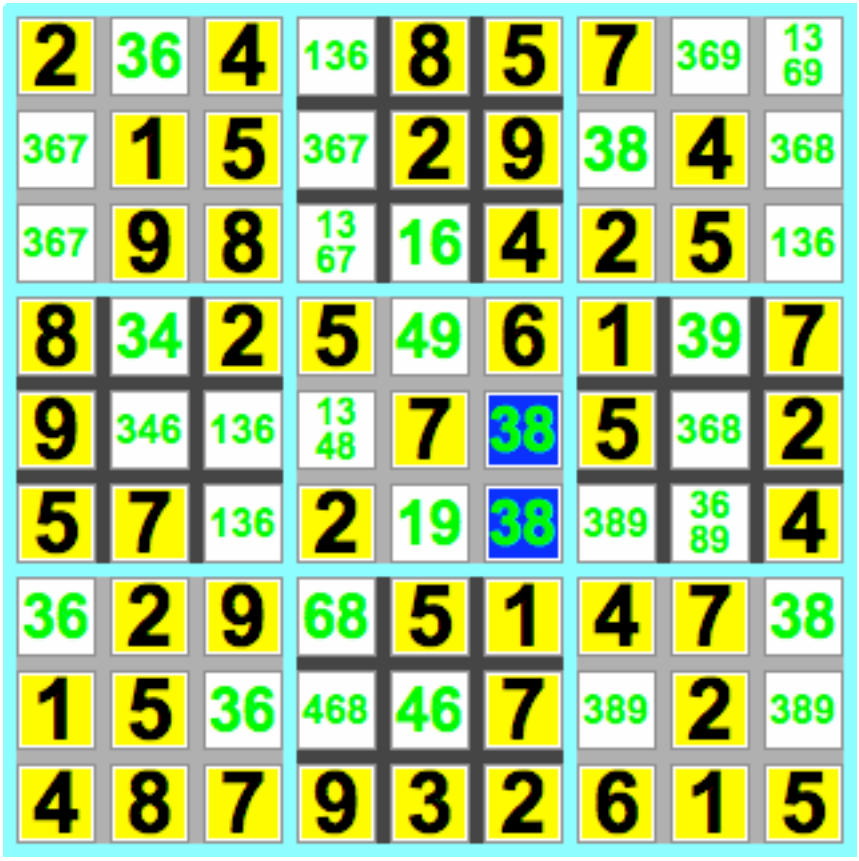
After any hilighting, if you mouse over a square and have **Help Tag on Puzzle** checked (or if you type **?**), you'll be told the reason why the square is hilighted. This is particularly handy when using some of the more complex hilighting features!

Simple Naked 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 "naked pair"; a set of three squares with the same three possibilities would be a "naked triplet", and so on.



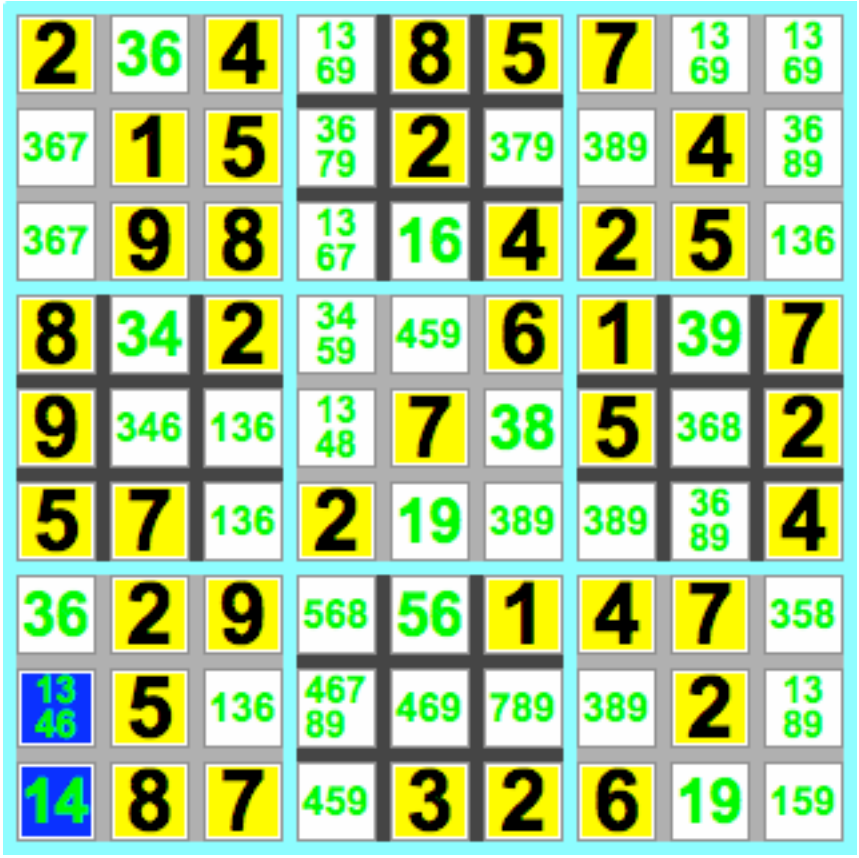
Type **n** to see the *useful* naked sets in a puzzle.

In the "Sample Naked Pair" puzzle, 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 naked pair in the bottom-left block, but since all the other squares are solved, knowing this does not advance the solution - so they are not hilighted. If you type N to invoke the hilighter, you'll also see the non-useful sets.

Simple Hidden Sets

Simple hidden sets are the inverse of simple naked sets. If you can find *n* squares, each of which has *n* possibilities that no other squares in the group can contain, then any extra possibilities those squares have can be removed. This always generates a simple naked set, thus the name: the set is "hidden" inside the possibilities.

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



Type **h** to expose the hidden sets in a puzzle.

In the "Sample Simple Hidden Set" puzzle, 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 naked pair on <14> in column 1 and in block 7 (which permits further progress).

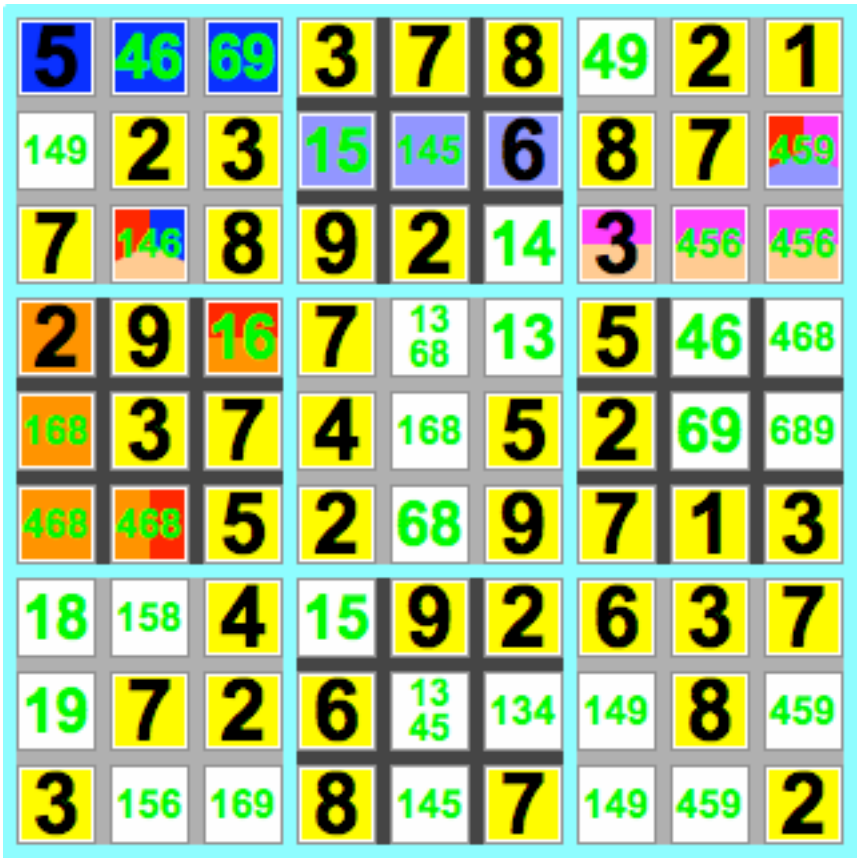
Note that hidden triples or even quads are possible, though rare.

Intersection Removal

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

It should also be clear that if the squares in a block with a particular possibility only intersect one column or row, the same rule applies.

In the "Sample Intersection" puzzle (below), the value <6> can only appear in squares R1C2 and R1C3 of row 1, and R1C1 is solved. Since the <6> must be in R1C2 or R1C3, the non-intersecting squares of block 1 cannot contain it, so <6> can be removed from R3C2. Note also the other intersections in the puzzle.



Type **i** to see the intersections in a puzzle.

In intersection hilighting, red indicates a square that can be reduced; the other colors in the squares indicate which intersections permit the reduction. Non-red squares are the actual intersections.

Remote Naked Pairs

If two squares A and B in a group have the same two possibilities XY, they form a "naked pair"; one will be X and the other Y.

If square B also forms a naked pair in another group with square C, then A and C are “complementary pairs”; whatever A is, C must be the same.

If C in turn forms a naked pair with D in yet another block, then A and D form a "remote naked pair"; if A is X, D must be Y, and vice versa.

Thus, any square that is a buddy of both A and D cannot contain X or Y!

Type \ to toggle Line Connectors on, then N to see all the naked sets, plus all the naked pairs, useful or not.

In the “Sample Remote Naked Pair” puzzle (shown above), squares R1C5 and R4C9 form a remote naked pair, and 9 can be removed from R4C5, which is in column 5 (the column of R1C5 and the row 4 (the row of R4C9.

