Making crosswords with Qxw

Mark Owen
qxw@quinapalus.com
http://www.quinapalus.com/qxw.html

Notes on Windows version by Peter Flippant

This guide describes releases 20190722 to 20200708 inclusive of Qxw. Significant differences from releases 20140131 and 20140331 are marked by a bar in the margin, as here.

July 8, 2020

Contents

Introduction

Qxw is a program to help you design and publish crosswords, from the simplest blocked grid to the most sophisticated thematic puzzle. It can make rectangular-, hexagonal- or circular-format grids with blocks, bars or both. It has an automatic grid-filling feature that can handle a wide range of answer treatments—you can even add your own answer treatment methods. Grids can be filled using letters, digits, symbols or a mixture of all three and a wide range of languages is supported.

Qxw produces output in a form ready for professional publication.

Qxw can also be used as a command-line tool to help automate the construction and filling of even more sophisticated grids, using a feature called 'decks'.

This guide is in two parts: the first part gives various informal examples of what you can do with Qxw and how you do it, while the second part is a more comprehensive and structured description of the facilities available.

 There are currently two versions of Qxw. One runs under the Linux operating system, and is tested on version 18.04 of the Xubuntu distribution: a Debian package is available. The other runs under the Windows operating system, from Windows 7 onwards. Some of the examples in this guide, particularly those involving the creation of customised answer treatments, are based on the Linux version. Differences affecting Windows users are discussed in Section 14.4.

Acknowledgements: Thanks to everyone who provided comments and bug reports on earlier versions of the program and its documentation, in particular to Nick Warne for extensive testing and assistance with troubleshooting. Please report any bugs you find to the address on the front page of this manual. Thanks also to Guido Bartoli, Janusz S. Bień, Adi Botea, Richard Brooksby, Csapai Andrea, Shirley Curran, Artti from Estonia, Matjaž Hladnik, Štefan Hozjan, Paul McKenna, Chris Montgomery, Jostein Sand Nilsen, Gareth Rees and Neil Shepherd for their assistance in implementing support for various languages in Qxw. Of course none of these bear any responsibility for any remaining infelicities in the program.

Offers of further assistance with improving and extending Qxw's built-in support for languages other than English are welcomed.

Part I

Examples

Chapter 1

A simple blocked grid

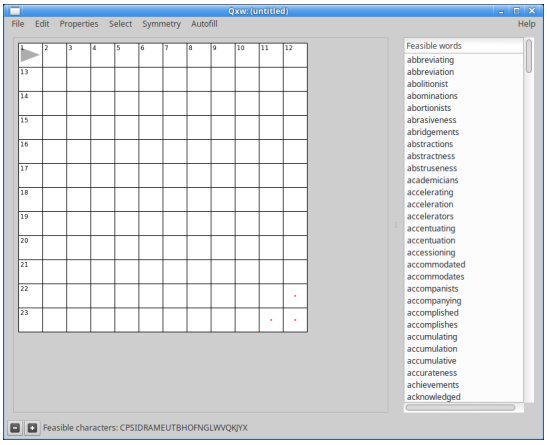

Figure: 1.1: How Qxw appears when it starts

The first thing to do when constructing a crossword with Qxw is to tell it the type and size of the grid you want. (You can change these later if necessary.) Select the menu item *Properties-Grid properties*. In the dialogue that appears, set the grid size to 7 columns by 5 rows. Leave the grid type as 'Plain rectangular'; you can fill in a title and author name if you wish. The dialogue should now appear as shown in Figure 1.2. Click on 'Apply' and the grid size will change as requested.

The grey triangle in the top left-hand corner of the grid is the cursor. You can move it using the arrow keys on the keyboard or by left-clicking in the middle of a grid cell with the mouse.

You can change the direction in which the cursor points using the 'Page Up' and 'Page Down' keys or the 'slash' ('/') key or, using the mouse, by clicking on top of the cursor. You can move it forward in the current direction by pressing the spacebar and backwards by pressing 'Backspace' (sometimes labelled with a left-pointing arrow).

You can now start to add blocks to the grid. Move the cursor down one cell and to the right one cell. Now press the 'Insert' key (labelled 'Ins' on some keyboards) or the 'comma' (',') key and a black block should appear under the cursor. There is a menu item *Edit-Solid block* that does

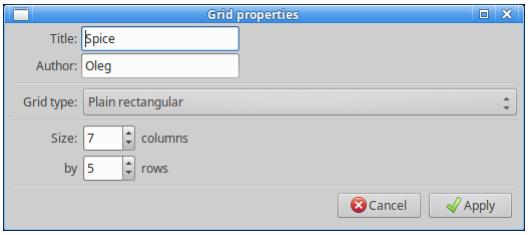

Figure: 1.2: The Grid properties dialogue

the same thing as pressing 'Insert'. You can also arrange things so that clicking the mouse in the corner of a cell creates or destroys a block there: see Section 10.1.

You will see that another block has also appeared in the grid, diagonally opposite the one you created. Qxw will try to maintain the symmetry of the grid as you construct it.

Continue adding blocks to the grid until it looks like Figure 1.3. If you make a mistake you can delete a block using the 'Delete' key (sometimes 'Del') or the 'full stop' ('.') key. There is again a corresponding menu item *Edit-Empty*.

Both 'Insert' and 'Delete' automatically advance the cursor.

You can also use *Edit-Undo* ('control-Z') to correct mistakes and *Edit-Redo* ('control-Y') to repeat mistakes when you realise they weren't mistakes after all.

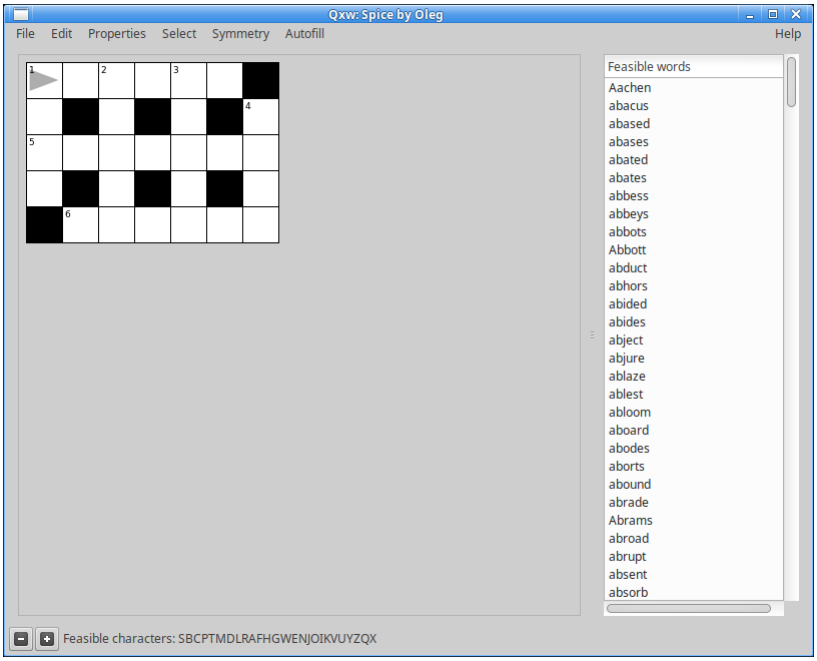

Figure: 1.3: A simple blocked grid

You can now proceed to fill the grid with words. Qxw can help with this process to various degrees, from suggesting individual letters and words to fully automated filling.

1.1 Automatic fill

When Qxw starts it will look for a suitable dictionary in one of the standard places on your computer. You will need a dictionary to use Qxw's automatic filling features: see Chapter 9 for more information.

For a completely automatic fill, select the menu item *Autofill-Autofill* (or press 'control-G'). Assuming that Qxw managed to find a suitable dictionary when it started up, it will fill the grid with words. At this point the words are only Qxw's suggestions, and so are shown in grey. Select the menu item *Autofill-Accept hints* (or 'control-A') to accept these suggestions, turning the letters black: see Figure 1.4. The results you get with automatic filling depend on many factors, most significantly on the dictionary you are using. Your filled grid is therefore unlikely to be identical to the one shown: this applies to all the examples in this guide.

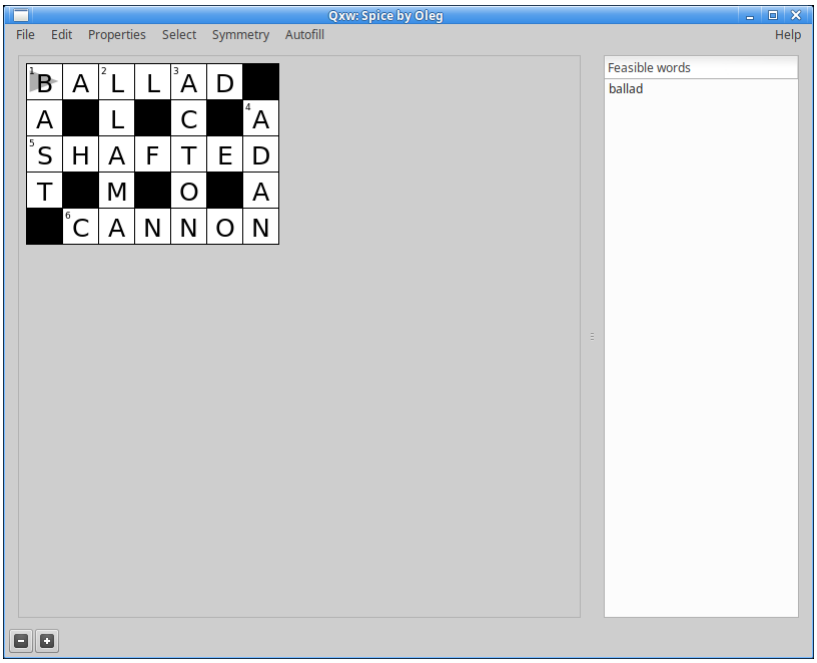

Figure: 1.4: An automatic fill of the simple blocked grid

Congratulations! You have just made your first crossword using Qxw.

Chapter 2

A more advanced blocked grid

In this chapter we will see how to construct a 15-by-15 blocked grid (a size used by many newspapers). We also have a few words that we want to include in the puzzle.

Begin by selecting the menu item *File-New-Blocked 15x15 template-No unches on edges*. This will provide you with a convenient starting point for creating the grid shown in Figure 2.1. Other sub-menus under *File-New* provide a range of useful starting points for various kinds of puzzle. You may want to make the window bigger to avoid scrolling if the grid doesn't fit; also, you can move the dividing bar between the grid and the panel to the right to make more space. The display zoom factor can be adjusted using the buttons in the bottom left-hand corner of Qxw's window or the menu item *Edit-Zoom*; as usual, there are keyboard equivalents and on many machines, you can hold down the control key while moving the scroll wheel on the mouse. Without the control key held down, you can also use the mouse wheel to scroll around the grid if it is too large to fit in its window.

Use the cursor keys and the 'Insert' or 'comma' keys as before to create the blocked diagram as shown.

You can save your work using the *File-Save As* menu item: you need to choose a filename for it, which should normally end '`.qxw`', although this isn't compulsory. Once you have established a filename you can subsequently use *File-Save*. Use *File-Open* to load a saved crossword back at a later date.

You can fill the grid manually by simply typing letters. As you type the cursor automatically advances in the current direction. You can delete the letter under the cursor using the 'Delete' or 'full stop' keys, or you can use 'Tab'. Unlike 'Delete', 'Tab' will not delete blocks. 'Shift-Tab' operates as 'Tab', but moves backwards rather than forwards.

Enter the thematic words for this crossword as shown in Figure 2.2.

At this point we could use the automatic filling feature to complete the grid; however, we will continue this example in a more interactive way to demonstrate how Qxw can gently guide you in choosing suitable words.

2.1 Guided fill

You may have noticed as you constructed the grid that the feasible character list at the bottom of Qxw's window and the list of words in the right-hand panel were continuously being updated.

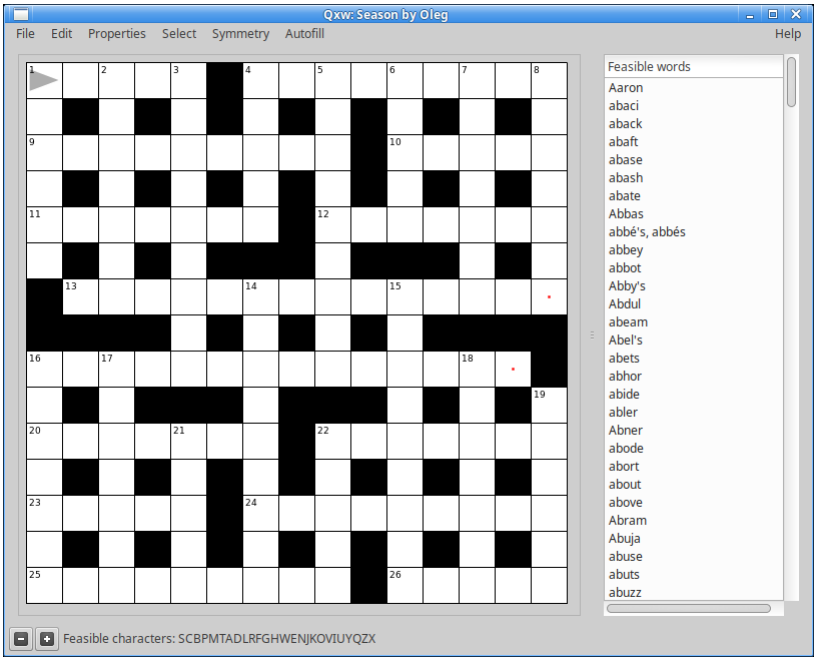

Figure: 2.1: Blocked grid

The feasible character list shows what characters can be used to fill the cell under the cursor (the 'current cell'), in order from most promising to least promising. The right-hand panel shows the words that can be used to fill the grid entry that runs through the cursor in the direction in which it points: this grid entry is called the 'current light'.

Also, you will see small red dots start to appear in the diagram. These are 'hotspots', where there are relatively few feasible letters. The bigger the red dot, the fewer possibilities there are in that cell. Qxw takes into account the possible combinations of crossing words when computing which letters are feasible in each cell—it uses a small amount of 'look-ahead'—and so can often see situations where a fill will be difficult or impossible before they become apparent to the user. See Section 2.1.1 for how you can configure this behaviour.

If only a single possibility remains for a cell, the forced letter will be shown in grey. To see this effect, try deleting any one of the letters of 'MISTLETOE': unless you are using a very odd dictionary Qxw will supply the missing letter. Using the menu item *Autofill-Accept hints* (or 'control-A') you can make any letters shown in grey in the grid into a permanent part of the crossword as if you had typed them in.

When no possibilities remain to fill a cell, a grey question mark is displayed. Qxw looks ahead far enough that grey question marks will propagate over the entire grid when a fill becomes impossible.

In the grid shown in Figure 2.2 the red dots tell us that the third down light (the nine-letter word starting with 'Y') is likely to be the most constrained. Move the cursor over to that light and press 'Page Up' or 'Page Down' or 'slash' until it points in the Down direction. On the right you will see a list of feasible words: see Figure 2.3.

You can choose one of these words by clicking on it. The word will be entered as the current light, and the rest of the grid will update to show you where the new hotspots are. If you are

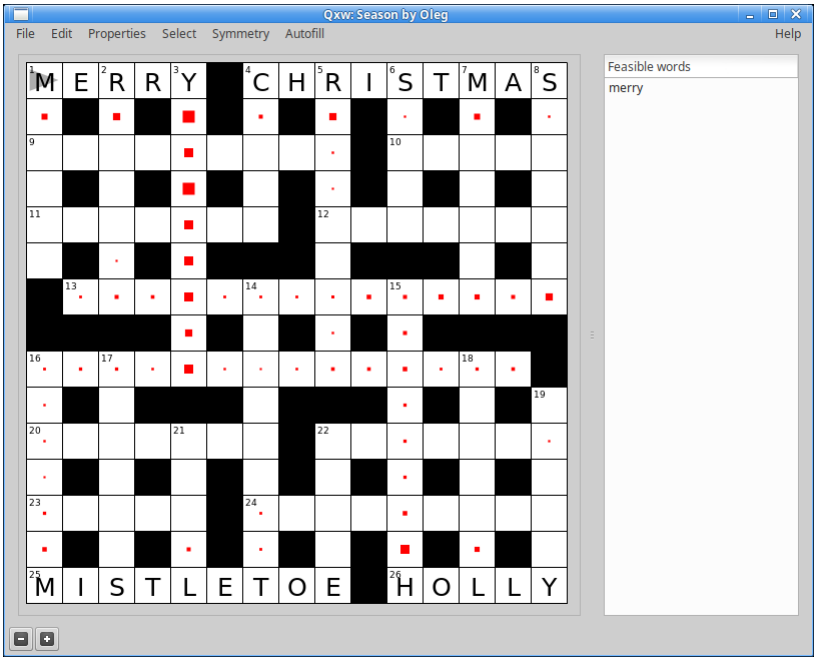

Figure: 2.2: Blocked grid with theme words added

unlucky, the remainder of the grid will turn to question marks. This means that there is no possible fill; delete the entry and try again. At this point you may find the *Edit-Undo* ('control-Z') and *Edit-Redo* ('control-Y') commands useful to save a lot of tedious deleting and re-entering of lights.

At any point you can right-click on a word in the feasible word list, which will offer you a range of options. You can look up the word using an on-line dictionary, encyclopaedia or search engine; you can copy it to the clipboard; or you can 'ban' it, which means that it will not be considered subsequently when filling the grid. All words can be unbanned using the menu option *Autofill-Unban all answers*.

In this way you can complete the process of filling the grid manually, with just a little assistance from Qxw. You don't need to enter whole words at a time if you don't want to: you can always type individual letters wherever you like in the grid. You also always have the option to use the automatic filling feature to complete the grid at any point.

If at any time you decide that you want to erase all filled entries and start again, use the *Edit-Clear all cells* ('control-X') command.

2.1.1 Configuring interactive assistance

It is possible to reduce the amount of automatic assistance Qxw offers you while constructing a grid: this can be useful to avoid distraction at an early stage of trying out grid ideas. It also slightly reduces the amount of processing power Qxw uses, which will in turn lead to a marginal improvement in battery life on portable machines. Use the menu item *Autofill-Interactive assistance-Off* to disable assistance altogether; use *Autofill-Interactive assistance-Light only* to arrange for Qxw to consider only the current light when building the feasible

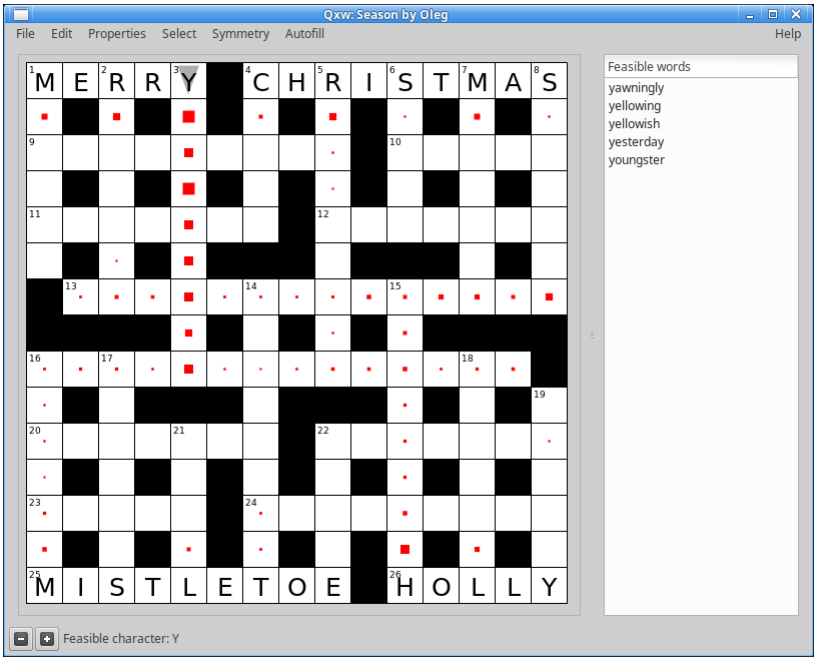

Figure: 2.3: Using the feasible word list

word list, ignoring the constraints implied by any checking lights and thence the rest of the grid; and use *Autofill-Interactive assistance-Entire grid* to force Qxw to go as far as it can in determining which characters and lights are feasible without embarking on a search that might entail backtracking. This last setting is the default when Qxw starts.

2.2 More advanced editing

The Edit menu includes several more powerful functions for modifying a grid. You can insert and delete rows and columns, flip the grid in the main diagonal, and rotate circular grids (see Section 4.2). For circular grids the row and column editing functions operate on annuli and radii respectively.

2.3 Saving and exporting

You should of course save your work regularly. (Qxw does not make automatic backups.) Saving a crossword in Qxw's native format saves the grid contents and size, the title and author, and the names of the dictionary files. It does not make a copy of the dictionaries themselves. The banishment status of words is not saved.

When you have completed your grid you will want to save it in a form suitable for publication in print or on the Internet: this is called 'exporting'. Qxw can export your puzzle in various ways in a range of file formats. The export options are listed under the *File* menu.

You can export the **blank grid image** (i.e., without answers) or the **filled grid image** (i.e., with

answers) in EPS, SVG, HTML or PNG formats. EPS ('Encapsulated PostScript') is a format for drawings widely used in professional publishing; SVG ('Scalable Vector Graphics') is an XML-based format for drawings also suitable for professional publishing applications; the HTML format makes your crossword into a web page using CSS ('cascading style sheets') to render the grid; and PNG is a bit-mapped graphics format also suitable for use on the Internet. Most modern browsers, including versions 9 and above of Internet Explorer, also support the SVG format directly. The EPS, SVG and HTML formats can be scaled up arbitrarily after export without loss of image quality; PNG format images look blocky when scaled up. Qxw cannot export non-rectangular grids in HTML format.

You can also export just the **text of the answers** either in simple HTML or in plain text: this output can be used as a skeleton for writing clues using your favourite word processor or HTML editor.

The **puzzle as a whole** or the **solution** can be output either as pure HTML (rectangular grids only) or as a combination of HTML with a PNG or SVG image to represent the grid. You can use an HTML editor to add clues, solution notes, or any other text.

When the text of the answers is included in an exported file, all possibilities are listed. If you have not completely filled the grid then the resulting files can be quite large and, especially if you are using some of Qxw's more advanced features, can take a long time to generate.

Qxw includes an experimental feature under *File-Export other format* to generate XML output intended to be compatible with Crossword Compiler. Since there are significant differences between the types of puzzle supported by Qxw and by Crossword Compiler, there can be no guarantees of compatibility, especially if your grid uses some of Qxw's more exotic features. The main limitations are that only rectangular grids are supported; only one character per cell is supported; merged cells, grid topologies, answer treatments, multiplex lights and free lights are not supported; and general corner marks are not supported, although circles around letters are.

Qxw will also attempt to read SYM and SYT format files: you can just open them in the normal way using *File-Open*. Again, compatibility is limited; and furthermore, since these file formats are not properly documented, there can be no guarantee that this function will work at all.

Warning: Qxw cannot recover a crossword from its exported form. You must save your work using the *File-Save As* **or** *File-Save* **menu options.**

Chapter 3

Barred grids and entry methods

Now we will look at how to create a barred grid in Qxw. Barred grids are popular for advanced crosswords that use more obscure vocabulary; this makes the choice of dictionary more critical, and it is a good idea to read Chapter 9 before embarking on the construction of such a crossword.

Qxw does not particularly distinguish between barred and blocked grids: you can even mix bars and blocks within a grid if you like. So we start in the same way as before, using the *Properties-Grid properties* menu item to set the overall size. For this example we will use a 12-by-12 grid, which is popular for this type of crossword.

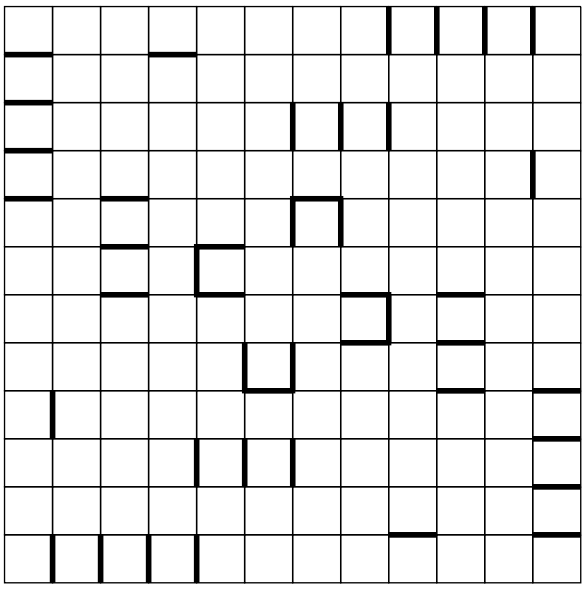

Figure: 3.1: Simple barred grid

3.1 Symmetry

The previous example grids we looked at had twofold (i.e., 180-degree) rotational symmetry automatically provided by Qxw. This is the default, but you can change it: here we will insist on fourfold (i.e., 90-degree) rotational symmetry. To do this, select the menu item *Symmetry-Fourfold rotational*. As you can see from the menu, Qxw offers a wide range of other symmetry types, including mirror symmetries and 'up-down' and 'left-right' symmetries, in which the bar or block pattern in one half of the grid matches that in the other. The various types of symmetry may be combined at will.

The menu option *Symmetry-None* gives you a quick way to clear all the symmetry settings.

If you change the symmetry setting in the middle of grid construction, any subsequent operations will respect the new setting, but the pattern will not otherwise change.

3.2 Adding bars

With the symmetry specified, you can proceed to add bars to the grid. Place the cursor *after* the point where you wish to insert a bar, pointing *away* from the bar position, and press 'Return'. Alternatively, use the *Edit-Bar before* menu item. Repeat the operation to remove the bar.

You can also arrange things so that clicking the mouse on the edge of a cell creates or destroys a bar there: see Section 10.1.

Add bars to the grid until it appears as in Figure 3.1.

The grid can now be filled manually or automatically as before. The result might look like Figure 3.2.

O	B	S	E	S	S	I	V	E	E	C	F
B	R	U	N	E	L	L	E	S	C	H	I
B	O	V	I	N	E	K	S	C	L	A	N
S	T	A	I	D	E	S	T	A	A	R	I
C	H	E	S	S	P	I	A	L	I	A	S
R	E	N	T	R	E	S	E	A	R	C	H
A	R	B	I	T	R	O	N	T	A	T	E
W	H	I	G	S	S	T	S	E	D	E	R
N	O	G	M	P	L	O	T	T	E	R	S
I	O	W	A	O	I	P	E	R	M	I	T
E	D	I	T	O	R	I	A	L	I	Z	E
R	S	G	A	N	E	C	D	O	T	E	S

Figure: 3.2: Simple barred grid with example automatic fill

3.3 Reversals, cyclic permutations and jumbles

Qxw will let you create grids with words entered forwards or backwards, or permuted in various ways. For example, start with a blank nineteen-by-nineteen grid, and select the menu item *Properties-Default light properties*) and then tick all the boxes 'Allow light to be entered normally', 'Allow light to be entered reversed', 'Allow light to be entered cyclically permuted', 'Allow light to be entered cyclically permuted and reversed' and 'Allow light to be entered with any other permutation'. An automatic fill of this grid (which may take some time to generate) might look like Figure 3.3.

E	E	O	I	T	A	R	N	E	T	R	S	N	E	R	P	S	I	M
I	T	M	C	A	E	S	N	P	R	T	I	D	N	U	O	P	L	E
S	I	A	R	M	M	T	G	E	I	S	T	R	N	E	N	F	O	A
N	N	H	L	I	S	V	P	A	S	I	E	E	O	R	R	T	E	I
I	T	L	R	C	C	T	L	L	E	A	O	E	A	U	S	Y	C	O
N	E	O	O	S	E	C	I	C	N	F	I	A	A	T	R	N	N	T
C	N	G	T	E	S	E	O	O	L	I	I	R	R	N	T	L	T	N
E	I	A	N	L	N	N	H	T	S	R	L	D	T	E	S	L	A	E
T	E	I	E	T	D	A	A	C	P	K	A	S	I	O	N	N	O	R
P	O	C	A	E	T	N	T	A	A	E	V	I	L	M	R	R	O	I
V	G	E	R	N	T	I	L	I	C	O	O	E	G	N	E	E	R	A
E	V	N	A	I	R	G	O	T	E	T	P	C	R	A	E	S	I	N
R	E	T	P	N	E	O	R	M	I	E	N	M	A	C	E	O	S	T
A	L	N	E	A	E	I	E	N	S	T	R	P	T	F	O	R	E	S
N	I	G	A	P	E	R	R	T	A	N	A	K	O	L	I	A	C	L
O	A	R	S	R	L	E	A	G	I	N	E	T	S	O	V	I	N	E
P	R	D	O	H	C	T	Y	E	O	C	C	R	I	H	A	E	L	S
S	N	I	E	D	O	A	C	E	V	I	T	I	N	I	T	O	F	R
A	N	T	H	O	N	I	G	R	O	O	D	T	I	S	E	I	T	P

Figure: 3.3: Grid filled with jumbled words

You can see the lights in unjumbled form by moving the cursor around the grid and looking in the feasible word list. (They are: remisrepresentation; cruel disappointment; antiferromagnetisms; inverse relationship; acoustoelectrically (or electroacoustically); sinfonia concertante; controlling interest; Netherlands Antilles; take as a precondition; comparative relation; regenerative cooling; prerogative instance; Remontoir escapement; false representation; Glacier National Park; overgeneralisations; scorched earth policy; overidentifications; photodisintegration; nonappreciativeness; ventilation engineer; come to a grinding halt; operational research; non-medical therapist; consecrated elements; tracer investigation; rontgenographically; electromagnetic tape; overspecialisations; restriction of intake; positive declaration; Kidderminster carpet; interorganisational; thermonuclear fusion; overrepresentations; interprofessionally; centre of oscillation; and representationalism.)