In the "Sample Remote Naked Pair #2" puzzle (shown above), squares R1C1 and R3C4 form the remote naked pair, and reductions can be made in 3 squares; R3C2 and R3C3 are in R1C1's block and R3C4's row, and R1C5 is in R1C1's row and R3C4's block.

The thing to keep in mind with remote naked pairs is that if the endpoints are in the same "row of blocks" or "column of blocks" (as in the second example; the two endpoints are in the top row of 3 blocks), then they can intersect at 6 possible squares. Otherwise, as in the first example, they can only intersect at 2 (the corners).

Comprehensive Naked Sets

Comprehensive naked sets are a superset of simple naked sets; a bit harder to see, but much more useful.

Look for a set of *n* squares in a group which have, between them, exactly *n* possibilities. Then each of those *n* possibilities must be in one 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 naked triple on <123>.

Similarly, <123>,<14>,<1234> and <23> form a naked quad on <1234>.

Simple naked sets, such as <123>,<123>,<123> are merely a special case of the more general rule.

Typing **n** also shows you Comprehensive Naked Sets.

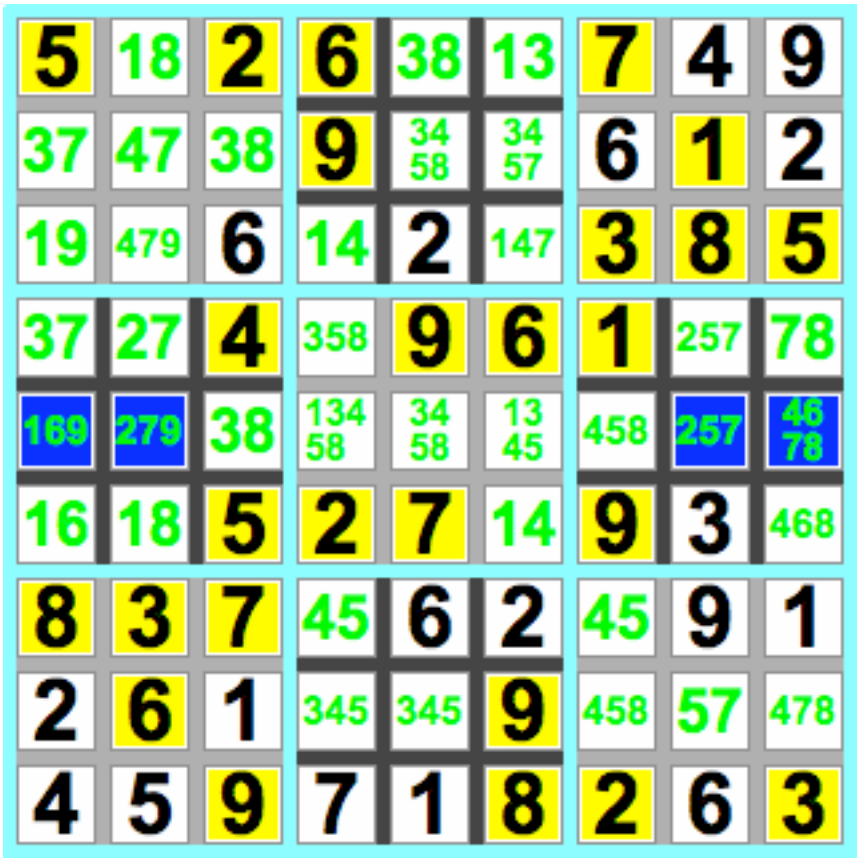
In the "Sample Naked Set" puzzle, you can find a comprehensive naked triplet consisting of squares R2C1, R2C3 and R3C2 in Block 1.

Comprehensive Hidden Sets

Comprehensive hidden sets are a superset of simple hidden sets, and now that you know about comprehensive naked sets, the definition should be obvious:

If you can find *n* squares that, between them, contain *n* possibilities that no other squares in the the group can contain, then any extra possibilities those squares have can be removed. This always generates a comprehensive naked set.

Simple hidden sets are, of course, a subset of comprehensive hidden sets.



Typing **h** also shows you Comprehensive Hidden Sets.

In the "Sample Hidden Quad" puzzle, Squares R5C1, R5C2, R5C8 and R5C9 all contain one or more of the possibilities <2679>. This permits 1 to be removed from R5C1, 5 from R5C8, and 48 from R5C9.

BUG Hunting

A particularly interesting situation sometimes occurs when a puzzle is almost completely solved. If the puzzle reaches a state where all but one of the unsolved squares has 2 values ("bivalue squares") (and thus, one unsolved square has more than two -- the "polyvalue" square), then it is possible to immediately make a reduction in the polyvalue square.

Type **b** to hunt for BUGs.

The way it's done is very cute. Simply look at the possibilities in the polyvalue square, and change it to a bivalue square such that each row, column and block in the puzzle contains each unsolved possibility exactly twice.

Such a configuration is actually illegal, because it will result in a puzzle that does not have a single solution! These configurations are called **Bivalue Universal Graves** (someone clearly loves acronymns!) or **BUG**s. The two remaining possibilities in the polyvalue square are called the **BUG Possibilities**.

Because they result in invalid puzzles, the two BUG possibilities can immediately be eliminated from the polyvalue square, and the puzzle should become easily solveable by forces and pins.

It should be noted that it is possible to construct BUGs from puzzles that have more than one polyvalue. However, it is often very difficult to figure out what inferences can be made from these BUGs without using techniques like forcing chains. Finding simple rules for BUG Hunting in multi-polyvalue puzzles is an ongoing topic of discussion in the Sudoku research community.

It is my hope to be able to report that BUG hunting is a more powerful and easily-usable technique in the near future.

Unique Rectangles

The unique rectangles test takes advantage of the fact that a Sudoku can only have one solution. Consider the following puzzle:

Typing **u** displays unique rectangles.

Note the squares R4C2, R4C3, R7C2 and R7C3; they form a group of 4 squares that share exactly 2 blocks, 2 rows and 2 columns. The important insight here is that <u>if they also only share 2 possibilities, then the puzzle has more than one solution</u>! It's a similar situation to BUGs.

If you think about it, it is obvious: if in the "Sample Unique Rectangle - Type 1" puzzle (shown above), the four squares had possibilities <25>, then two diagonally-opposite squares must be <2>, and the other two must be <5>. No matter which way you arrange them, the rows, columns and blocks would have one 2 and one 5, and you could exchange the 2's and 5's and the puzzle would still be valid -- so it has more than one solution. I call this configuration of 4 squares with the same 2 possibilities in two rows, two columns and two blocks the "**deadly pattern**."

Find it, and you know you've gone wrong. But you can use it to your advantage...

Here's how: If you can find a rectangle such as the one shown above, with 4 squares sharing 2 rows, columns and blocks, 3 of which share the same two possibilities, and the 4th having the two possibilities plus one or more extra possibilities, then you can remove the original two possibilities from the 4th square. In this case, R4C3 can be reduced to <36>.

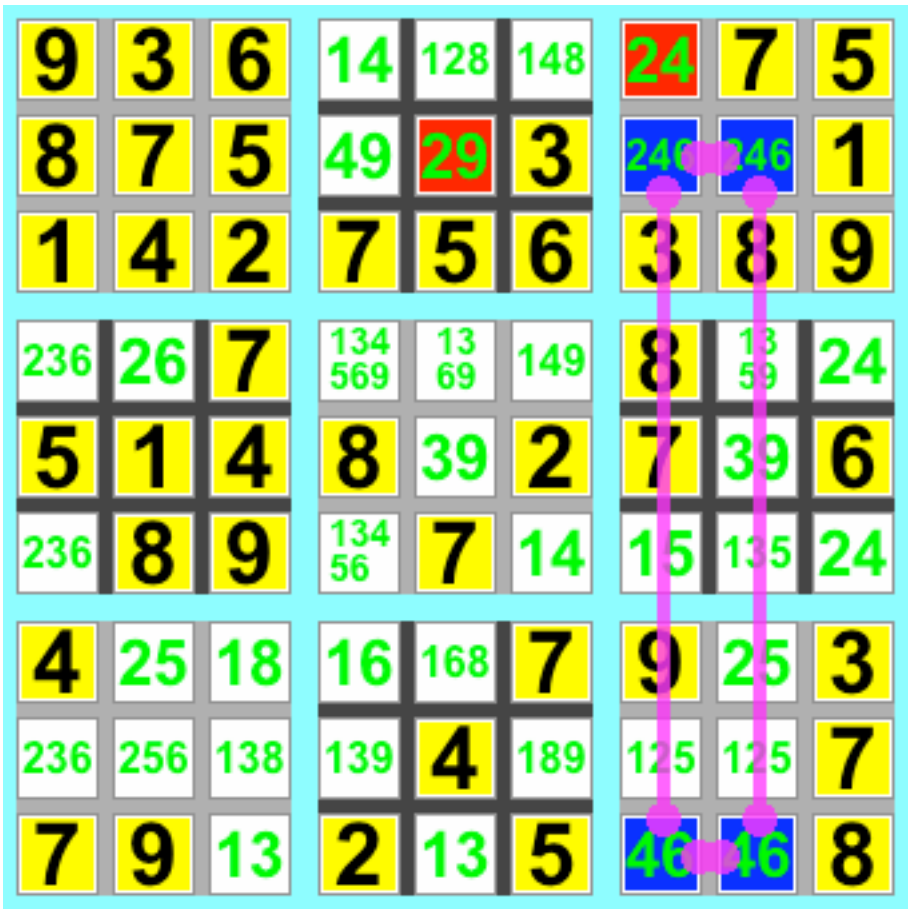
The proof is pretty straightforward once you get your head around the basic idea.

Assume R4C3 is 2. That forces R4C2 to be 5, R7C2 to be 2, and R7C3 to be 5. That's the deadly pattern; you can swap the 2's and 5's and the puzzle still can be filled in. So if the Sudoku is valid, R4C3 cannot be 2.

The exact same logic applies if you assume R4C3 is 5. So R4C3 can't be a 2, and can't be a 5 -- it must be either 3 or 6.

This pattern is called a "Type-1 Unique Rectangle." But it turns out there are several other interesting unique rectangle patterns, all of which depend on having to avoid the deadly pattern.

Consider the puzzle on the next page:



Here we have a similar pattern, but this time, two of the squares *which share the same block* have a single extra possibility - in this case, <2>.

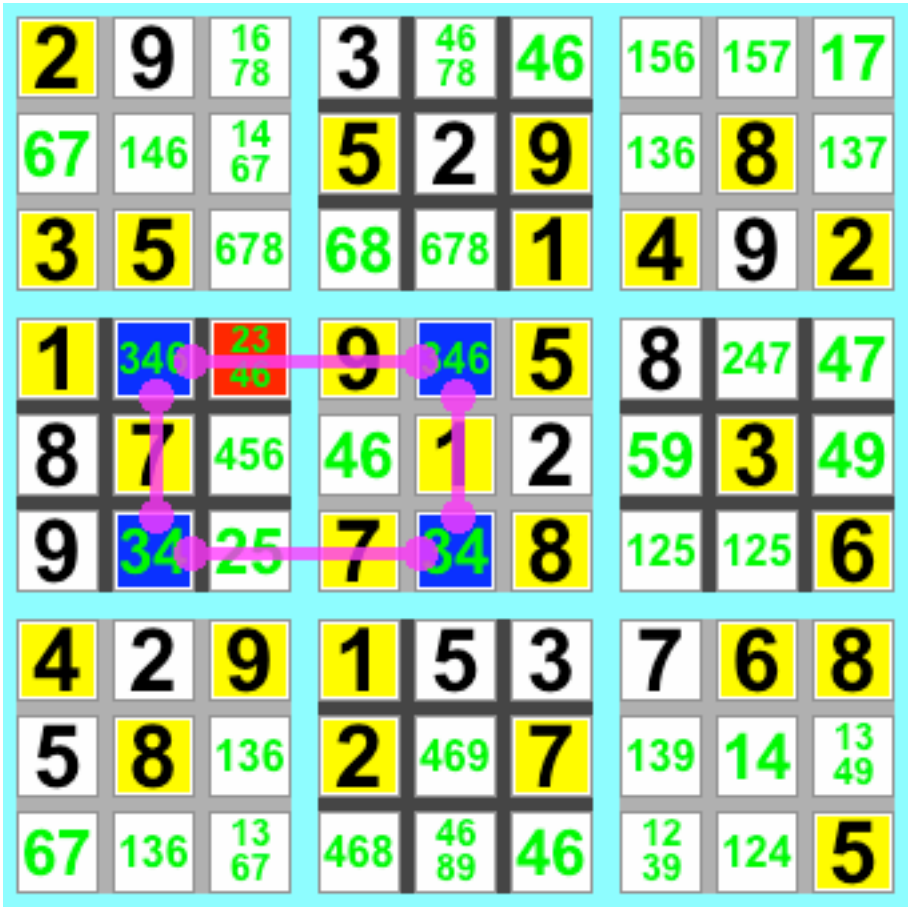
Terminology: the two squares that only have two possibilities are called the floor squares (because they form the foundation of the Unique Rectangle); the other two squares, with extra possibilities are called the roof squares.

In the "Type-2 Unique Rectangle" puzzle, one of the blocks contains the floor squares, and the other contains the roof squares. In order to avoid the deadly pattern, 2 must appear in either R2C7 or R2C8 (the roof squares). Therefore, it can be removed from all other squares in the groups that contain both of the roof squares (in this case, row 2 and block 3).

Now that you've gotten your head around the basic unique rectangle concept, the proof should be pretty obvious:

If neither R2C7 or R2C8 contains a <2>, then they both become squares with possibilities <46>. This results in the deadly pattern - so one of those squares must be the <2>, and none of the other squares in the intersecting groups can contain the 2.

There is a second variant of Type-2 Unique Rectangles:



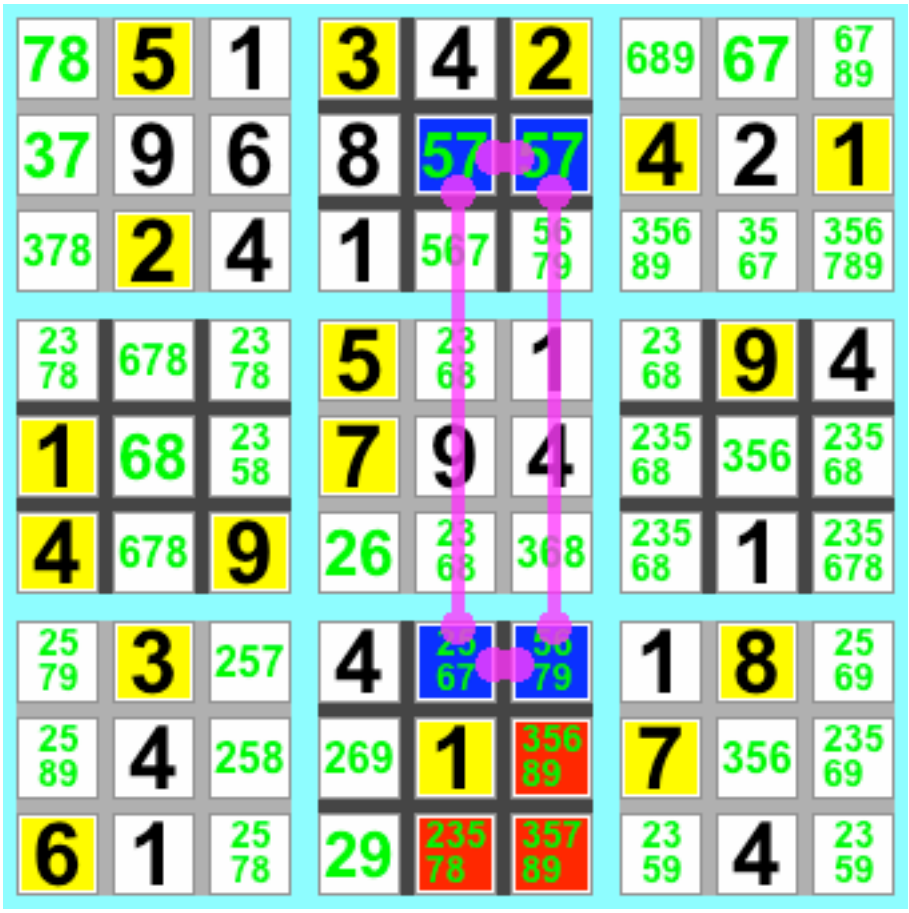
In the “Sample Unique Rectangle - Type 2B” puzzle, we have the same pattern of 4 squares in 2 blocks, 2 rows and 2 columns. However, in this Unique Rectangle, each of the blocks contains one floor and one roof square. This is perfectly fine, but it means that the only group that contains both of the roof squares is row 4, so that is the only group that you can attempt to reduce. This is called a “Type-2B Unique Rectangle”

In the “Sample Unique Rectangle - Type 3” puzzle, the roof squares contain the same two extra possibilities. Squares R7C7 and R7C8 both have possibilities <2369>; the <26> matching the floor squares, plus extra possibilities <39>.

In order to preclude the deadly pattern, at least one of R7C7 and R7C8 has to be a 3 or a 9. We don’t know which square it’s in, or whether it is a 3 or a 9. It’s sort of fuzzy, which reminds me of quantum physics. But what we can do is treat the two squares as a single “quantum square” with possibilities <39>, and use this to find naked sets in their shared groups that permit reductions to be made.

For example, in the puzzle above, R7C7+R7C8 are the quantum square, and it plus R7C9 form a naked pair on 39 in both row 7 and block 9. We know there has to be a 3 or a 9 in R7C7 or R7C8 in order to prevent the deadly pattern; if it is a 3, then R7C9 must be a 9; if it is a 9, then R7C9 must be a 3. Either way, <39> can be excluded from other squares in their common groups. So R7C3, R9C7 and R9C8 can have <3> removed.

It is important to realize that these “Type-3 Unique Rectangles” are not limited to roof squares that share 2 possibilities. In fact, you can treat the roof squares as a single quantum-square containing all the possibilities that are not in the floor squares! Consider the “Sample Unique Rectangle - Type 3 #2” puzzle:



Here, the roof squares (R7C5 and R7C6) contain extra possibilities <26> and <69> respectively. This means they form a quantum-square with possibilities <269>, which forms a naked triple in block 8 with R8C4 and R9C4.

Finally, just as Type-2 Unique Rectangles have a -B variant, so do Type-3’s! And if you think about it, Type-1’s don’t have a -B variant because they actually are both at the same time, depending on which squares you consider to be the floor squares.

Also, a Type-2 is really a Type-3, but instead of a naked set, you've got a "naked single."

Wrapping up our discussion of Type-3 Unique Rectangles, the "Sample Unique Rectangle - Type 3B" puzzle contains an example of a Type-3B:

As with Type-2B's, since the roof squares are not in the same block, you can only look for reductions in row 2. In this case, the quantum square plus R2C6 form a naked set on <28>, and you can remove <8> from R2C2.

Interestingly, you can also look for reductions by looking for hidden sets on the floor possibilities that do not include the extra roof possibilities. For example, in the above example, if you pretend the quantum square is <69>, then you can find a hidden set on <1569> using the quantum square<69>, R2C2<568>, R2C4<16>, and R2C9<15>; this lets you remove the <8> from R2C2.