Chapter 4

Grid shapes and topologies

Qxw lets you create grids in a variety of shapes beyond the conventional rectangle or square.

4.1 Cutouts

The simplest way to customise the shape of the grid is to use cutouts. You start with a conventional rectangular grid and remove unwanted cells. An unwanted cell is removed by placing the cursor on it and pressing 'control-C' (or selecting the menu item *Edit-Cutout*). The removed cell is shown shaded on the screen and is not shown in exported versions of the grid. To return a cell to the grid, press 'Delete' or 'full stop' (to turn it into an empty cell) or 'Insert' or 'comma' (to turn it into a block). Figure 4.1 shows a simple example of custom grid shape created in this way.

> Like 'Insert' and 'Delete', creating a cutout automatically advances the cursor.

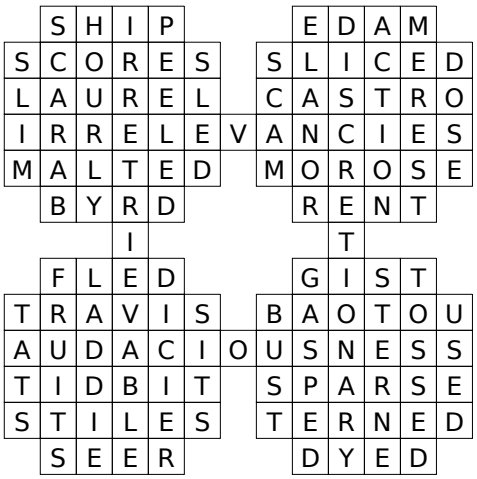

Figure: 4.1: Custom grid shape created using cutouts

4.2 Circular grids

Qxw can create crosswords with circular grids. First select the menu item *Properties-Grid properties*. In the dialogue that appears set the grid type to 'Circular' and the size to 20 radii and 8 annuli. The resulting grid template appears as shown in Figure 4.2.

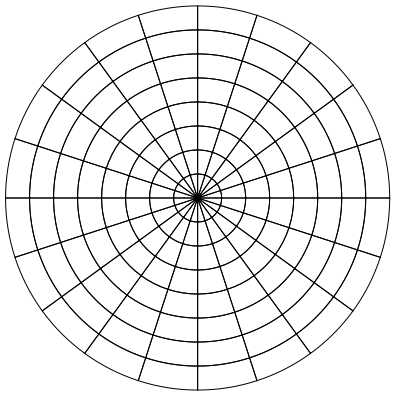

Figure: 4.2: Circular grid template

You can move the cursor using the arrow keys: the left and right arrow keys move the cursor forwards and backwards within an annulus, while the up and down arrow keys move the cursor radially away from and towards the centre. The 'Page Up', 'Page Down' and 'slash' keys change the direction of the cursor as before.

Although Qxw will happily let you fill them, the cells near the centre of the grid are too small to fit a letter in comfortably. There are two approaches to solving this problem, which can be used in combination.

First, you can use cutouts to remove the centre cells from the grid as described above. (The job can be made a bit quicker by temporarily increasing the rotational symmetry setting—note that with 20 radii Qxw offers you, for example, the possibility of fivefold rotational symmetry.) The result might look like Figure 4.3.

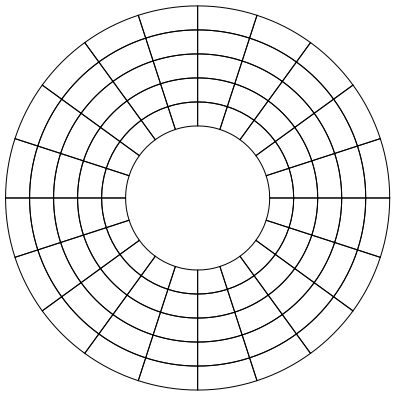

Figure: 4.3: Circular grid with central cutouts

Second, Qxw gives you the option to *merge* two or more cells into one by deleting the grid line that divides them. With the cursor pointing at the grid line to be deleted, press 'control-M' (or select the menu item *Edit-Merge with next*). The grid line ahead of the cursor will disappear: two cells have been merged into one. The operation respects the settings selected under the *Symmetry* menu. An example of the kind of grid you can create using merged cells is shown in Figure 4.4.

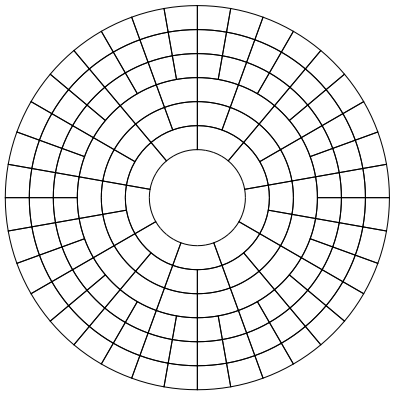

Figure: 4.4: Circular grid with cutouts and merged cells

When filling the grid, a set of merged cells is filled just as if they were a single cell.

Pressing 'control-M' within a merged cell with no grid line immediately ahead of the cursor will restore that grid line, undoing (although only partially if more than two cells have been merged into one) the merge operation.

All the sub-cells comprising a merged group must lie in a line in one direction; in other words, for a circular grid, they must lie consecutively within one annulus or radius. Qxw will enforce this restriction by demerging cells as necessary.

Figure 4.5 shows an example of a more complex circular grid, with bars added to create lights within the annuli. (If there are no bars within a given annulus, Qxw treats it as if all the letters in that annulus are unchecked. To obtain the effect of a single light occupying the whole annulus, add a bar: the innermost annulus in the figure illustrates this.)

It is common in circular grids for words to be entered either forwards or backwards. To achieve this effect, select *Properties-Default light properties* and tick the box 'Allow light to be entered reversed'; then click 'Apply'. Figure 4.5 was made with this setting. It is possible to allow reverse entry for only certain lights, for example only radial lights, using 'light properties': see Chapter 12.

Cell merging can be used in conjunction with Qxw's other grid types, although this is less commonly wanted. Figure 4.6 shows a simple example of a rectangular grid with merged cells.

In general using merged cells increases the degree of checking between the entries in the grid, and thus can make the grid harder to fill.

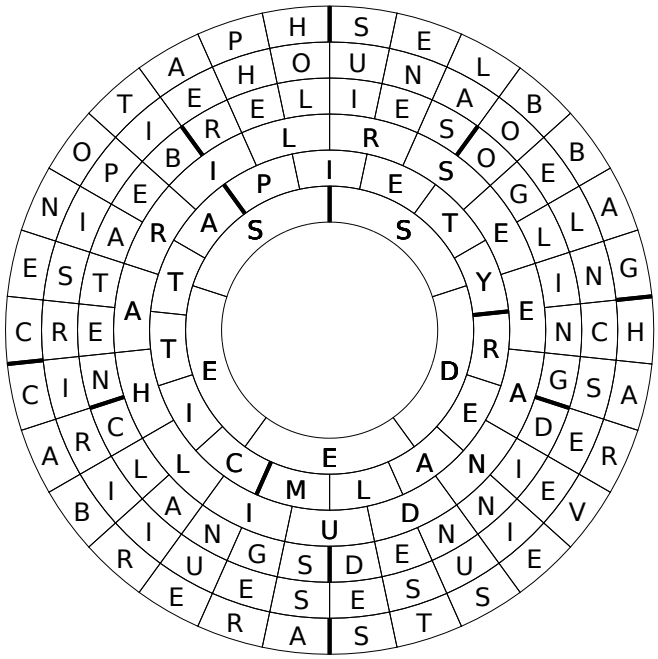

Figure: 4.5: Filled circular grid with some words entered backwards

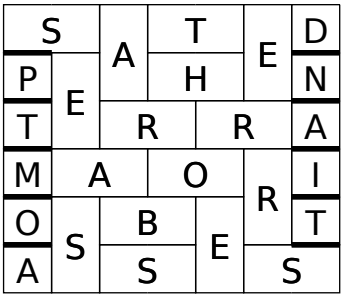

Figure: 4.6: Filled rectangular grid with merged cells

4.3 Hexagonal grids

Qxw can also create crosswords based on hexagonal grids. In a hexagonal grid lights can run in three different directions. Two types of hexagonal grid are provided: one with lights running northeast, southeast and south, and the other with lights running east, southeast and southwest. As usual, you specify the grid type and size using the *Properties-Grid properties* menu item and you can cycle the cursor direction through the available options using the 'Page Up', 'Page Down' and 'slash' keys. In other respects, constructing a hexagonal grid is just like constructing a rectangular or circular grid. Figure 4.7 shows a simple example of a hexagonal grid.

Because most cells can have lights running through them in three different directions, it is possible for cells to be triply checked. This means that care is needed to ensure that overall the degree of checking is not so high that the grid cannot be filled.

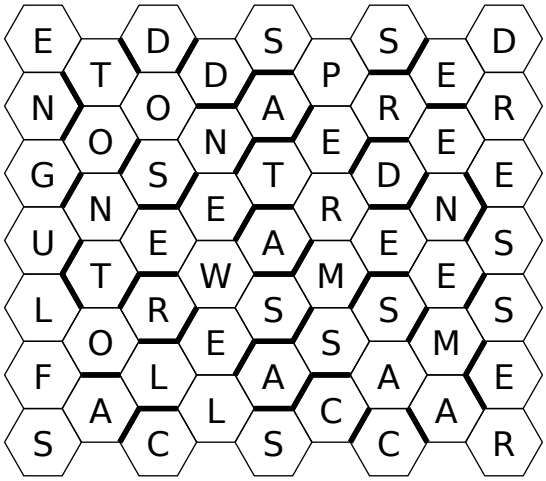

Figure: 4.7: Filled hexagonal grid

4.4 The Isle of Wight

Qxw lets you construct grids where there is more than one letter in certain cells. You can configure how the letters in the cell contribute to lights running through that cell.

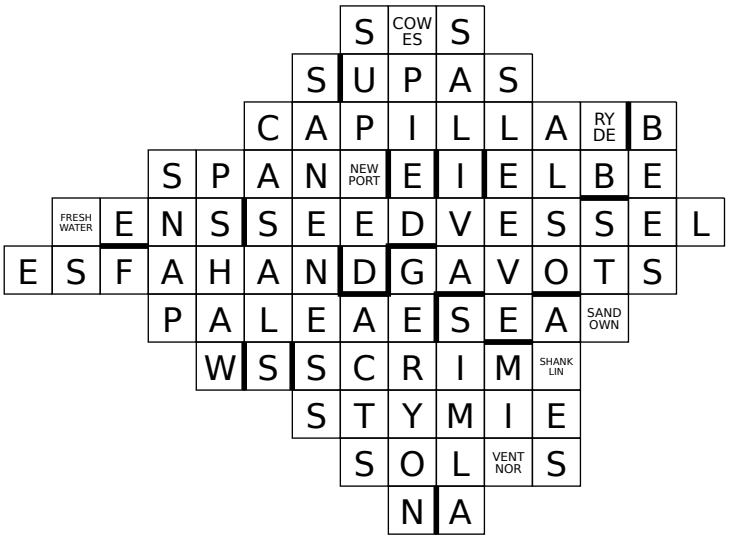

Figure: 4.8: The Isle of Wight

Figure 4.8 shows a map of the Isle of Wight with major towns marked. In each case the name of the town is divided into two parts, the first part being given by the across light through the cell and the second part by the down light. So, for example, 'Ventnor' is located at the intersection of 'solVENTs' and 'miNOR'.

A grid like this is created and automatically filled as follows. Start with a rectangular grid of suitable size to enclose the whole map, and use cutouts (see above) to make the desired overall

shape. Then select the cells where towns are to be placed: move the cursor to each in turn and choose the menu item *Select-Current cell* ('shift-C'). Now choose the menu item *Properties-Selected cell properties* and tick the 'Override default cell properties' option. At the bottom select 'Lights intersecting here need not agree (vertical display)'. Click 'Apply'.

Move the cursor to the first selected cell and choose the menu item *Edit-Cell contents* ('control-I'). In the two text boxes enter the two parts of the town name: for example, enter 'VENT' for the contribution to Across lights, and 'NOR' for the contribution to Down lights. Obviously you will want to divide the name in such a way that it does not make the fill too difficult. Repeat this for each selected cell, entering the name of each town.

You can now deselect the cells using the menu item *Select-Nothing* ('shift-N'). Then, with a certain amount of trial and error, you can add bars to the grid to allow it to be filled.

4.5 Grid topologies

Qxw lets you construct variations on the plain rectangular grid where lights running off one edge of the grid reappear on the opposite side. If you join a pair of opposite edges directly, the result is equivalent to a circular grid as described above; if you join a pair of opposite edges 'with a flip', then the result is a grid with the topology of a Möbius strip; and if you join both pairs of opposite edges the result has the topology of a torus. More advanced possibilities available include Klein bottles (both pairs of opposite edges joined, one pair with a flip and the other pair without) and the projective plane (both pairs of opposite edges joined with a flip). These options are all available as 'Grid types' in the Grid properties dialogue that appears when you select the *Properties-Grid properties* menu item.

E	A	C	T	O	R	A	D	V	A	N	C
T	O	V	A	P	O	R	E	T	T	O	A
H	U	M	P	E	D	I	N	V	O	I	C
E	D	D	E	R	I	S	I	O	N	S	T
N	A	T	T	A	S	T	R	E	C	E	N
E	D	G	A	S	C	O	O	L	E	D	R

Figure: 4.9: Grid on a Möbius strip

Figure 4.9 shows an example grid with the topology of a Möbius strip: lights that reach the right-hand edge of the grid continue in the cell diametrically opposite on the left-hand edge.

Figure 4.10 shows an example grid with the topology of a torus. Here lights running off any edge reappear in the corresponding position on the opposite edge.

Note that with grid topologies that have joined edges (which includes the special case of a circular grid as mentioned above), it is possible to create unbounded lights. If Qxw cannot determine where a light should start and finish it will not attempt to fill it.

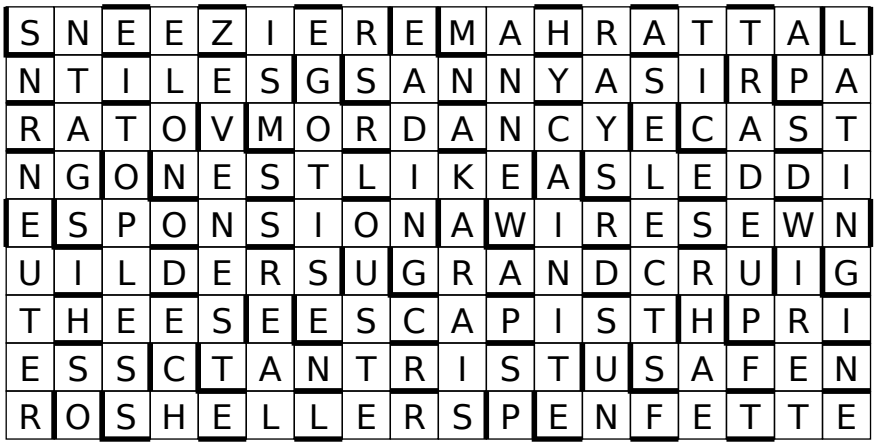

Figure: 4.10: Grid on a torus

4.6 Multiplex lights

A 'multiplex light' is one that can be filled with two or more alternative words. For example, suppose you wish to design a grid where one light can read 'BLACK' or 'WHITE', all entries being real words in either case. Visit the first cell of the five-cell light and, using *Edit-Cell contents* ('control-I'), set its contents to 'BW' (the two possible first letters). Repeat for the remaining four cells, setting their contents to 'LH', 'AI', 'CT' and 'KE' in turn. Finally, for this light and all others that intersect it, use the Light properties dialogue to enable the 'Multiplex light' setting. (If light properties are not being used for any other purpose in your grid, you can simply enable 'Multiplex light' in the Default light properties dialogue: the setting has no effect on lights where all cells contain a single letter.)