Typically, the more extra floor possibilities there are, the harder it is to find the naked set, and the easier it is to find the hidden set. The Susser currently only reports the naked set reduction.

In the "Sample Unique Rectangle - Type 4" puzzle, we see yet another interesting inference that can be drawn from Unique Rectangles:

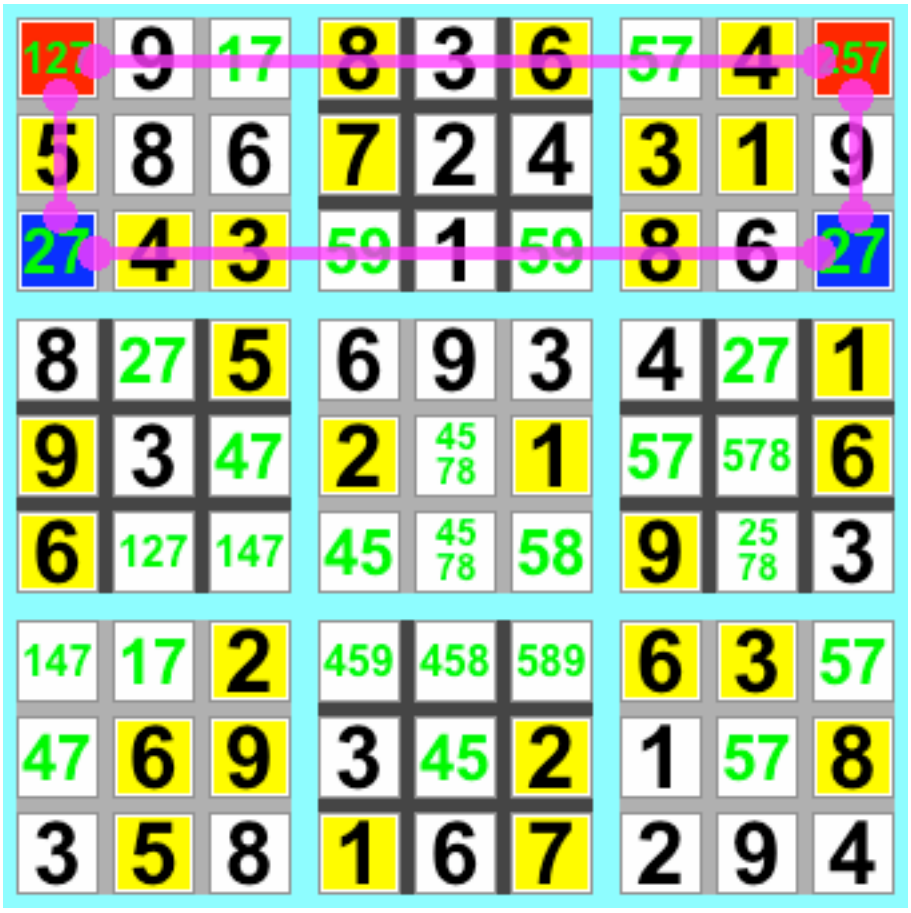
Look closely at the "roof" squares, R7C4 and R7C6, but this time, don't look at their extra possibilities; look at the possibilities they share with the "floor" squares.

If you look carefully, you'll see that in block 8, the roof squares are the only squares that can contain an <9>. This means that, no matter what, one of those squares must be <9> -- and from this you can conclude that neither of the squares can contain a <5>, since this would create the "deadly pattern"! So you can remove <5> from R7C4 and R7C6.

When two squares are the only two squares in a group that can have a particular value, they are referred to as a "conjugate pair" on that value.

This is an example of a “Type-4 Unique Rectangle”. As you have probably realized, since the roof squares are in the same block, you can search for conjugate pairs in both of their common groups (the row and the block, in this case).

And, as you might expect, there is a “Type-4B Unique Rectangle” variant, in which the floor squares are not in the same block, and you can only look for the conjugate pairs in their common row or column. The “Sample Unique Rectangle - Type 4B” puzzle demonstrates this, as follows:



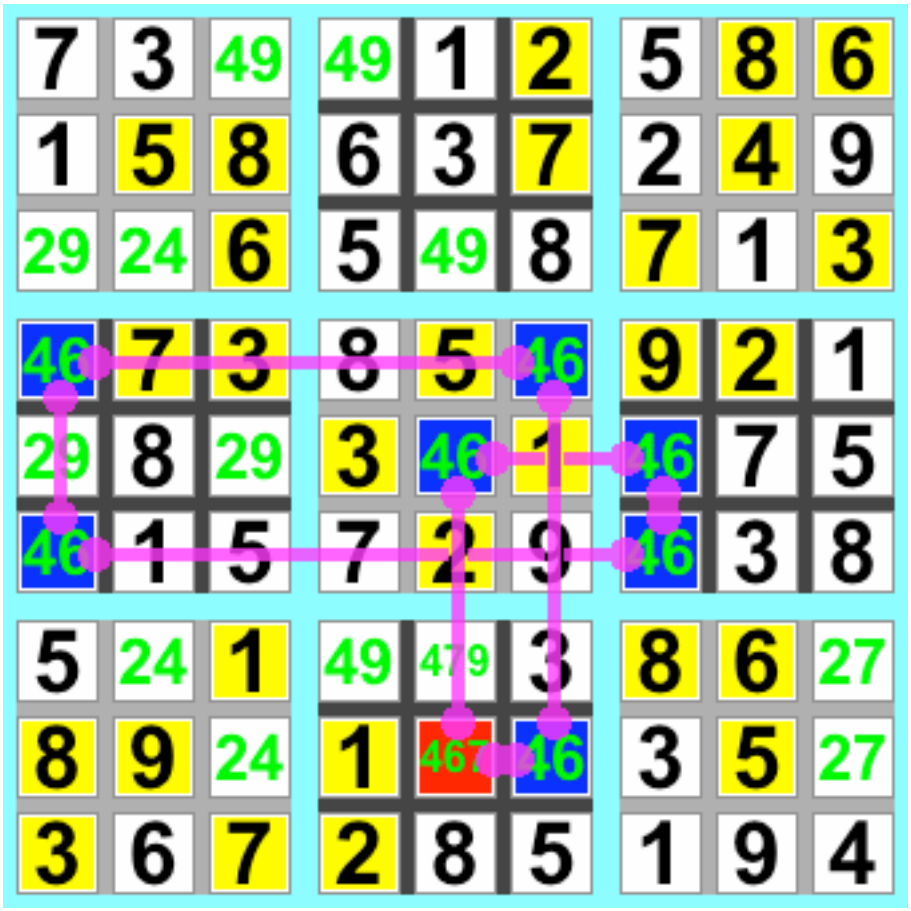
In this case, since <2> can only appear in row 1 in the roof squares, 7 can be removed from both of them.

If this puzzle looks familiar, it’s because once you find the Type-4B Unique Rectangle, it reveals the sample Type-4 Unique Rectangle!

As Type-4 Unique Rectangle solutions “destroy” the Unique Rectangle, it is usually best to look for them only after you've done any other possible Unique Rectangle reductions.

Unique Loops

The Unique Rectangles concept can be extended to a general case called Unique Loops. If, starting at the floor squares, there is a loop in the puzzle of squares containing the same possibilities, such that each row, column and block touched by the loop squares have exactly two loop squares in them, then you have a deadly pattern. Since this cannot occur in a valid sudoku, if you can find a loop that contains roof squares "opposite" the floor squares, you can apply the Unique Rectangles reductions.



Typing **u** shows you unique loops as well!

In the "Sample Unique Loop - Type 1 - 8 Squares" puzzle we see a fairly complicated Unique Loop (in fact, it's the only one I've found that has 8 squares; all the others have 6). The floor squares are R6C1 and R6C7. R8C5 must be 7 in order to preclude the deadly pattern.

These patterns are rare, but quite easy to see if you're looking for Unique Rectangles (which are, of course, simply 4-square Unique Loops). Once you have some candidate floor squares, do your search for roof squares as normal. If you don't find any, but do find a set of squares identical to the floor squares (but not lined up, because that would be the deadly pattern), then consider them the "2nd floor" squares, rotate your perspective 90 degrees, and keep on looking.

In the example on the previous page, we start out at R6C1 and R6C7, and since they share a row, we look in C1 and C7 for roof squares. We don't find them, but we do find R4C1 and R5C7, which contain the same values. These become our "2nd floor" squares, and since we were searching columns before, we now look in the rows.