When the cursor is sitting on a multiplex light, the 'change direction' keys ('Page Up', 'Page Down' and 'slash') step through the alternative entries before moving on to the next direction; the green number next to the cursor increments and decrements as you cycle through the alternatives.

A quicker way to enter letters in multiplex lights is to use the Light contents dialogue (*Edit-Light contents*, 'control-L'), which lets you type in the text of each alternative in one step.

Figure 4.11 shows an example where the light across the centre of the grid can read 'BANKER'S CLERK' or 'HIPPOPOTAMUS' with all grid entries being real words in either case.

I	S	L	E	T	P	A	C	E	C	A	R
S	T	I	R	R	A	G	L	I	O	M	A
A	R	M	O	I	R	E	O	U	M	A	S
BH	AI	NP	KP	EO	RP	SO	CT	LA	EM	RU	KS
A	P	I	A	N	F	A	H	L	O	R	E
R	U	N	N	E	L	R	E	G	N	A	L
P	E	G	A	S	U	S	S	I	S	S	Y

Figure: 4.11: Grid with a multiplex light

Chapter 5

Grids with secrets

5.1 'Letters Latent'

In some advanced cryptic crossword puzzles, one or more of the clue answers are modified in some way before entry in the grid. Qxw calls such modification 'answer treatment'. Qxw will help you construct grid fills with a wide range of such treatments, including misprints, 'letters latent', various enciphering schemes, and many others. And, as we will see in Chapter 7, you can even construct your own answer treatments.

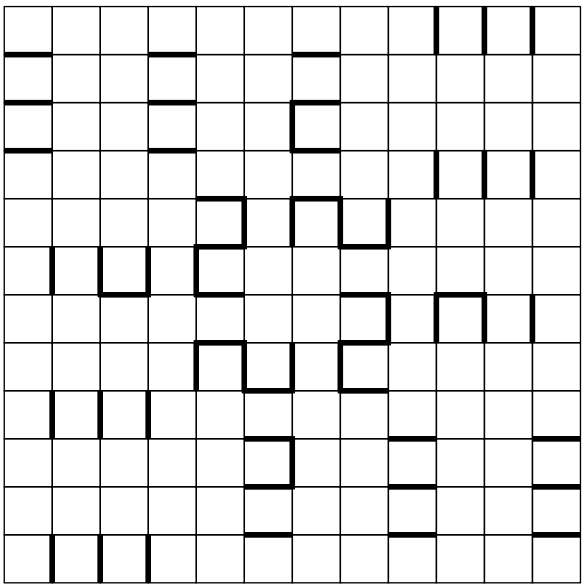

Figure: 5.1: Blank grid for 'letters latent' puzzle

In this section we will construct a puzzle using the 'letters latent' answer treatment. Such puzzles usually come with a preamble along the lines of 'one letter is to be omitted from the answer to each clue, wherever it occurs, before entry in the grid; in each clue the subsidiary indication leads to the grid entry'.

Thus a clue answer of 'QUINQUEREME' might give rise to a grid entry of 'UINUEREME', the 'Q' being latent. It is not usually a requirement that the grid entry must itself be a word, but often the sequence of latent letters, taken in clue order, provides a helpful or thematic message to the solver.

The puzzle we will construct will apply the 'letters latent' treatment to the across answers only.

Start from a blank 12-by-12 rectangular grid (as provided by Qxw when it starts up). Add bars to produce the diagram shown in Figure 5.1. Now tell Qxw which lights are to be treated. First select them: use the menu option *Select-Lights-in current direction* with the cursor pointing in the across direction. This will select all the across lights. Now choose the menu item *Properties-Selected light properties*, tick the boxes 'Override default light properties' and 'Enable answer treatment', and click 'Apply'.

You can deselect the lights now if you wish, either by pressing 'shift-N' or using the menu item *Select-Nothing*.

For more information on how to select lights see Chapter 11; for more about what you can do with light properties, see Chapter 12.

Finally we need to specify the details of the answer treatment. Bring up the Answer treatment dialogue by choosing the menu item *Autofill-Answer treatment*. At the top of the dialogue you can choose the desired method of answer treatment: select 'Letters latent: delete all occurrences of letter'. In the box marked 'Letters to delete' enter the message 'better a witty fool'. The result should look like Figure 5.2.

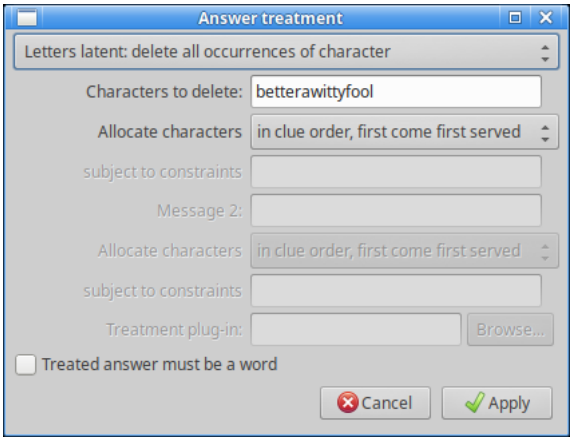

Figure: 5.2: Answer treatment dialogue

You can now proceed to fill the grid as normal. Figure 5.3 shows an example automatic fill. The first and last across lights were chosen manually.

5.2 Hidden words

Qxw includes a powerful feature called 'free lights'. These let you specify that some sequence of cells in the grid—not necessarily in a straight line or even adjacent to one another—constitutes an additional light that must be filled from a dictionary. The examples that follow give an idea of the range of effects that you can achieve using free lights, either on their own or in combination

A	L	I	N	G	R	O	O	K	S	T	U
M	I	D	D	L	A	M	R	I	C	A	N
F	L	O	R	U	I	C	A	N	A	R	S
B	L	I	T	T	L	I	N	G	T	A	T
I	I	S	O	Y	H	S	G	S	H	M	E
O	M	T	C	B	E	A	C	H	E	A	R
M	A	G	C	W	A	N	D	I	T	S	I
E	R	R	A	M	D	G	A	P	E	A	L
T	L	E	T	E	T	R	A	S	T	L	E
R	E	E	I	N	G	A	L	C	R	A	N
I	N	C	N	S	I	D	E	R	A	T	E
C	E	E	A	H	A	O	W	S	D	A	Y

Figure: 5.3: 'Letters latent' automatic fill

with other features of Qxw. Note that free lights cannot be multiplexed.

For a simple example, start with a plain five-by-five square grid. Move the cursor to the top left-hand corner and select the menu item *Edit-Free light-Start new*. An orange square will appear at the cursor position. Move the cursor one square diagonally down and right, and select the menu item *Edit-Free light-Extend selected* ('control-E'). An orange line will appear, showing the path of the free light you are constructing. Move the cursor one square diagonally down and right once more, and again select the menu item *Edit-Free light-Extend selected* ('control-E'). Repeat the process twice more to construct a free light that runs down the leading diagonal of the grid: see Figure 5.4.

Figure 5.5 shows an example fill of this grid: the word on the leading diagonal is 'QUAGS'.

5.3 Hidden quotations

A common thematic device is to arrange for a quotation to appear around the perimeter of the grid. In simple cases this is just a matter of entering the text of the quotation in the grid before embarking on the rest of the fill. However, it is often acceptable for the quotation to start at an arbitrary point on the perimeter and proceed either clockwise or anticlockwise. Qxw can take advantage of this flexibility to find a superior fill for the rest of the grid.

Figure 5.6 shows a grid with a single free light running around the perimeter. Suppose we want this light to spell out 'IN THE WINDMILLS OF YOUR MIND'. Three further steps must be taken before you can ask Qxw to fill this grid.

The first step is to ensure that Qxw does not also try to fill the seven-letter lights around the edges of the grid with individual words. To do this, select the four lights in question: place the cursor on each in turn, pointing in the direction of the light, and choose the menu item *Select-Current light* ('shift-L') for each. With all four lights selected, choose the menu item *Properties-*

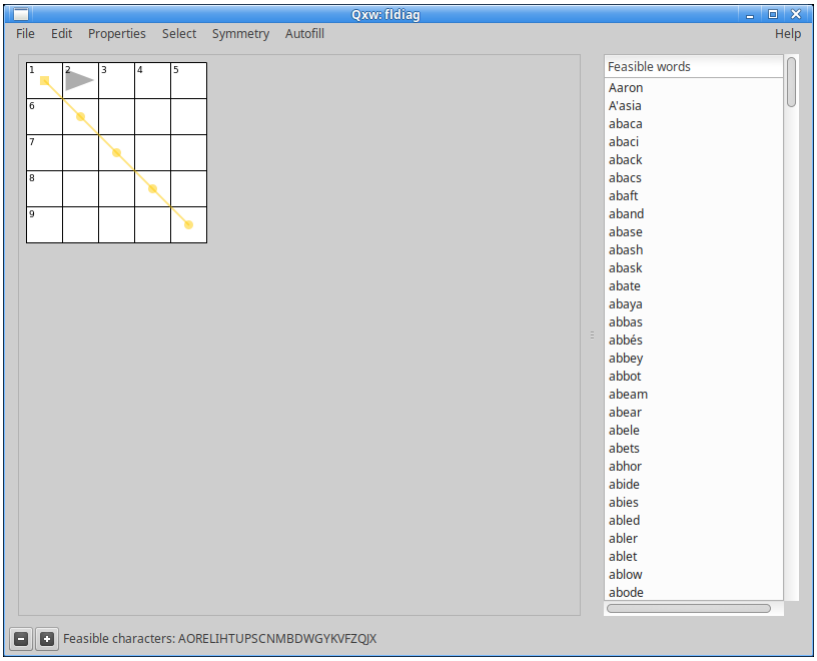

Figure: 5.4: Square grid with free light running down the leading diagonal

Q	B	O	A	T
O	U	T	D	O
P	R	A	A	M
H	E	R	G	E
S	T	Y	E	S

Figure: 5.5: Example fill of grid with free light

Selected light properties and in the dialogue that appears, tick the box 'Override default light properties' and untick all the 'Use dictionary' boxes. Qxw will not now try to fill these lights.

The second step is to make a dictionary containing the desired quotation. You can do this using an ordinary text editor to make a dictionary file with just one entry, but Qxw offers a short-cut. Choose the menu item *Autofill-Dictionaries*. In the second row, make sure the 'File' entry is blank and enter 'IN THE WINDMILLS OF YOUR MIND' (without the quotation marks) on the right under 'Answer filter'. This automatically creates a dictionary containing a single entry.

The third step is to set the properties of the free light. Select it by choosing the menu item *Select-Free light* ('shift-F') and then choose the menu item *Properties-Selected light properties*. Tick the boxes 'Override default light properties' and 'Use dictionary 2'; make sure all the other dictionary boxes are unticked. Also tick four of the 'entry method' boxes: 'Allow light to be entered normally', 'Allow light to be entered reversed', 'Allow light to be entered cyclically permuted' and 'Allow light to be entered cyclically permuted and reversed'.

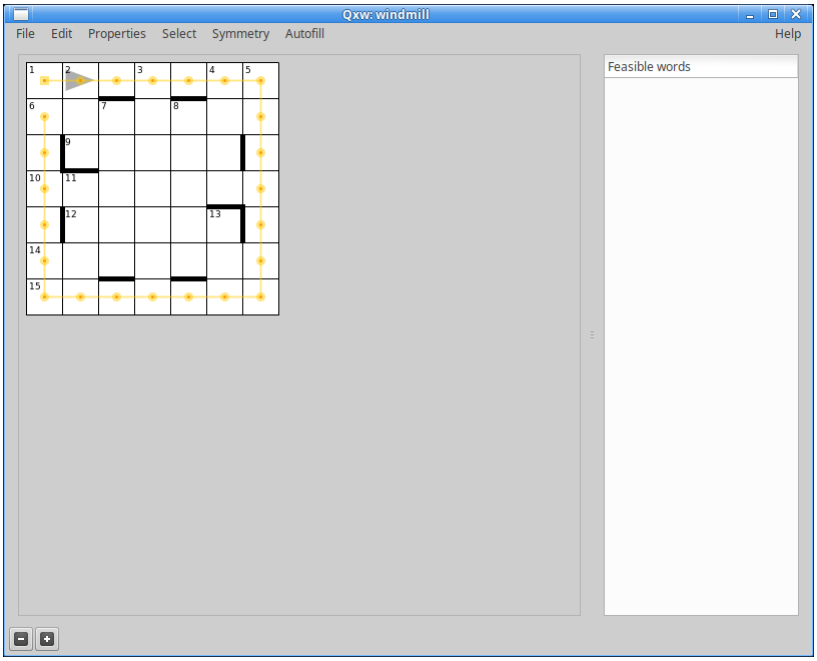

Figure: 5.6: Grid with free light running around perimeter

U	O	Y	F	O	S	L
R	E	P	O	S	A	L
M	R	O	O	K	S	I
I	D	O	L	I	S	M
N	A	L	I	E	N	D
D	R	E	S	D	E	N
I	N	T	H	E	W	I

Figure: 5.7: Filled grid with quotation running around perimeter

Click 'Apply' and you can try creating a fill: Figure 5.7 shows an example fill using a large dictionary for the body of the grid.

The maximum length of a free light (indeed, the maximum length of any light) is 250 characters.

5.4 'Cherchez la femme'

In a 'Cherchez la femme' puzzle Across and Down lights do not agree in certain grid cells. The letter to be entered in these cells is in each case some function of the two clashing letters: for example, it might be the letter appearing midway between the two clashing letters in the alphabet (treated cyclically), so that 'B' can arise from the clashes 'A/C', 'Z/D', 'Y/E', 'X/F',

'W/G' or 'V/H'—but not 'U/I', which would resolve to 'O'. Traditionally the clashes resolve to spell out, in grid order, a girl's name, although similar treatments are used for many different kinds of theme.

You can use free lights in conjunction with a specially-constructed dictionary to make such a puzzle using Qxw.

The dictionary contains three-letter 'words' such as 'ACB', where 'A' and 'C' are the clashing letters and 'B' the desired resolution. In simple cases (or if you have a lot of patience) such a dictionary can be constructed manually using an ordinary text editor, but it is easier to write a small program to do the task.

You can of course use any programming language you like: creating dictionaries is usually not a computationally intensive job, so interactive and interpreted languages such as Python or BASIC—or even a spreadsheet—are perfectly good choices. The example code shown in Figure 5.8 is an implementation in C of a program to create a 'Cherchez la femme' dictionary.

```
#include <stdio.h>
main() {int i,j;
   for(i=0;i<26;i++) for(j=1;j<7;j++) {
     printf("%c%c%c\n",(i+j+26)%26+'a',(i-j+26)%26+'a',i+'a');
     printf("%c%c%c\n",(i-j+26)%26+'a',(i+j+26)%26+'a',i+'a');
     }
   return 0;
   }
```

Figure 5.8: Program to create a 'Cherchez la femme' dictionary

Use an ordinary text editor create a file `mkclfdict.c` containing the program code shown. Linux users can compile it at the command line and run it as follows; compiling C programs under Windows is discussed in Section 14.4.

```
gcc mkclfdict.c -o mkclfdict    ./mkclfdict > clfdict.txt
```

This will create a file called `clfdict.txt` containing the 312 three-letter 'words' that represent possible clashes and their resolutions. You can implement different methods of clash resolution by making suitable modifications to the C program and then repeating the compilation and run commands.

Now you can construct the grid. It will be in two parts: the grid proper and a separate isolated area where the girl's name will appear. For simplicity we will construct a five-by-five grid with clashes down the leading diagonal, and we will make the clashes resolve to spell 'FEMME'. Figure 5.9 shows the grid with an isolated area at the bottom containing the word 'FEMME'.

To load the specially-constructed dictionary, choose the menu item *Autofill-Dictionaries*. The Dictionaries dialogue will appear. Click on the 'Browse' button in the second row and navigate to where you created the file `clfdict.txt` and open it. Then click on 'Apply': this will load your dictionary.

Now we select the five cells down the diagonal of the main grid: move the cursor to each in turn and choose the menu item *Select-Current cell* ('shift-C'). Now choose the menu item *Properties-Selected cell properties* and tick the 'Override default cell properties' option. At the bottom select 'Lights intersecting here need not agree (horizontal display)'. Click 'Apply'.

Move the cursor to the first selected cell and choose the menu item *Edit-Cell contents* ('control-

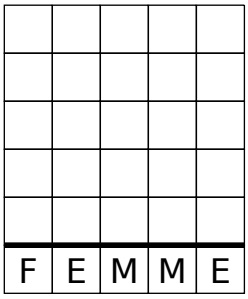

Figure: 5.9: Initial 'Cherchez la femme' grid

I'). In each of the two text boxes enter a single full stop ('.'): you will probably find that the first box is already correctly set up. Repeat for each selected cell.

You can now deselect the cells using the menu item *Select-Nothing* ('shift-N').

If you wish, you can try filling the grid now using the menu item *Autofill-Autofill* ('control-G'). The cells down the diagonal will be filled with (possibly) clashing letters, but the clashes will bear no relation to the theme word.

To make the clashes generate the theme word we create five free lights, one per clash. Move the cursor to the first clash cell (in the top left corner) and choose the menu item *Edit-Free light-Start new*. An orange square will appear at the cursor position. Move the cursor to the first letter of the theme word (the 'F' of 'FEMME') and choose the menu item *Edit-Free light-Extend selected* ('control-E'). An orange line will appear. Repeat these steps four times, starting a new free light in each clashing cell and extending it to the corresponding letter of the theme word.

Select all five free lights: choose the menu item *Select-Free light* ('shift-F') and then *Select-All* ('shift-A'). The grid should look like Figure 5.10.

Select the menu item *Properties-Selected light properties* and tick the boxes 'Override default light properties' and 'Use dictionary 2'; make sure that the other 'Use dictionary' boxes are not ticked. Click 'Apply'.

The free lights you have created now have to be filled from the special dictionary. Each free light gets its first two characters from the clashing entries in its main grid cell and its third character from the corresponding cell in the theme word. This therefore enforces the constraint we need.

Assuming you have a reasonably large main dictionary, choosing the menu item *Autofill-Autofill* ('control-G') will now create a grid with the clashes as required.

Figure 5.11 shows an example fill. Figure 5.12 shows an example fill of a six-by-six grid, and Figure 5.13 shows a fill of a six-by-six grid where the clashes are resolved by 'adding' the clashing letters (with $A = 1, B = 2, \ldots, Z = 26$) and treating the alphabet cyclically.

If you are not constrained to use a particular name you can of course leave the choice to Qxw by providing it with a suitable dictionary containing a list of girl's names, which you can make up yourself or obtain from any number of public sources. You would then set the light properties on the isolated light to ensure it is filled from this dictionary.

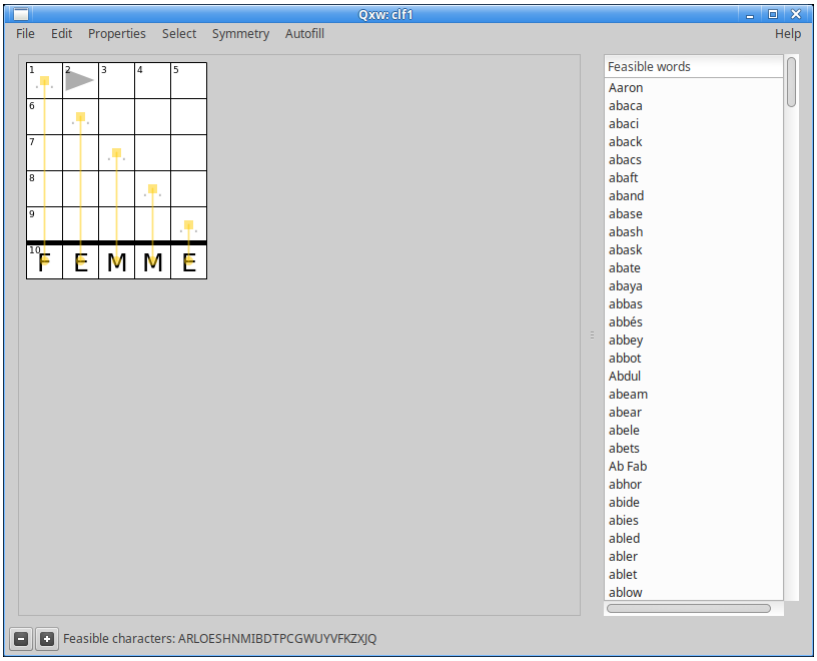

Figure: 5.10: 'Cherchez la femme' grid with free lights added

CI	L	A	I	M
N	AI	U	R	U
G	A	LN	A	S
A	N	T	IQ	S
N	A	S	I	KY
F	E	M	M	E

Figure: 5.11: Automatically-filled 'Cherchez la femme' grid

5.5 'Eightsome reels'

An 'eightsome reels' grid consists of eight-letter words. Each is entered cyclically around a blacked-out square. The starting point of each word and its direction of entry (clockwise or anticlockwise) can be freely chosen by the setter.

5.5.1 Creating the grid manually

It is easy (if tedious) to set up such a grid in Qxw manually. For a small example, start with a blank seven-by-seven grid and black out nine squares as shown in Figure 5.14.

Now prevent Qxw from trying to fill the seven-letter lights. Select all the lights: *Select-Current light* ('shift-L') and then *Select-All* ('shift-A'). Now use *Properties-Selected light properties*, tick

PH	I	D	O	G	S
E	UG	O	U	A	E
F	U	NL	R	U	N
T	A	M	IA	L	S
E	N	E	R	VT	E
D	A	N	I	S	HD
L	A	M	E	U	F

Figure: 5.12: Automatically-filled 'Cherchez la meuf' grid

TR	H	A	N	K	S
E	LO	U	E	N	T
S	O	ID	R	E	E
O	L	I	VI	I	A
R	I	O	T	EP	D
B	E	S	E	E	MS
L	A	M	E	U	F

Figure: 5.13: Automatically-filled 'Cherchez la femme' grid with clashes resolved by adding letters

the box 'Override default light properties' and untick all the 'Use dictionary' boxes.

Next create an eight-letter free light around each of the nine blacked-out squares. Once that is done, select all the free lights using *Select-Free light* ('shift-F') and then *Select-All* ('shift-A'). Use *Properties-Selected light properties* and tick the four 'entry method' boxes: 'Allow light to be entered normally', 'Allow light to be entered reversed', 'Allow light to be entered cyclically permuted' and 'Allow light to be entered cyclically permuted and reversed'.

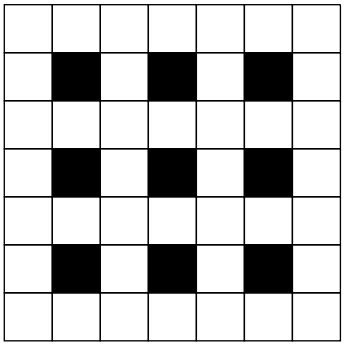

Figure: 5.14: Starting pattern for 'eightsome reels' grid

Click 'Apply' and you can try creating a fill: Figure 5.15 shows an example result.

A	D	E	T	H	A	I
E	■	L	■	E	■	R
D	I	A	I	R	A	C
A	■	S	■	D	■	E
R	O	P	O	R	A	C
D	■	E	■	T	■	E
N	E	S	I	O	I	R

Figure: 5.15: Example fill of 'eightsome reels' grid

5.5.2 Creating the free lights automatically

The tedious step in the above method is the creation of the free lights, and it would have been yet more tedious if we had started with a larger grid.

An alternative approach is to use an external program to create a text file specifying the free lights. The file should consist of a sequence of coordinate pairs, one pair per line.

The two elements of each coordinate pair (horizontal then vertical, or angular then radial in the case of circular grids) must be separated by a space. Coordinates are counted from zero at the top left of the grid (or top centre for circular grids). The sequence of coordinates comprising one free light must be separated from those of the next by a blank line.

The free lights we need for the 'eightsome reels' grid can be created using a simple program such as Figure 5.16, which is written in the C programming language; almost any programming language is suitable for this kind of task.

You can compile and run this program as described above for the 'cherchez la femme' dictionary. Collect its output in a file called `eightsome.txt` and then import this file into Qxw using the menu item *File-Import free light paths*.

There is also a menu item *File-Export free light paths* that writes a text file in the above format containing the paths of the currently-defined free lights. You can edit this file using a text editor and then import it back into Qxw: in more complex situations this may be more convenient than editing the paths within Qxw.

5.6 'Alphabetical jigsaw'

In an 'alphabetical jigsaw' puzzle the lights start in twenty-six distinct cells, each containing a different letter of the alphabet. Except where an across and a down light start in the same cell, each light therefore starts with a different letter. Conventionally the clues are presented in alphabetical order of their answers, the solver being required to determine where they fit.

To create such a puzzle, first design a suitable blank grid. Usually a fifteen-by-fifteen blocked grid is used, but other types are possible. To have Qxw fill the grid, create a free light running

```
#include <stdio.h>
int main() {
  int x,y;
  for(x=1;x<=5;x+=2) {
    for(y=1;y<=5;y+=2) {
      printf("%d %d\n",x-1,y-1); // NW corner
      printf("%d %d\n",x,y-1); // N
      printf("%d %d\n",x+1,y-1); // NE corner
      printf("%d %d\n",x+1,y); // E
      printf("%d %d\n",x+1,y+1); // SE corner
      printf("%d %d\n",x,y+1); // S
      printf("%d %d\n",x-1,y+1); // SW corner
      printf("%d %d\n",x-1,y); // W
      printf("\n"); // blank line
      }
    }
  return 0;
  }
```

Figure 5.16: Program to create 'eightsome reels' free lights

through the light starting cells (of which there must be exactly twenty-six). Create a single-entry dictionary containing the word 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' using the same technique as in Section 5.3, and set the properties of the free light so that it is filled from this dictionary with all entry permutations allowed. You can now use the automatic fill function as usual: a possible result is shown in Figure 5.17.

Figure: 5.17: Automatically-filled ‘alphabetical jigsaw’ grid

Chapter 6

Discretion

In the 'letters latent' example in Chapter 5 we explicitly selected which lights were to be treated specially by the automatic filler before putting it to work on the grid. Sometimes, however, you will have some flexibility in this choice: for example, the length of the message to be spelt out by misprints in the grid may be less than the number of lights, and so you would choose to leave some lights untreated. In other cases, for example if the grid is to be presented as a *'carte blanche'*, you might be happy for the characters of a message to be spelt out in any order.

In these situations it would be greatly preferable if you could leave to the automatic filler the job of choosing which lights to treat and which not to treat, or of deciding upon the order in which the lights should be treated. Qxw can handle both these cases, and even the two in combination, using the filler's 'discretionary modes'. Not all the built-in answer treatments allow the use of these modes.

Start from a blank six-by-six grid. Enable answer treatment for all lights under *Properties-Default light properties*. Call up the Answer treatment dialogue *Autofill-Answer treatment* and select 'Variable Caesar cipher'. (See Chapter 14 for information about this treatment.) Against 'Encodings of A:' enter 'JULIUS' and beneath that, set the letter allocation to 'in clue order, at discretion of filler'. Click 'Apply' and run the filler with *Autofill-Autofill* ('control-G'). If you have a reasonably large dictionary, you should get a result similar to Figure 6.1. Here the filler has chosen to treat the second and last across lights, and the first, third, fourth and last down lights, using the letters of 'JULIUS' in order.

T	E	R	M	L	Y
Y	N	C	J	A	M
S	T	O	U	T	S
L	O	R	C	H	A
F	I	C	H	E	S
W	L	O	M	N	U

Figure: 6.1: Grid filled with discretion

When you press *Autofill-Accept hints* ('control-A') to accept the filler's suggestions a box will

appear telling you that the 'answer treatment constraints' have been updated. The answer treatment constraints, which are shown in the Answer treatment dialogue beneath the allocation method, record the filler's decisions about which lights to treat. Initially we left this box blank, but if you now call up the Answer treatment dialogue you will see that it has changed: for the example here the box would be changed to read '-J---UL-IU-S'. In this string there is one character for each light with answer treatment enabled (twelve in total). The string shows which message characters have been allocated to which lights, with a dash ('-') indicating that the filler decided not to treat this light after all.

If you now clear the grid using the *Edit-Clear all cells* ('control-X') command you will be prompted 'Reset answer treatment constraints?'. If you choose 'Yes' the constraints box in the Answer treatment dialogue will be cleared; if you choose 'No' the contents of the box will be preserved, and reinvoking the filler will use the same allocation of characters. You can also change this string manually if you wish.

If instead of selecting 'in clue order, at discretion of filler' you had selected 'in any order, at discretion of filler' you could have obtained a fill similar to the one shown in Figure 6.2. In this case the answer treatment constraints were updated to read 'UI---S-J--UL', reflecting the permutation of the message chosen by the filler.

M	C	W	E	Y	L
A	X	I	L	M	A
M	A	N	A	N	A
M	A	G	I	L	P
E	R	E	N	O	W
E	M	D	E	M	D

Figure: 6.2: Another grid filled with discretion

Chapter 7

Creating a customised answer treatment

Qxw's range of answer treatments includes most of those commonly found in advanced puzzles, but you are not limited to the built-in selection. In this chapter we will create a puzzle where answers are 'beheaded'—in other words, where an answer has its first letter removed to make the light.