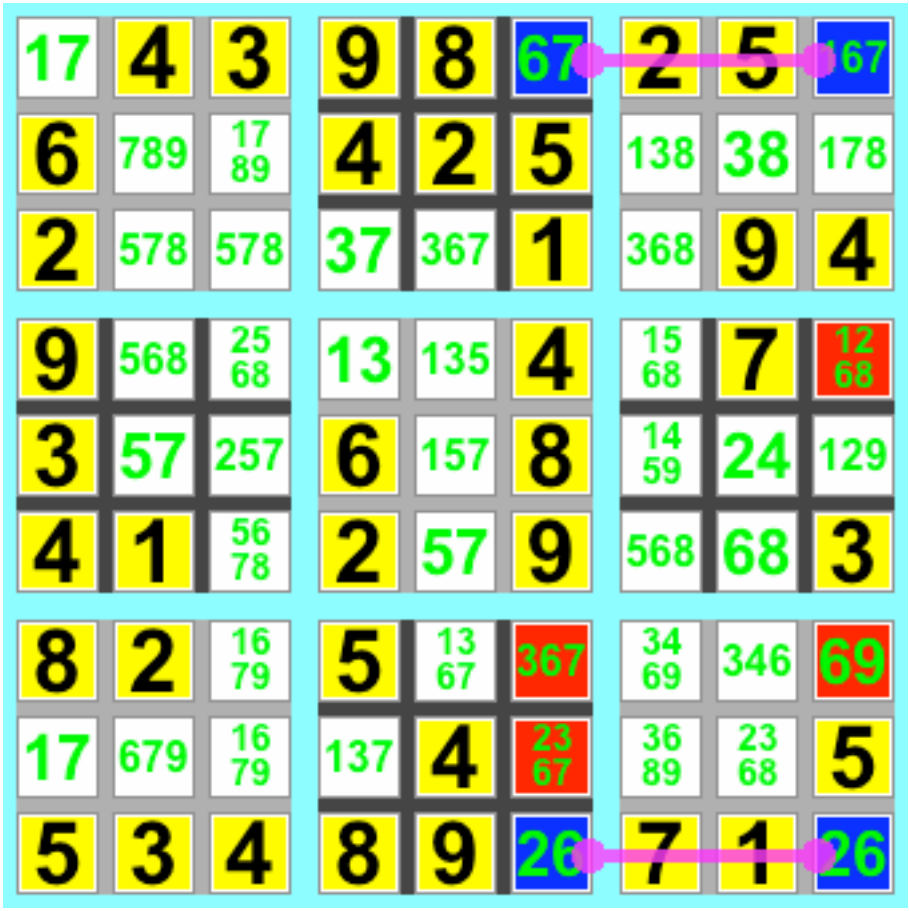
We again don't find any roof squares, but we do find R4C6 and R5C5, which become our "3rd floor squares", so we continue the search, again looking at columns. Finally we arrive at roof squares R8C5 and R8C6.

Looking back at our path, we see that each column, row and block the squares touched contain exactly 2 squares, so we have a valid Unique Loop.

These patterns are actually much easier to see than to describe. Try finding them in some of the other Sample Unique Loop puzzles, then highlighting to see if you were correct.

Simple X-Wing

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 (also known as a "conjugate pair"), 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.



Type **x** to display X-Wings and other "Fishy Cycle" patterns.

In the "Sample X-Wing" Puzzle, 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.

X-Wings are easy to understand 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!

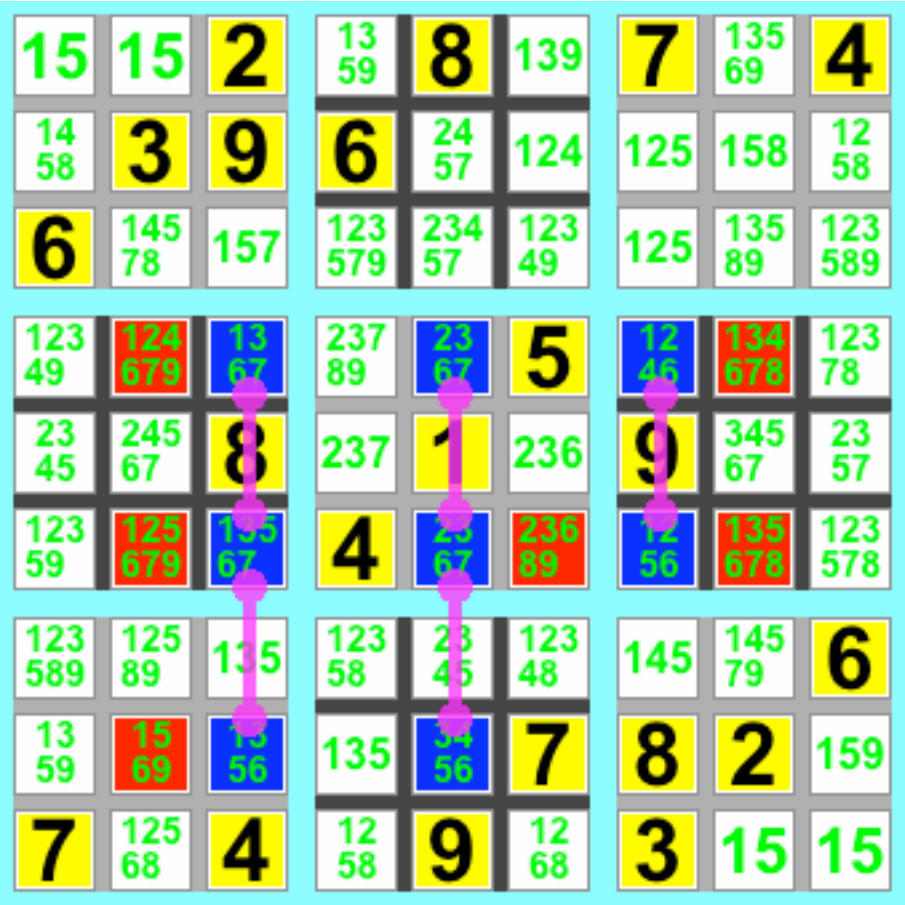
In the "Sample X-Wing #2" puzzle, you can see an example of an X-Wing that uses two columns that share two rows.

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.



Type **x** to display Swordfish and other "Fishy Cycle" patterns.

In the "Sample Swordfish" puzzle (shown here), a Swordfish can be seen, but is not needed to solve the puzzle.

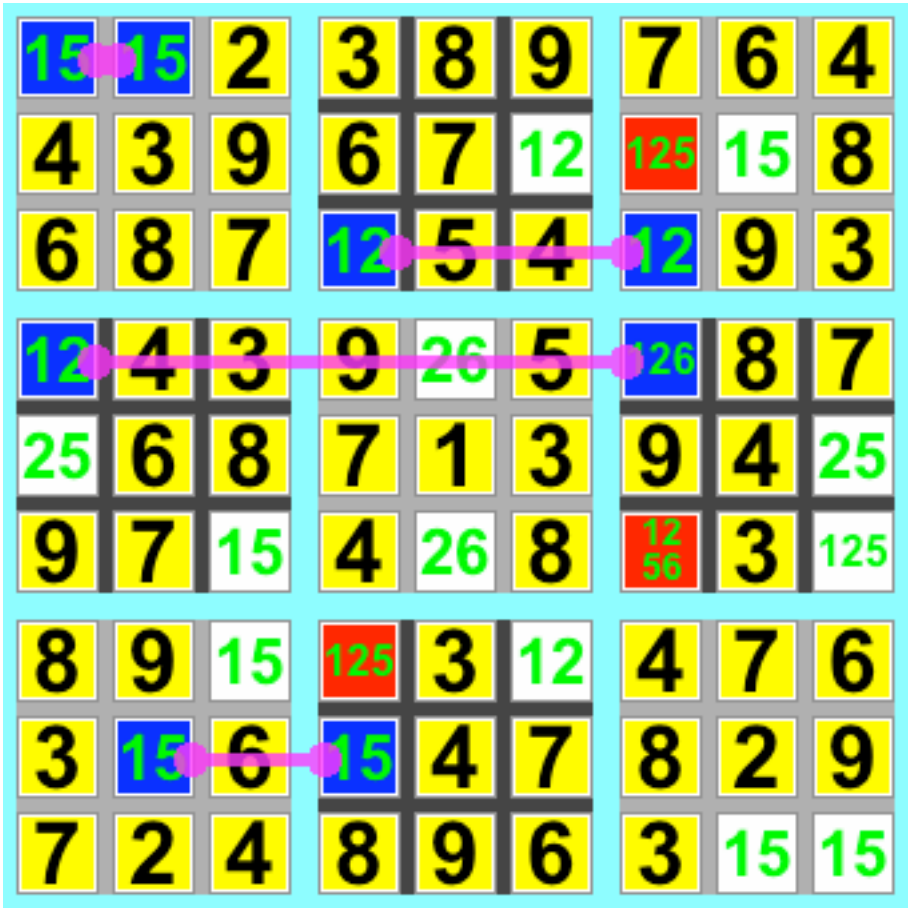
Note that two of the columns in this Swordfish have 3 squares that can be a 6, while the third has only two. It is quite possible to find Swordfish with 1, 2 or all 3 columns/rows having 2 possibilities.

The "each (common column/row) containing at least 2 squares" restriction above isn't really needed, but a Swordfish that has a column or row with only one square is also an X-Wing.

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.



Are you getting tired of me telling you to type **x** to see this pattern?

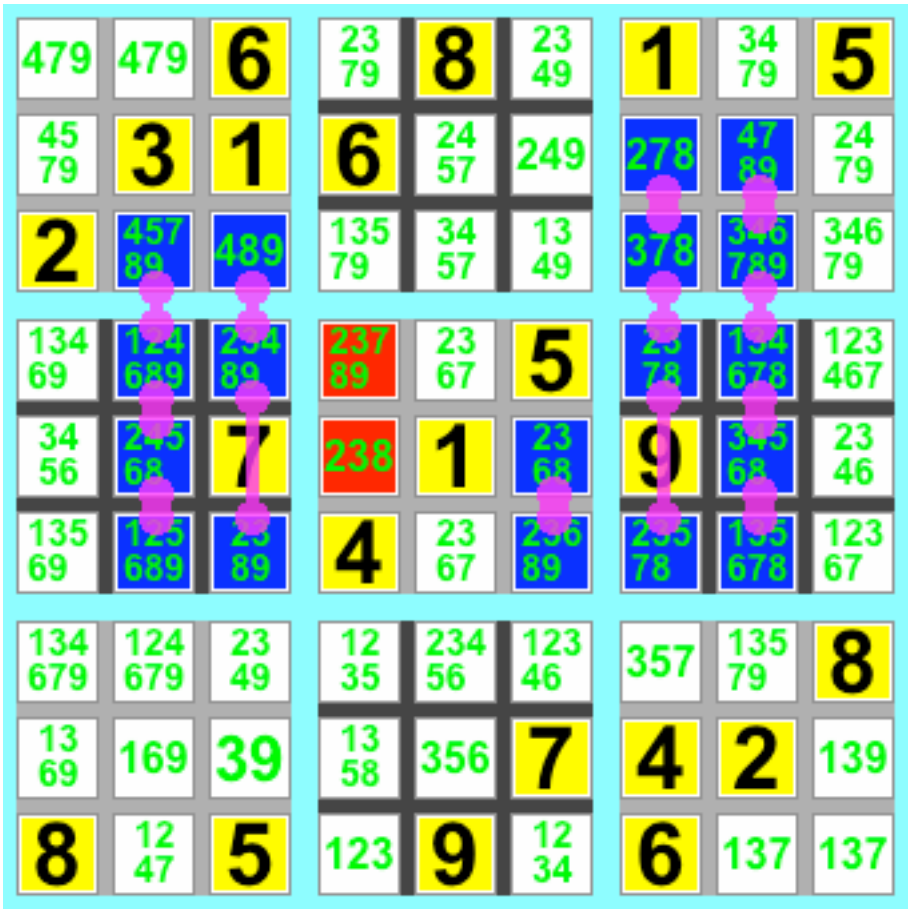
In the "Sample Jellyfish" puzzle, it's hard to see, but a useful Jellyfish does exist. To display it, you must mouse around the puzzle into R1C1, since there's also a swordfish in the puzzle that the hilighter will find first.

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 Squirmbag, 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, you can find a Squirmbag on possibility 8, but the deducer does not need to use it to solve the puzzle.

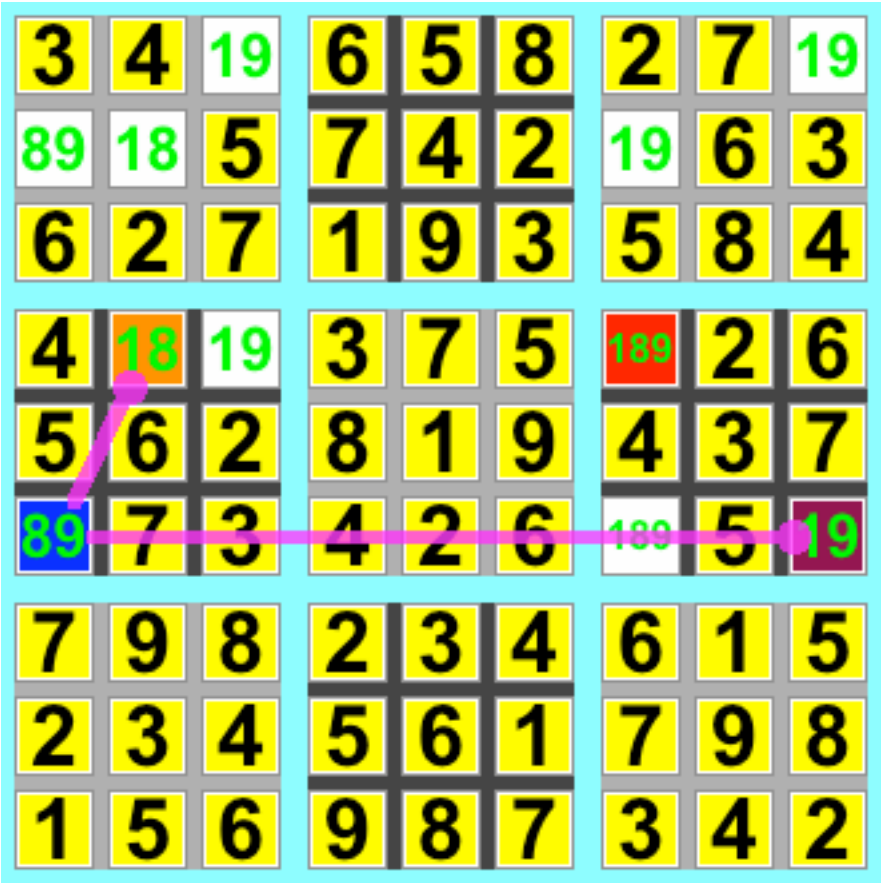
Squirmbags (and higher-order patterns of the same type) are not required to solve sudoku because simpler methods (like pins and intersections) or lower-order patterns embedded in them will always be available that permit reductions. Still, I couldn't resist adding a Squirmbag rule!

XY-Wings

XY-Wings are interesting patterns formed by sets of 3 bivalue (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.

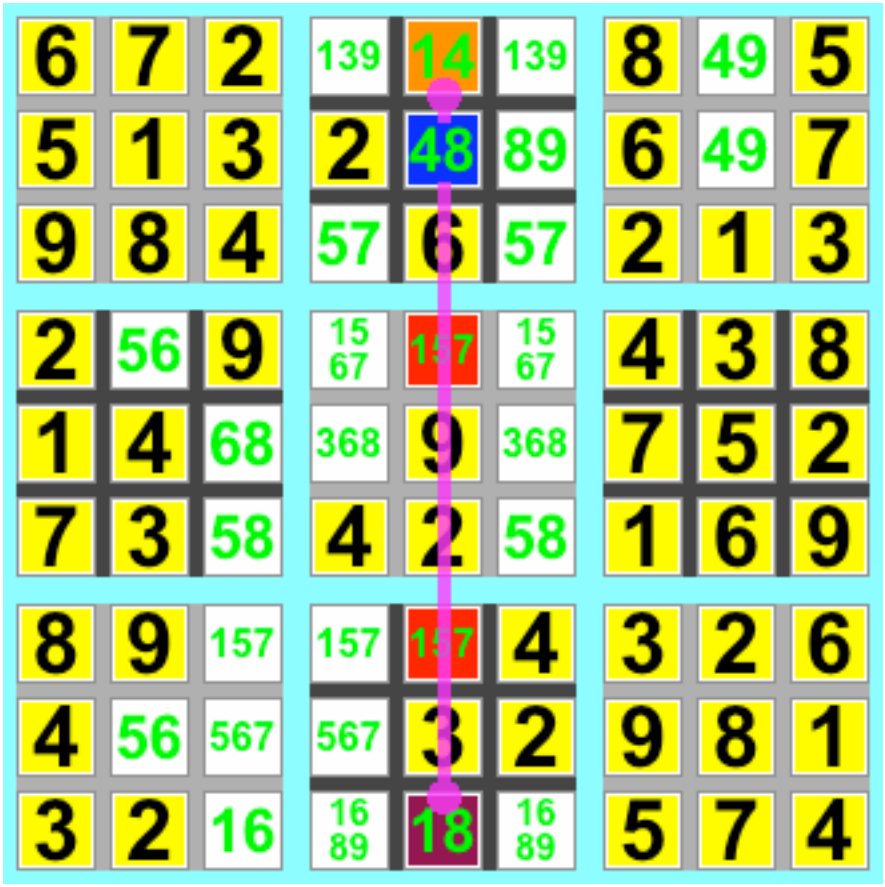


Type y to display XY-wings.

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. Note that the line connectors only have one "bump" on them, since they are indicating a force.

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 naked set. There's an example of one in the "Sample Bowman Bingo" puzzle.



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.

Type **z** to display XYZ-wings.

In the "Sample XYZ-Wing" puzzle, 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.

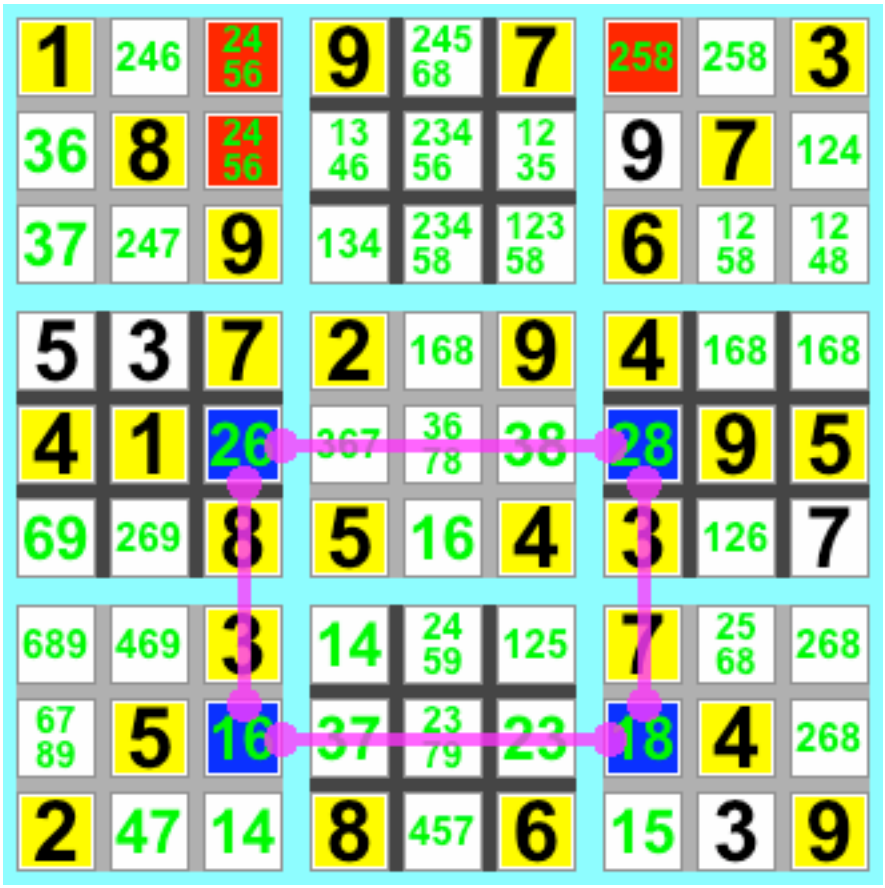
In a "Degenerate XYZ-Wing," XYZ, XZ, YZ and the square(s) that can't contain Z are all in the same row, column or block; this is the same as a comprehensive naked set.