7.1 Compiling a simple plug-in

Expressing a new answer treatment method to Qxw involves writing a program, called a 'plug-in'. The job of the program is to generate the possible lights that can arise from a candidate answer. To create the plug-in you will need to have a little experience with using the command line and you will need to make sure that you have the `gcc` C compiler installed. (There is an alternative approach under Windows: see Section 14.4.) But don't worry if you have not programmed in C before: in most cases the program will be very short indeed and easy to understand, and writing a plug-in makes an ideal gentle introduction to C programming.

The examples in this chapter assume that your plug-in only has to deal with the letters 'A' to 'Z' of the default Roman alphabet. Writing a plug-in to deal with other alphabets can be a more complicated proposition: see Chapter 14 for full details.

```
#include "qxwplugin.h"
int treat(const char*answer) {
   strcpy(light,answer+1);
   return treatedanswer(light);
   }
```

Figure 7.1: Plug-in code to remove the first letter of an answer to make a light

Figure 7.1 shows a program that 'beheads' answers. It consists of a single function, called `treat`. The work is done by the `strcpy()` command, which copies the answer string with an offset of one character (hence '`answer+1`') to the light.

Using an ordinary text editor create a file `behead.c` containing the program code shown. Under Linux, you can compile it at the command line as follows. (Under Windows, the situation is slightly more complicated: see Section 14.4.)

```
gcc -fPIC behead.c -o behead.so -shared
```

This creates the compiled plug-in `behead.so`, which Qxw can use.

Now, in Qxw, recreate the simple blank grid of Figure 1.3. There are two further steps to make Qxw use your new plug-in. First, select the menu item *Autofill-Answer treatment*, bringing up the Answer treatment dialogue. At the top, select 'Custom plug-in' (Figure 7.2). Now click 'Browse' next to 'Treatment plug-in' to locate the plug-in file `behead.so`, and then click 'Apply'.

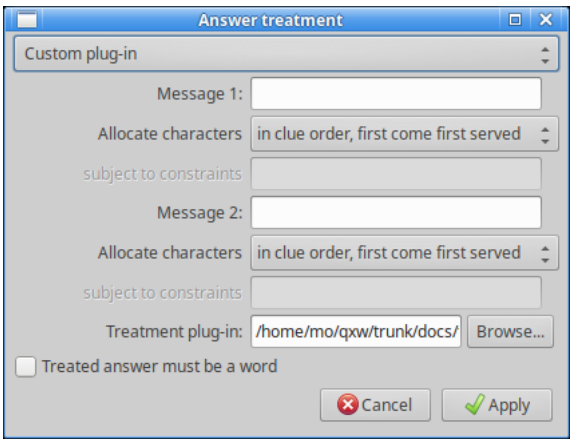

Figure: 7.2: Selecting a custom plug-in in the Answer treatment dialogue

Second, as with the built-in answer treatments, you need to tell Qxw which lights are to be treated. For this example, we will have all lights subject to treatment: select *Properties-Default light properties* and tick the 'Enable answer treatment' box. Now, if you select *Autofill-Autofill*, you should get a result similar to that in Figure 7.3.

A	T	E	R	E	D	■
T	■	T	■	C	■	A
E	A	T	H	E	R	S
D	■	L	■	N	■	E
■	R	E	A	T	H	S

Figure: 7.3: Grid with all lights being 'beheaded' words

You can force all lights to be words: select *Autofill-Answer treatment* and tick the 'Treated answer must be a word' box. (This option is available for all answer treatments, although using it will of course sometimes constrain the grid so much that it cannot be filled.) A fill might then look like Figure 7.4.

It is possible to write answer treatment plug-ins that make use of messages, just like the built-in 'Letters latent' or 'Misprint' plug-ins. Figure 7.5 shows a simple example.

Figure: 7.4: Grid with all lights being 'beheaded' words that are themselves words

```
#include "qxwplugin.h"
int treat(const char*answer) {
  if(answer[0]!=msgcharAZ09[0]) return 0;
  strcpy(light,answer+1);
  return treatedanswer(light);
  }
```

Figure 7.5: Plug-in code to behead answers so that deleted letters in clue order yield a message

This example checks that the character that is about to be deleted matches the appropriate letter of the message before allowing it as a treated answer. Figures 7.6 and 7.7 show grids filled using this plug-in.

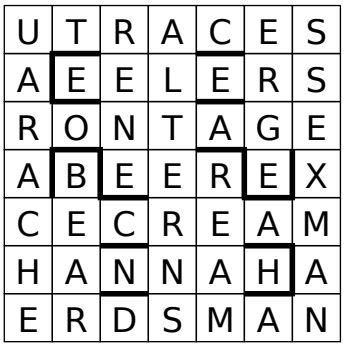

Figure: 7.6: Grid with all lights being 'beheaded' words, with deleted letters in clue order spelling out 'OFF WITH HIS HEAD'

Section 14.3 gives a full description of what is possible, including a discussion of how to write a plug-in where two or more different lights can arise from a single answer.

7.2 Combining plug-ins and discretion

You can use your plug-in in conjunction with the discretionary modes of the filler described in Chapter 6. There are certain restrictions you must observe: see Section 14.3.3 for details.

Consider the modification to the code in Figure 7.5 shown in Figure 7.8. This code makes use of both messages via the variables `msgcharAZ09[0]` and `msgcharAZ09[1]`. The intention is that

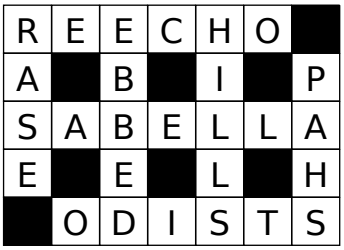

Figure: 7.7: Grid with all lights being 'beheaded' words that are themselves words, with beheadings in clue order spelling out 'PIMENTO'

the second message consists only of the digits 0 and 1, and that both messages are used 'in clue order, at discretion of filler'.

```
#include "qxwplugin.h"
int treat(const char*answer) {
  if(msgcharAZ09[0]=='-' && msgcharAZ09[1]=='-') return treatedanswer(answer);
  if(lightdir==0 && msgcharAZ09[1]!='0') return 0;
  if(lightdir==1 && msgcharAZ09[1]!='1') return 0;
  if(answer[0]!=msgcharAZ09[0]) return 0;
  strcpy(light,answer+1);
  return treatedanswer(light);
  }
```

Figure 7.8: Plug-in code to behead answers with control over distribution of answers

There are several tests in the code. The effects of these tests are: a character will be used from message 0 if and only if a digit is used from message 1; if no digit is used from message 1 then the answer is used untreated as the light; a '0' digit is used from message 1 only in across lights and a '1' only in down lights; and where a character is used from message 0 it must match the first character of the light, which is then removed.

Now, for example, if this plug-in is used with a grid with twelve across lights and twelve down lights, with message 0 set to 'HALFWAYHOUSE' and message 1 set to '000000111111' (six zeros and six ones), then the result will be that six (i.e., exactly half) of the across lights and six (again, exactly half) of the down lights will be treated, and the deleted characters will spell out 'HALFWAYHOUSE'. An example fill is shown in Figure 7.9.

This idea can be extended to apply different answer treatments to different lights depending on their direction, position or other characteristics, with as much of the decision-making as you wish left to the automatic filler.

Figure: 7.9: Grid with exactly half of all across lights and half of all down lights being beheaded, the beheadings in clue order spelling out 'HALFWAYHOUSE'; all lights are words

Chapter 8

Puzzles using digits, accents and non-Roman characters

8.1 Numerical puzzles

Qxw can fill grids with digits instead of, or as well as, letters. Figure 8.1 shows an automatically-filled example, constructed using two custom dictionaries and setting light properties to allow reverse entry.

6	4	1	4	1	9
3	2	0	0	0	2
1	7	0	7	4	9
7	2	4	4	4	8
9	4	8	0	0	7
1	8	9	3	9	1

Figure: 8.1: Grid with each across light being a prime or the reverse of a prime, and each down light being a square or the reverse of a square

Digits can be used in a conventional word-based puzzle as proxies for special characters or for other thematic purposes. Figure 8.2 shows a simple example of what can be done. Again, a special-purpose dictionary was used to create this grid.

8.2 Accents and non-Roman characters

Although by default Qxw starts up prepared to fill grids with the letters A to Z and digits, it can be configured to accept accented characters, non-Roman characters and a wide range of symbols. Qxw comes with a number of predefined alphabet configurations to support various languages, or you can create your own alphabet configuration: see Chapter 9 for details. Note,

P	O	S	T	P	1	D
A	■	H	■	H	■	I
R	E	A	S	1	R	S
A	■	R	■	R	■	H
G	R	E	Y	S	T	1
1	■	B	■	I	■	S
D	O	1	S	N	U	T

Figure: 8.2: Grid containing a mixture of letters and digits

however, that Qxw can only cope with up to 60 different symbols (plus blank) in the grid at once, and so cannot handle languages such as Chinese whose writing system uses a very large number of symbols, or the Hangul writing system, which combines vowels and consonants into syllabic blocks.

N	E	O	O	Σ	I	E	T	M	Σ	Y
A	Y	O	A	T	Σ	K	N	E	I	I
K	E	K	Θ	E	O	N	A	E	O	Λ
Σ	Λ	A	I	T	M	E	Ω	K	P	O
T	A	Y	T	P	O	Σ	O	Δ	Π	E
H	K	Λ	N	Ω	I	Σ	I	I	H	O
A	Δ	E	Ω	Σ	T	Π	N	O	T	P
Λ	A	N	T	O	Π	A	Σ	O	E	Π
Y	I	M	Σ	Π	K	O	E	O	N	A
E	E	Σ	A	O	P	M	N	N	A	Γ
Θ	I	E	Π	E	A	H	Y	I	I	Σ

Figure: 8.3: Grid filled with jumbles of words from the works of Herodotus: identifying the unjumbled forms is left as an exercise for the reader

To use a different alphabet select *Autofill-Alphabet* to bring up the Alphabet dialogue (see Figure 9.2) and click on 'Initialise from language default'. You will be presented with a list of the built-in alphabet configurations, which includes Czech, Danish, Dutch, Estonian, Finnish, French and Italian, German, Ancient Greek (polytonic), Modern Greek (monotonic), Hungarian, Norwegian, Polish, Romanian, Russian, Slovenian, Spanish and Swedish, plus 'Empty' (which you can use to create a new configuration from scratch). Select one of these and the boxes below will be populated with the appropriate settings.

Of course, in order to use the automatic filling facilities of Qxw you will need a dictionary that matches the alphabet you are using. The alphabet configuration includes information that

lets Qxw process a dictionary file correctly when it is loaded. For example, the handling of 'Ö' differs between Finnish, German and Hungarian: in Finnish it is a letter in its own right, distinct from the unaccented version; in German crosswords (but not in German Scrabble!) it is conventionally expanded into the sequence 'OE'; and in Hungarian crosswords it is a letter in its own right, allowed to check 'Ő' but not 'O' or 'Ó'. See Section 9.6 and Section 9.7 for more details. Figure 8.3 shows an example prepared using a special dictionary file.

You will also need to know how to generate the symbols you want using your keyboard. If you need to enter a symbol that is not directly supported by your keyboard you can use the Cell contents dialogue (see Section 12.3) or you may be able to use a 'Character Map' or similar application.

It is possible to create a customised alphabet to produce effects like that illustrated in Figure 8.4. Here an alphabet consisting of all the symbols required to write both English and Greek words is used, with appropriate mappings to allow English and Greek words to check where their letters can be represented by the same symbol, even if they do not represent similar sounds. Again, see Section 9.7 for more details.

Ρ	Ε	Μ	Π	Ε	Λ	Ε	Ψ	Α
Η	■	Υ	■	C	■	Χ	■	S
Τ	Ε	Χ	Ν	Ο	Δ	Ο	Μ	Η
Η	■	Ο	■	Ν	■	Ε	■	Τ
Α	Θ	Ε	Τ	Ο	Υ	Ν	Τ	Α
L	■	D	■	Μ	■	Ζ	■	R
Α	Ν	Ε	Π	Ι	Λ	Υ	Τ	Ο
Τ	■	Μ	■	S	■	Μ	■	Τ
Ε	Π	Α	Ν	Ε	Τ	Ε	Θ	Η

Figure: 8.4: Horizontal words are Greek; vertical words are English

8.3 Answer treatments using non-Roman alphabets

As far as possible Qxw adapts its answer treatment mechanisms to suit different alphabets: see Section 14.1 for more details.

Qxw's handling of non-Roman alphabets has been designed to be backwards-compatible with previous versions of the program. Existing plug-ins will still work when using the 'Roman A-Z', 'Digits only 0-9' and 'Roman plus digits A-Z 0-9' alphabets (and indeed with any alphabet that only uses characters from the 7-bit ASCII set).

Many plug-ins can be adapted very simply to work with non-Roman alphabets thanks to Qxw's use of 'internal character codes' (ICCs), which can be handled exactly like ordinary C characters; and strings of ICCs can be handled exactly like ordinary C strings. So for example the 'beheading' plug-in shown in Figure 7.1 can be modified as shown in Figure 8.5 and will then work with any alphabet.

```
#include "qxwplugin.h"
int treat(const char*answer) {
   strcpy(light,answerICC+1);
   return treatedanswerICC(light);
   }
```

Figure 8.5: Plug-in to behead an answer to make a light, compatible with any alphabet

It is also possible to write your plug-in in terms of UTF-8 or Unicode character encodings: for full technical details see Section 14.3.1.

Part II

Reference

Chapter 9

Dictionaries and alphabets

When Qxw starts it looks for lists of words in various places on your system. If not otherwise configured, it will try to find a suitable list and load it to use as its dictionary for automatic filling. Often the first word list it finds is designed for use with a spell checker and this is not always ideal for constructing crosswords: in particular it may contain many abbreviations, proprietary names, and words ending in apostrophe–s.

You can change this behaviour by setting a 'default dictionary' in the Preferences dialogue: see Section 10.1 for more details. Qxw will then attempt to load this dictionary as its default from the next time you start it up. You can also configure an accompanying file filter and answer filter: see Section 9.2.

If you are setting a different default startup dictionary you may also want to change the default alphabet on startup: see Section 9.6.

The dictionary used at startup can also be set from the command line using the `-d` option under Linux; under Windows the invocation is similar, with suitable changes to reflect the syntax of Windows path names. A dictionary specified from the command line takes priority over one specified in the Preferences dialogue.

To change the dictionary configuration after startup use the menu item *Autofill-Dictionaries*. This will open the Dictionaries dialogue (see Figure 9.1). Ignore everything except the top-left corner for the moment: either enter the full filename in the topmost box under 'File', or click on the topmost 'Browse' button and navigate to the word list file you want to use. Then click 'Apply'. (You will get an error message at this point if, for example, the specified word list file does not exist.) If you are using the default Roman alphabet, Qxw will automatically construct a dictionary by removing all punctuation, accents and spaces from the words in the list; see Section 9.6 for information about how other alphabet configurations behave.