Simple Forcing Loops

One of the nice things about squares that have only two possibilities left is that it is easy to see chains of forces in the puzzle, where choosing one of the values for one square forces another square to assume a single value, which in turn forces another square to assume a single value, and so on.

These chains of forces often form loops that end up back at the original square, and when they do, they have a very interesting property; they are always bidirectional.

Consider the following loop, from the "Sample Simple Forcing Loops" puzzle:



Type l to display simple forcing loops.

Starting at R5C3, if we assume it is a 2, then R5C7 must be 8, which forces R8C7 to be 1, which forces R8C3 to be 6, which forces R5C3 to be 2. Going the other way, assuming R5C3 is 6 forces R8C3 to be 1, which forces R8C7 to be 8, which forces R5C7 to be 2, which forces R5C3 to be 6.

This becomes important when you realize that this means that each adjacent pair in the loop has a single possibility in common, and that one of those squares must contain the possibility. So that means that none of their common buddies can contain it.

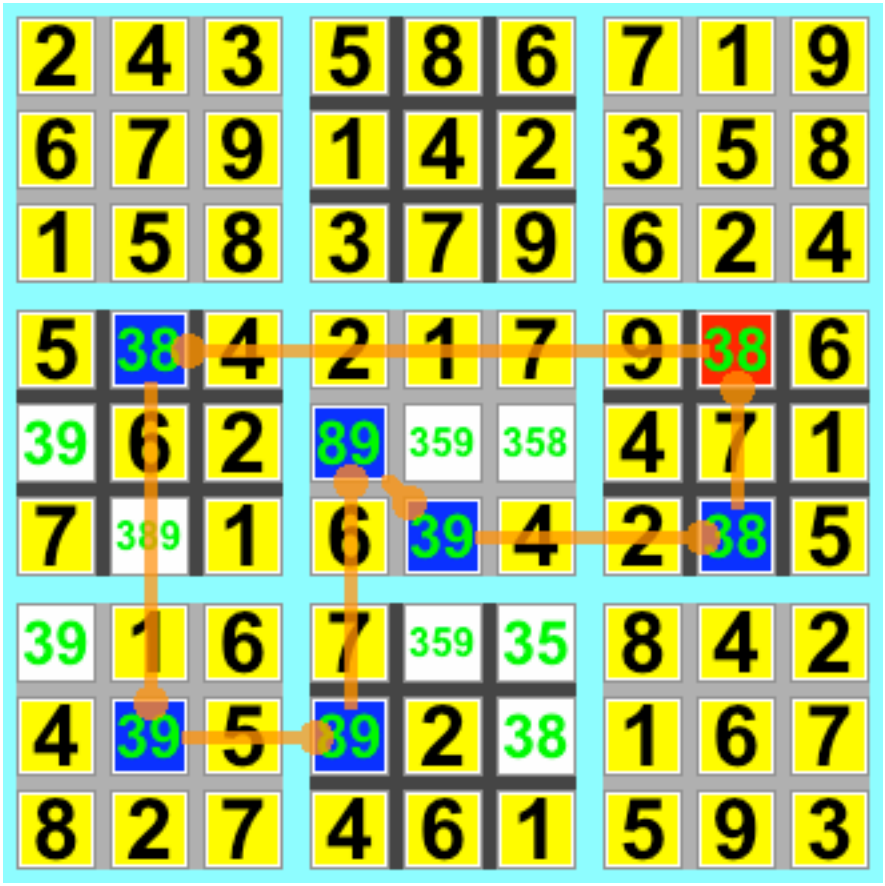
For example, R5C3 and R5C7 share the 2 in common, so none of their buddies can be a 2 (this doesn't help us, since their common buddies are the other squares in row 5, and none of them has any 2 possibilities. But consider R5C3 and R7C3! Their common value is 6, which can therefore be removed from R1C3 and R2C3. Similarly, R5C7 and R8C7 permit the removal of 8 from R1C7.

In order for a loop to be a true "forcing loop", it must span at least 2 rows, 2 columns and 2 blocks. Otherwise, it is actually a hidden set!

Forcing loops can be very long. The "Sample Forcing Loop (long)" puzzle contains a real snake - 12 links long!

Simple Forcing Chains

Simple forcing chains are almost identical to simple forcing loops; however, the idea this time is that when you get back to the original square, if you find a contradiction, you can eliminate your original choice from the puzzle!



Type **c** to see simple forcing chains.

In the "Sample Forcing Chain" puzzle, 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.

One big difference between Simple Forcing Loops and Simple Forcing Chains is that the initial square can have more than 2 possibilities; you can try to follow the loop choosing each initial possibility in turn looking for a contradiction. Note that if you do start on a 2-possibility square and manage to get back to the original square, you've either found a loop or chain, depending on whether or not you found a contradiction upon your return to the original square.

There is a second type of forcing chain that can sometimes be found. This is where the final link in the chain does not force the original square into a contradiction, but instead forces a buddy of the original square to have the same value as the original choice. Since two buddies can't have the same value, this is a contradiction, and the original choice must be invalid.

Comprehensive Forcing Chains

Comprehensive Forcing Chains are an expansion on Simple Forcing Chains. In these patterns, we allow ourselves to follow links in the chain by pins as well as forces.

Type **c** to see comprehensive forcing chains. Links in the chain that are formed by pins are hilighted in purple.

Note: the Chains hilighter ("o") first looks for simple forcing chains, and then comprehensive forcing chains.

Note that because we are following pins as well as forces, these chains are not inherently bi-directional. Thus, there is no simple way to do "Comprehensive Forcing Loops".

Comprehensive Forcing Chains are very rare, but you can occasionally spot a pin that is literally the missing link to finding a forcing chain. Also note that, as in the above example, some of the intermediate links in the chain can have more than 2 possibilities if and only if they are forced by a pin.

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 rules.

To use the **Nishio hilighter**, turn on possibility hilighting for the possibility you want to test, and select the square you wish to Nishio (it'll be hilighted, since it contains that possibility). Then type s or click the smiling samurai icon.

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, selecting it and typing **s**. You can only *nishio* 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. If you don't, however, then you can make no inferences from all your work.

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 a link to the online discussion that explains how they work:

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

Type x to display fishy cycles.

In the "Sample Fishy Cycle" puzzle, a fishy cycle with link number 9 exists that permits a cell to be reduced.

One type of Fishy Cycle that deserves 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, the X-Wing on number 9 is formed by row 2 and block 9, along with columns 8 and 9.

Conjugate Pairs

As you know from reading some of the previous explanations, conjugate pairs are a feature that appears in many of the more complicated patterns. So if you're totally macho (or macha...), try hilighting just the conjugate pairs and find the patterns for yourself.

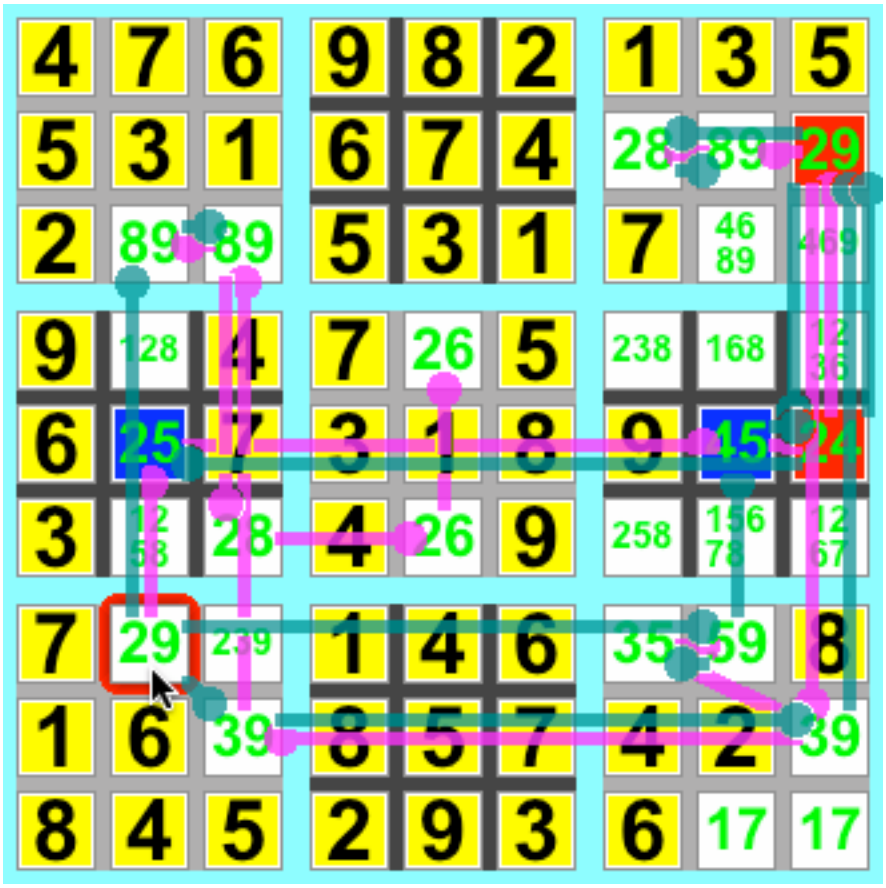
Type J to hilight conjugate pairs.

The "Sample Generalized X-Wing" Puzzle has many conjugate pairs!

Consequence Trees

This is a new feature I'm playing with in preparation to adding some advanced chaining heuristics. Consequence Trees shows you all the consequences of particular choices; after activating it, mouse over a square to hilight the chains of forces (and pins, if you want) that result from each possibility remaining in the square.

Squares that are **verities** (no matter what the initial value of the moused square, these squares are forced to the same value; see Trebor's Tables for more details) are hilighted in blue. Squares that prove **contradictions** (one of the initial possibilities of the moused square causes the square to be forced to two different values, so that initial possibility must be invalid) are marked in red.



Press **t** to display Consequence Trees; **shift-T** to include pin consequences.

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, you'll find a bingo that unlocks the solution to the puzzle. However, you must turn off all the rules but Bowman Bingo, then deduce, in order 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 has developed 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 one 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 things 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.

How to Table

Tabling is done exclusively through menu options in the **Trebor's Tables** menu. There are no keyboard shortcuts because tabling is a true expert option that is rarely used, and it could be disconcerting to accidentally trigger it.

Make Tables for all squares does a round of tabling for each square in the puzzle. You can do repeated rounds until the table is fully expanded (it'll tell you).

Make Tables for current square does a round of tabling for the currently selected square (you'll have to select it using the arrow keys).

The two **Show Tables** options both do a round of tabling, plus show the gory details of the tables. **Show Tables for all squares** can generate a huge amount of output.

Clear Tables resets the tables.

The **Table Settings** options are described in the next section of the manual.

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 Naked 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.

If **Sets and Intersections** is enabled, then implications are created not only for forces and pins, but also for more complex features such as **naked sets**, **hidden sets**, and **intersections**. The current "toughest puzzle" champion requires the tabling of intersections in order to be solved.

Note: naked and hidden set tabling has not actually been implemented yet, since no known puzzle requires it, and not doing it makes it easier to find a new champion puzzle that does!

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 Naked Pair puzzle, make R1C5 the current square, and Type "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 naked 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<>6, 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 Type "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.

Automating Internet Puzzle Fetches

If you create a file with a name that ends in ".url", and drag it into the app, the Susser will look for urls in the file, one url per line, and attempt to fetch puzzles from the internet from those urls.

For example, a file named dailysudokus.url containing these lines:

```
http://www.menneske.no/sudoku/eng/random.html?diff=5
http://www.menneske.no/sudoku/eng/random.html?diff=5
http://www.menneske.no/sudoku/eng/random.html?diff=5
http://www.menneske.no/sudoku/eng/random.html?diff=5
http://www.menneske.no/sudoku/eng/random.html?diff=5
http://www.palmsudoku.com/pages/s-o-t-d.php?st=2
```

would, when dragged into the Susser, load 5 random hard puzzles from the Menneske.no archive, plus the current PalmSudoku.com puzzle of the day.

There are some limitations on this feature; it only looks at the text returned by fetching the web-page, not at the graphics, and what is returned has to be either a simple puzzle text (which palmsudoku.com, for example, provides) or a format that the Susser has been programmed to recognize (such as the menneske.no archive pages).

Rating Puzzles

The Susser has a puzzle rating feature that is mostly of use to Sudoku puzzle creators. Here is how to use it:

- Create a folder containing puzzle files. These are usually text files containing puzzles, but can be graphic puzzles as well; basically, anything you can drag into the Susser normally.

  As things can slow down when there's a huge number of puzzles loaded, I recommend rating no more than 500 puzzles at a time, and quitting/restarting the Susser after doing each set of ratings.

- Check the deduction heuristics you want the Susser to use (and if you're using Trebor's Tables, use the menu options to select how you want tabling to proceed).

- From the **File** menu, select **Load and Rate Puzzles...**, and select the folder you want to rate.

The Susser will first load all the puzzles in the folder and add them to the puzzles list. Then it will solve each puzzle, and save the results to a file.

To speed up rating (and to reduce the impact of the memory management bug), the Susser won't update its puzzle display while rating puzzles.

A folder will be created in the same folder as the Susser application called "Rated Puzzles". In it will be subfolders, named for the various solving techniques (eg: "Pinned Squares", "Simple X-Wing", etc.) Each puzzle will be put in the folder that has the name of the most difficult technique needed to solve the puzzle. Puzzles that cannot be solved with the currently selected heuristics go into a "Unsolved" folder.

The filename used consists of a digit string that represents which heuristics were used, and how many times they were used (0-9), followed by the name of the puzzle, with a .txt extension. This means that when you view the directory, you can display the puzzles in rough order of difficulty.

In each file is a text representation of the puzzle, a list of the heuristics needed to solve it (and how many times they are used), and a full trace of the solution route.

Note: As the rated puzzles are added to the puzzles list, when you quit the Susser after rating, you probably don't want to save your changes!

In addition, a “Contains” folder will be created. In it will be files for every technique used to solve the puzzles, containing all the puzzles that required that technique, one per line.

Of interest only to total Sudoku Puzzle Generation Geeks is the **AutoRate** feature.

When the Susser launches, if it sees a file or folder named **AutoRate** in the same folder as the application, it will automatically load all the puzzles contained in the file or folder, rate them, and then quit.

After loading, but before rating begins, if AutoRate is a file, it will be deleted. If AutoRate is a folder, its contents will be deleted.

Also, the loaded puzzles will not be saved on quit.

This permits you to use the Susser as part of a sequence of automated operations (such as a loop that creates a bunch of puzzles, dumps them into the AutoRate folder, and runs the Susser to rate them).

In order to maximize rating speed, the main Susser window does not even become visible during AutoRate. It will appear after rating has completed, and you will have 15 seconds to cancel the shutdown (hold down the **shift** key) before the Susser quits.

Note that you can plop any file that the Susser can load -- text puzzles, graphic files, files that contain a ton of single-line puzzles, etc -- into the AutoRate folder (same goes for Load & Rate Puzzles).

You can even rate a .url file that contains a list of urls that are used to fetch puzzles from the internet! This is particularly useful if you want to download a few hundred puzzles from an archive and rate them in order to find puzzles with a particular feature. See ***Automating Internet Puzzle Fetches*** earlier in the manual for details on constructing a .url file.

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.

KrazyDad's Sudoku PDF's

http://www.krazydad.com/sudoku/

Great source of PDF puzzle books; if you need some dead tree sudokus, this is a good place to go. Puzzles can be cut&pasted out of the PDFs into the Susser, but if you're on a Mac, use Adobe Reader instead of Preview; Preview has a problem with the images and does not transfer them to the Susser properly.

Vanhegan.net

http://vanhegan.net/sudoku/

Publishes 6 daily puzzles of increasing difficulty (which the Susser can automatically fetch). Lots of other Sudoku info too.

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 Naked 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

I learned a lot, and with permission, stole some examples from, **Simon Armstrong**'s excellent Sudoku pages:

http://www.sadmansoftware.com/sudoku/techniques.htm

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, Bob Williams, Brian Ferguson, David Egyud, Gaby Vanhegan, Gérard Mattern, Graham Harrisson, Harald Hecht, Harry Erwin, Jim Wansey, John Harris, Kevin Stone, Lars Forsblad, Lluís Miret, Michael Karlsson, Nissim Sambethai, Noel Dillabough, Paul Vaderlind, Peter Kirk, Philip Milne, Richard Fowell, Richard Cain, Richard Karpinski, Richard Miller, Scott Cardais, Jean Smits, Steve Lurya, Steve Wozniak (yeah, him), Terry Chan, Toby Yamashiro, 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/**

The icons used in the Susser were adapted from Microsoft Clipart, except for the brain, which was provided courtesy of 3DScience.com (way cool images, check them out!).

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...

Known Bugs

All programs have bugs -- even mine, alas. But some of them are beyond my control because they are artifacts of the underlying operating system (such as Mac OSX) or development system (RealBasic). Until these are fixed, or workarounds are found, they are likely to be in the app for a long time.

If you find a bug not on this list, please email me at **trebor@animeigo.com** with the details. Make sure your subject line contains **SUDOKU SUSSER** and **BUG**, so it'll stand out and not get spamfiltered.

When reporting a bug, it is crucially important that you provide a complete sequence of operations that can repeatedly demonstrate the bug. Best of all is a sequence that starts "launch the program, load puzzle X, then..."

Without such an explicit sequence, it is often impossible for me to replicate the bug. And if I can't do that, I can't fix it.

*If you can repeat a bug (in particular, a crashing bug), one of the best ways of reporting it is to check the **Automatically save when anything changes** preference and run the program until it crashes, then send me both the sudokus.txt file and an exact and detailed description of your last action (not just what you did, but how you did it). Assume I'm a total idiot (you won't be far wrong) and explain in excruciating detail.*

*Finally, **please give me complete details about your computer configuration**. Nothing makes me feel more stupid than spending an hour trying to repeat a bug on my Mac, only to learn that it's a Windows-only problem! My wife insists that I have many talents -- but being psychic isn't one of them!*

Current Known Bugs:

Can't command while help-tag is helping: On the Mac, if you try and type a cmd-key while one of the yellow "help tags" is visible, it won't be processed. Workaround: none (but then, the Susser doesn't use many command keys!

Mysterious Print Documents Options: There's apparently a Macintosh plugin called *Print Documents* that adds itself as an option to Susser popup menus. It's harmless but annoying.

Linux puzzle display messups: One of my Linux users is reporting that the puzzle sometimes misdisplays (overprinting squares, or sometimes blacking them out). At present, I cannot replicate this problem; it may be an isolated issue.

Drag-n-Drop & Graphical Import in Linux / Windows: Most Linux and some older Windows OS variants have problems with graphical import and drag-n-drop. It all depends on what app you are dragging from, and the exact format of the graphic. So use this feature with caution and if it causes hangs, don't be surprised.

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