When you now use the autofill feature you should see that the new dictionary is being used.

Qxw is normally used with dictionaries stored as plain text files. However, it also includes a feature that allows it to attempt to read dictionaries stored in 'TSD0' and 'TSD1' formats, which conventionally have a '.TSD' file extension. Since these file formats are not properly documented, there can be no guarantee that Qxw will read such dictionaries correctly.

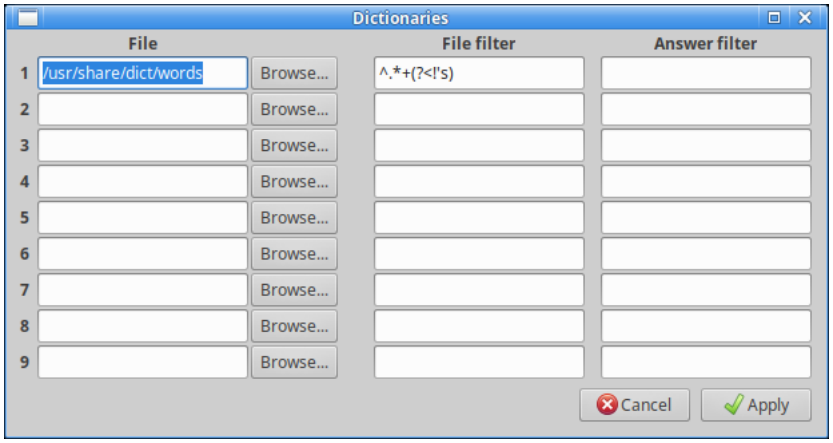

Figure: 9.1: The Dictionaries dialogue

9.1 Using multiple dictionaries

Qxw can handle up to nine dictionaries at once. Each light in the grid can be drawn from any combination of these dictionaries (see Chapter 12). The word list files used can be specified at the command line using repeated `-d` options or using the Dictionaries dialogue as above, entering one filename in the leftmost box of each row.

By default Qxw will only use the first dictionary to fill normal grid entries. To change this, use the Light properties dialogues (see Chapter 12). If you add a new dictionary and no lights are currently configured to use it, Qxw will display a warning.

9.2 Customising the dictionaries

The contents of a dictionary can be customised using the filters provided in the Dictionaries dialogue. There are two filters associated with each dictionary. By default these filters are blank, which means that all the entries in the original word list are accepted. However, by entering a simple matching pattern, a custom dictionary can be created containing only words with a certain property. The filter syntax is that of 'Perl compatible regular expressions' or PCREs. See http://en.wikipedia.org/wiki/PCRE for more details, and there are some useful examples at http://en.wikipedia.org/wiki/Regular_expression; the most authoritative source of information is at http://www.pcre.org/.

The first filter, the 'File filter', acts on the lines of text in the original word list file. Only lines that match the given PCRE are processed further; others are rejected. The primary use of the file filter is to help eliminate undesirable entries from the dictionary. One situation you might encounter is that you have a dictionary file that includes comment lines whose first character is hash ('#'). To have Qxw ignore these lines when reading the file, set the 'File filter' to read

```
^[^#]
```

The word list found on many systems at `/usr/share/dict/words` often includes many entries ending apostrophe–s, which are not suitable for use in a crossword grid. They can be removed

by specifying a file filter that reads

```
^.*+(?<!'s)
```

Qxw automatically applies exactly this filter when trying to load words from a default system dictionary (but not if you specify this dictionary's name explicitly on the command line).

The second filter is called the 'Answer filter'. This acts on the string of characters obtained from the line in the word list file after punctuation, accents and spaces have been removed in accordance with the current alphabet configuration. Only answers that match the PCRE are accepted for potential use in grid filling; others are rejected. The answer filter provides an easy way to create a crossword with a simple theme. Here are some examples; consult the documentation mentioned above for the full range of possibilities.

To create a dictionary where:	**set the answer filter to:**
every entry starts in 'e'	`^e`
no entry starts in 'e'	`^[^e]`
every entry ends in 'e'	`e$`
no entry ends in 'e'	`[^e]$`
every entry contains an 'e'	`e`
no entry contains an 'e'	`^[^e]*$`

Both the file filter and the answer filter are insensitive to case.

The same source word list can be used in conjunction with different filters to make two or more different dictionaries. Using this in conjunction with 'Light properties' (see Chapter 12) you can construct a grid where (for example) no across answer contains the letter 'e' and each down answer contains the letter 'q'.

9.3 Single-entry dictionaries

In some situations it is useful to construct a dictionary with a single entry. Although it is straightforward enough to use a text editor to create a suitable file (see below) there is an even easier way: leave the 'File' field in the Dictionaries dialogue blank, and enter the desired word under 'Answer filter'.

9.4 Making dictionaries using external tools

Since Qxw's dictionaries are simple plain text files with one entry per line, you can create your own using any ordinary text editor (be sure to save the result as 'plain text', with UTF-8 encoding if you are offered a choice). You can use any of the standard Linux command-line text processing utilities such as `grep`, `awk` and `perl`, or Windows Notepad. For a Linux example, to create a dictionary `vowel` containing just those entries in `ukacd18` that start with a vowel, type:

```
grep "^[aeiouAEIOU]" <ukacd18 >vowel
```

9.5 Dictionary file encodings

Qxw can read dictionaries stored as plain text files. It will make reasonable efforts to determine if your file is encoded using UTF-8, UTF-16 (big-endian or little-endian), UTF-32 (big-endian

or little-endian) or eight-bit ISO/IEC 8859-1, and should almost always work correctly in these cases.

However, there is a small chance that your dictionary file, especially if it was originally prepared under an early version of the Windows operating system, will be encoded differently, typically in an eight-bit format according to a particular 'code page'. Qxw cannot reliably detect automatically exactly what encoding is used in these cases, and will most likely treat the text as if it were encoded according to ISO/IEC 8859-1. The result will be that some characters—in particular accented and non-Roman characters—will be processed incorrectly. To use such a file with Qxw you should convert its format to UTF-8.

Under Windows you can load your file (having made a backup copy!) into Notepad, verify that it is displayed correctly, and then use 'Save As...'. This will give you an 'Encoding' option which you should set to 'UTF-8'. Save the file under a new filename, and you should find that Qxw will read the new file correctly.

The free program Notepad++ also has built-in conversions from a wide range of character encodings.

It is relatively rare to find plain text files on Linux systems encoded using anything other than UTF-8; but if you do need convert the encoding of a file, the `iconv` utility is very versatile. An example invocation to convert a file from code page 1250 (used for Central and Eastern European languages based on a Roman alphabet) to UTF-8 is as follows.

```
iconv -f CP1250 -t UTF-8 <inputfile >outputfile
```

9.6 Alphabets

Qxw supports the creation of crosswords using symbols beyond just the letters A–Z and the digits 0–9.

The alphabet that Qxw uses can be selected using the Alphabet dialogue (*Autofill-Alphabet*). It can also be configured at startup by setting a 'default alphabet' in the Preferences dialogue: see Section 10.1 for more details. Qxw will then use this alphabet as its default from the next time you start it up. You can also configure a default dictionary at startup.

Another way to configure the alphabet in use at startup is from the command line, using the `-a` option followed by one of the names (without quotation marks) specifying an alphabet according to the following table.

Alphabet	**Names**
Roman A-Z	`"Roman", "AZ"`
Digits only 0-9	`"Digits", "09"`
Roman plus digits A-Z 0-9	`"RomanDigits", "AZ09"`
Czech	`"Czech", "CZ"`
Danish and Norwegian	`"Danish", "Norwegian", "DA", "NO"`
Dutch	`"Dutch", "NL"`
Estonian	`"Estonian", "EE"`
Finnish	`"Finnish", "FI"`
French and Italian	`"French", "Italian", "FR", "IT"`
German	`"German", "DE"`

Alphabet	Names
Ancient Greek (polytonic)	`"AncientGreek", "AG"`
Modern Greek (monotonic)	`"ModernGreek", "MG", "EL"`
Hungarian	`"Hungarian", "HU"`
Polish	`"Polish", "PL"`
Romanian	`"Romanian", "RO"`
Russian	`"Russian", "RU"`
Slovenian	`"Slovenian", "SI"`
Spanish	`"Spanish", "SP", "ES"`
Swedish	`"Swedish", "SE"`
Empty	`"Empty", "Null"`

An alphabet specified from the command line takes priority over one specified in the Preferences dialogue.

9.7 Custom alphabets

It is possible to create your own custom alphabet, for example to support a language not included in the built-in set, or to allow special symbols to appear in the grid. Qxw uses the Unicode system to represent characters: this is an extension of ASCII (American Standard Code for Information Interchange) that assigns a number (called a 'code point') to virtually every symbol used in virtually every language of the world, living or dead. Unicode code points are conventionally written 'U+' followed by the number in hexadecimal, so for example U+0041 represents capital 'A' and U+03A9 represents Greek capital omega. You may find an application called something along the lines of 'Character Map' on your computer, which will let you explore the characters available to you and show you their Unicode code points. The most authoritative source of information on Unicode can be found at http://www.unicode.org and the code charts are indexed at http://www.unicode.org/charts/.

Except as noted below, any printable symbol in the first three Unicode planes (that is, with code points up to 0x2FFFF inclusive) can be used in Qxw. These are the basic multilingual plane, the supplementary multilingual plane, and the supplementary ideographic plane.

Under Windows, Qxw supports Unicode version 6.0; the version supported under Linux depends on the versions of libraries present in the system, but for a reasonably recent installation will typically be at least version 10.0.

The following characters, which have various special meanings in Qxw, may not be used: vertical double quotation marks (U+0022); hash (U+0023); comma (U+002C); hyphen (U+002D); full stop (U+002E); question mark (U+003F); commercial at (U+0040); square brackets (U+005B and U+005D); caret (U+005E).

Specifying an alphabet involves more than just giving a list of symbols that can go in the grid. For each possible symbol that can appear in a grid entry, Qxw also needs to know what characters in dictionary files can map to that symbol, whether it is considered a vowel or a consonant, and whether it forms part of a contiguous sequence of symbols in 'dictionary order'. This information is entered in the Alphabet dialogue (see Figure 9.2).

The leftmost column gives the symbol that will be displayed in the grid (the 'entry'), which will most often be an upper-case letter. The next column gives a list of all the characters that can appear in a dictionary file that you would like mapped to that grid entry. This list will usually comprise various accented forms of the grid entry letter.

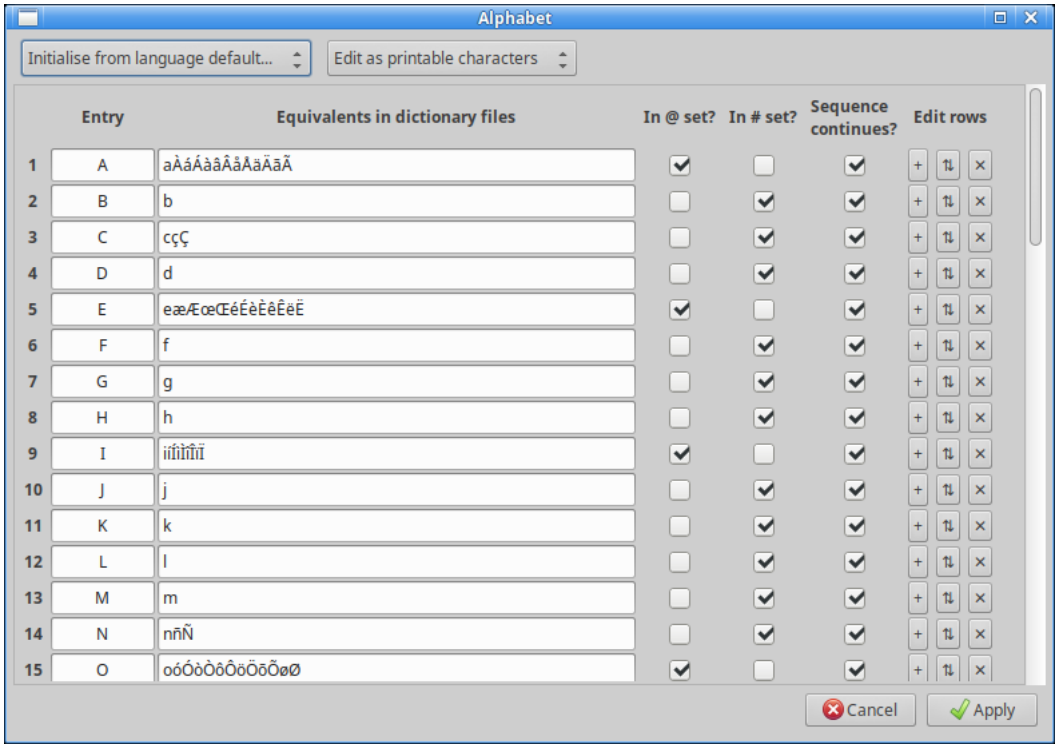

Figure: 9.2: The Alphabet dialogue

These first two columns can be edited as Unicode code points if you prefer: use the button at the top of the dialogue to switch between editing as printable characters (the default) and editing as Unicode code points. This can be helpful in situations where you are using symbols that look similar or identical to one another on the screen, such as Roman capital 'A' and Greek capital alpha.

The box in the third column is checked to indicate that the entry in question is a vowel, and the box in the fourth column is checked to indicate that the entry in question is a consonant. More precisely, you are specifying here whether the entry is a member of the set represented by the special symbols '@' and '#': although these are conventionally used to represent the sets of vowels and consonants respectively, there is no reason why they cannot be configured for a different purpose.

The box in the final column is checked to indicate that the natural successor to the entry in dictionary order is given in the next row of the table. This is so that Qxw can know that it can represent, for example, the set of letters 'FGHIJKL' compactly as 'F-L'. In the configuration 'Roman plus digits A-Z 0-9' the entry 'Z' has this box unchecked. That means that Qxw will write the set of symbols 'WXYZ0123' as 'W-Z0-3' and not as 'W-3'.

If your alphabet contains both letters and digits, you should normally list all the letters first, followed by the digits. Any symbols and other special characters you wish to use should appear at the end. The reason for this is explained in Section 9.9.

The groups of three buttons on the far right allow you to manipulate the alphabet table row by row. The first button inserts a new blank row in the table, pushing the subsequent rows

down by one position; information in the bottom row is lost. The second button exchanges the information in the current row with the information in the next row. And finally the third button deletes the current row, bringing all subsequent rows up by one position and introducing a new blank row at the bottom of the table.

9.7.1 Two-character expansions

If the first field in a row of the Alphabet dialogue contains two characters, then it is treated specially: when a dictionary is read, characters given in the second field are mapped to that pair of characters. Each character in that pair should occur elsewhere in the Alphabet dialogue as a separate possible grid entry. This allows the expansion of umlauted vowels in German to the undecorated vowel followed by the letter 'E' (e.g. Ä to AE) and scharfes S ('ß') to 'SS', and, in Ancient Greek, the conversion of iota subscript to iota adscript.

9.7.2 Further remarks on alphabets

The alphabet configuration is saved along with the grid. It is not possible to use 'Undo' to reverse changes that have been made to the alphabet configuration after you have clicked on 'Apply'. Internally, Qxw uses codes in the range 1–60 to represent characters: these 'internal character codes', or ICCs, are the numbers shown to the left of each row in the Alphabet dialogue. These codes are not adjusted when the alphabet is changed, and so changing the alphabet when a grid is half-completed may give unexpected results. When a change is made to the alphabet all dictionary files are re-read and any answer treatment plug-in is reloaded.

9.8 Diagnosing problems with dictionaries and alphabets

It is easy to make a mistake when setting up an alphabet and dictionary, resulting in unexpected 'words' in the feasible word list or in a grid fill. To help diagnose this kind of problem, you can use the Dictionary statistics dialogue, which you can call up by selecting the menu item *Autofill-Analyse dictionaries*: see Figure 9.3.

There is one tab in this dialogue for each dictionary, showing the following information. First is the name of the dictionary file, or, in the case of a single-entry dictionary, its entry. Next is the number of 'usable entries' it contains, which is the number of lines in the dictionary file that contain at least one valid character in the current alphabet. Following that is the number of entries remaining after the 'file filter' and then the 'answer filter' are applied. Finally there is a list of all the characters found in the file that are not representable in the current alphabet and which have therefore been ignored. Each is given as its Unicode code point, printed representation, and number of occurrences. For plain-text dictionary files, the line number of its first occurrence is also shown.

If you find that the entries in your dictionary files are not being displayed correctly in the feasible word list or in a grid fill, you should also check that your files use a character encoding that Qxw understands: see Section 9.5.

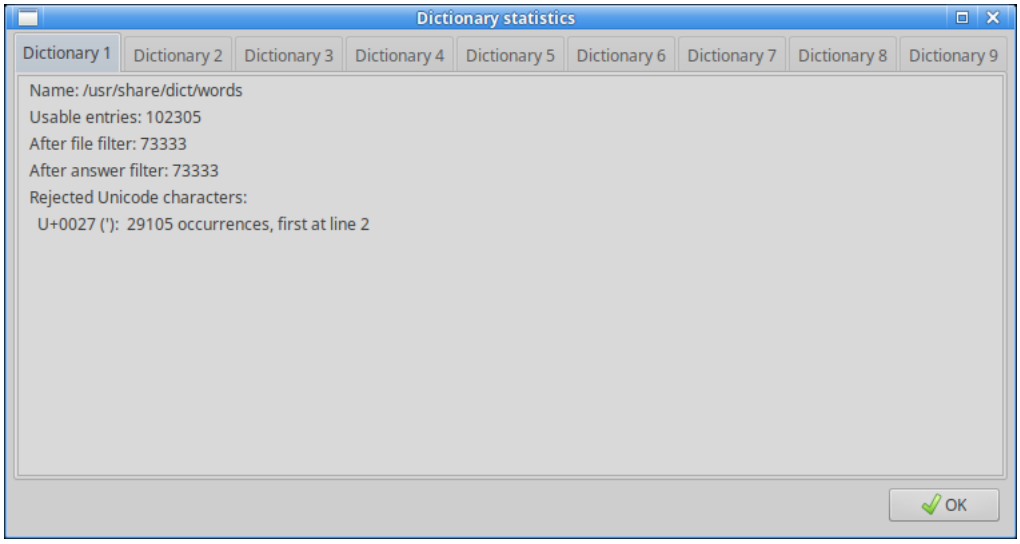

Figure: 9.3: The Dictionary statistics dialogue

9.9 Character classification

Qxw classifies each character of the current alphabet as a *letter*, a *digit*, or a *symbol*. It uses this classification, for example, when performing a Caesar cipher answer treatment to ensure that letters are always encoded as letters, digits as digits, and symbols as symbols.

Qxw bases its classification on the Unicode properties of each character in the current alphabet. Starting with the first character of the alphabet it collects all those whose Unicode properties indicate that they are alphabetic, and groups these into the 'letter' class. It then collects any subsequent characters whose Unicode properties indicate that they are numeric, and groups these into the 'digit' class. All subsequent characters are grouped into the 'symbol' class.

That means that if you follow the advice above and list letters followed by digits followed by symbols you will get the expected behaviour. But you can also arrange to have a character that would normally be thought of as a letter to be classified as a symbol. For example, if your alphabet is (in order) 'ABCX012*' then A, B, C and X will be considered letters, 0, 1 and 2 digits, and * a symbol. If, on the other hand, your alphabet is 'ABC012X*' then A, B and C will be considered letters, 0, 1 and 2 digits, and X and * symbols. Thus the sets 'letters', 'digits' and 'symbols' are represented by three consecutive blocks of internal character codes.

Chapter 10

Preferences and statistics

10.1 Preferences

The Preferences dialogue (Figure 10.1) is accessed via the menu item *Edit-Preferences*. It allows you to configure a number of details of the way Qxw behaves.

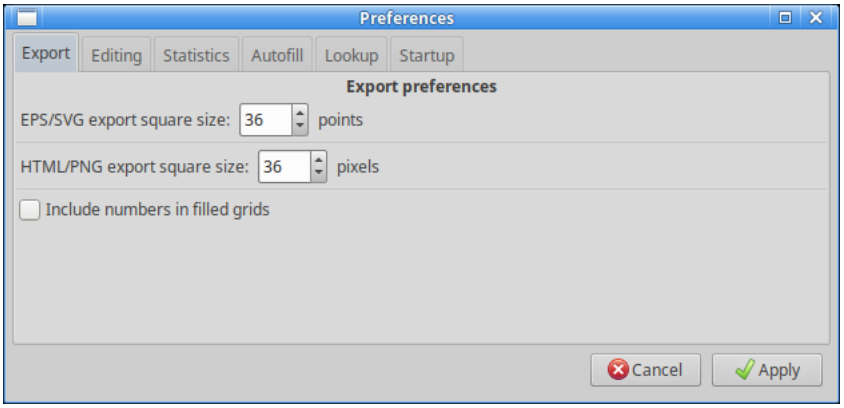

Figure: 10.1: The Preferences dialogue: 'Export' tab

Under the **Export** tab you can specify the size of cells in exported grids. For square grids, this is the length of the side of the square; for hex grids, the approximate distance between two parallel sides of a cell; and for circular grids, the height of the cell in the radial direction.

There is also an option to specify whether light numbers are included in exported solutions. Other marks are always included.

Normally, clicking the mouse in the grid moves the cursor, or, if you click on top of the cursor, changes the current direction. Under the **Editing** tab you can configure Qxw so that when you click near an edge of a cell a bar is added or removed, or so that when you click near a corner a block is added or removed. You can have both of these behaviours active at once if you like.

Also under **Editing** you can arrange for light numbers to be displayed in the grid as you edit, or you can suppress them to reduce clutter. Other marks always appear.

You can configure Qxw's idea of 'underchecked' and 'overchecked' under the **Statistics** tab.

The minimum checking ratio (before a light is deemed 'underchecked') is expressed as the percentage of checked cells in the light. The default value of 66% means that one unch is allowed in lights at least 3 cells long, two unches in lights of at least 6 cells, and so on. This is reasonable for typical barred grids, but you may wish to reduce the ratio to 50% for blocked grids.

The maximum checking ratio (before a light is deemed 'overchecked') is expressed as the percentage of checked cells in the light plus one cell. This permits fully-checked entries of up to a certain maximum length. The default value of 75% means that lights of up to 4 cells may be fully checked, lights of up to 8 cells must have one unch, lights of up to 12 cells must have two unches, and so on.

Under the **Autofill** preferences tab you can say whether the automatic fill function should always give the same result ('Deterministic') or potentially a different result every time it is invoked ('Slightly randomised' or 'Highly randomised'). The filler implements randomisation by not necessarily trying to put letters in cells in (what it regards as) the optimal order; as a consequence enabling randomisation may make the filler run more slowly.

By default the autofill function checks that no (pre-treatment) answer or light occurs twice in the grid. Untick 'Prevent duplicate answers and lights' to remove this check.

Under the **Lookup** tab you can configure what appears in the 'context menu' that pops up when you right-click on a word in the feasible word list. For each of the six available slots, you can specify its 'name' (how it is listed in the menu) and the corresponding URI (Uniform Resource Identifier) that Qxw will attempt to open. Within the URI you should include the sequence `%s`, which Qxw will replace with the word to be looked up. For security reasons, Qxw will only try to open a URI that starts `http://` or `https://`, but note that a maliciously-designed preferences file could allow an attacker to exploit a security weakness elsewhere in your system by opening an unwanted page in your browser.

Under the **Startup** tab you can configure the dictionary and alphabet that Qxw will use when it starts up. See Chapter 9 for more information. Note that if you are changing the default alphabet at start-up you will almost certainly want to configure a matching default dictionary too.

Under Linux, the preferences file is normally located at `.qxw/preferences` in the user's home directory. Under Windows, it is normally located at `C:\Documents and Settings\`*username*`\Application Data\Qxw\Qxw.ini`.

10.2 Statistics

Selecting the menu item *Edit-Show statistics* brings up the Statistics dialogue: see Figure 10.2.

At the top of the dialogue under the 'General' tab is a table analysing the lights in the grid by length. For each length it shows the number of lights of that length, the number of these (and percentage) that are under- or over-checked, the average checking ratio (proportion of checked letters in a light) and the minimum and maximum checking ratios.

Below the table some more general statistics appear, including the total light count, the mean light length, the number of letters checked across all lights, the number of checked grid cells, a count of lights with double unches and triple-and-above unches, and the number of free lights.

You can adjust Qxw's idea of what 'underchecked' and 'overchecked' mean in the Preferences dialogue: see Section 10.1.

The Statistics dialogue also reports when there are any lights in the grid that are too long for the

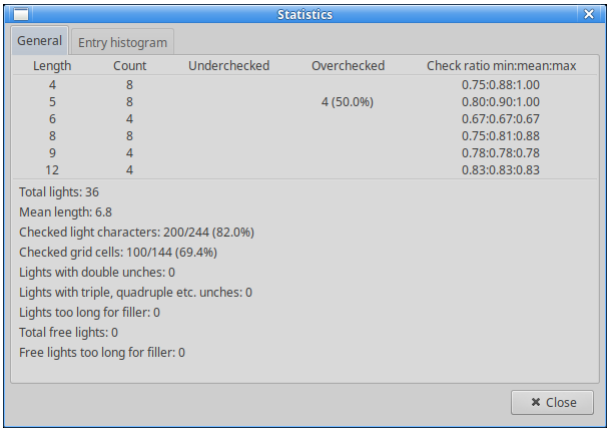

Figure: 10.2: The Statistics dialogue: 'General' tab

automatic filler to operate on. This can happen when using multiple characters per grid cell, with types of grid that allow lights to wrap around, or when using free lights.

The dialogue will also show a warning when there are too many lights with answer treatment enabled to allow the use of the discretionary fill modes.

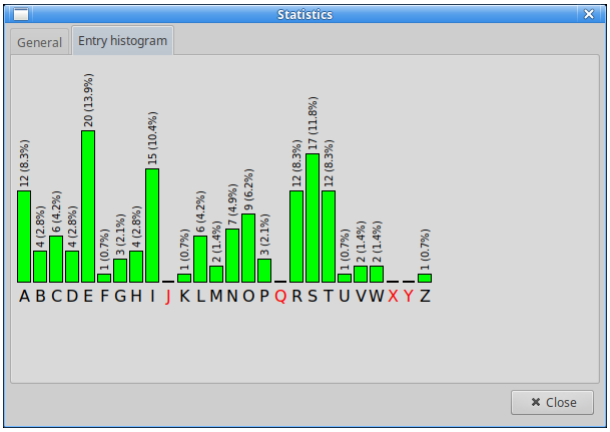

Figure: 10.3: The Statistics dialogue: 'Entry histogram' tab

Clicking on the 'Entry histogram' tab will show a histogram of the characters used in the grid. Only characters that have been entered into the grid are counted: grey 'hint' letters are not counted. You can use this, for example, to check whether a fill is pangrammatic.

Unlike Qxw's other dialogues, you can leave the Statistics dialogue active while you edit the grid. The information it displays is continuously updated as you work.

Chapter 11

Selecting cells and lights

Several of Qxw's functions operate on a selected set of cells or lights in the grid. For example, the menu item *Edit-Clear selected cells* ('shift-control-X') erases the letters that have been entered in the selected cells. Keyboard short-cuts for commands to do with selection involve the shift key.

Qxw can be in one of two selection modes: 'cell mode' and 'light mode'. The cell selection commands switch Qxw to cell mode; the light selection commands switch Qxw to light mode. Blocks and cutouts cannot be selected. Selected cells are shown with a solid highlight; selected lights are shown highlighted by a thick line.

See Section 13.2 for information on how to select free lights.

11.1 Selecting cells

Cells can be selected and deselected by holding down the shift key and the left mouse button while moving the mouse over the grid.

The command *Select-Current cell* ('shift-C') switches Qxw to cell selection mode (if it is not already in that mode) and adds the current cell to or removes the current cell from the cell selection.

When in cell selection mode, the command *Select-Nothing* ('shift-N') deselects all cells, leaving Qxw in cell selection mode; the command *Select-All* ('shift-A') selects all cells; and the command *Select-Invert* ('shift-I') selects all cells previously not selected, and deselects all cells previously selected.

The command *Select-Cells-overriding default properties* switches Qxw to cell selection mode (if it is not already in that mode) and selects those cells which have been set to override the default cell properties (see Section 12.1).

The command *Select-Cells-flagged for answer treatment* switches Qxw to cell selection mode (if it is not already in that mode) and selects those cells which have been flagged for answer treatment (see Section 12.1).

The command *Select-Cells-that are unchecked* switches Qxw to cell selection mode (if it is not already in that mode) and selects those cells which are not checked.

11.2 Selecting lights

Lights running in the direction in which the cursor is pointing can be selected and deselected by holding down the shift key and the right mouse button while moving the mouse over the grid. (This feature is not available under all versions of the Windows operating system.)

The command *Select-Current light* ('shift-L') switches Qxw to light selection mode (if it is not already in that mode) and adds the current light to or removes the current light from the light selection.

When in light selection mode, the command *Select-Nothing* ('shift-N') deselects all lights, leaving Qxw in light selection mode; the command *Select-All* ('shift-A') selects all lights; and the command *Select-Invert* ('shift-I') selects all lights previously not selected, and deselects all lights previously selected.

The command *Select-Lights-in current direction* switches Qxw to light selection mode (if it is not already in that mode) and adds to or removes from the selection those lights which run in the direction the cursor is currently pointing.

The command *Select-Lights-overriding default properties* switches Qxw to light selection mode (if it is not already in that mode) and selects those lights which have been set to override the default light properties (see Section 12.2).

The command *Select-Lights-with answer treatment enabled* switches Qxw to light selection mode (if it is not already in that mode) and selects those lights whose properties have been set to enable answer treatment (see Section 12.2).

Select-Lights-with double or more unches, *Select-Lights-with triple or more unches*, *Select-Lights-that are underchecked* and *Select-Lights-that are overchecked* are commands that allow you to locate those lights so flagged, as described in the discussion of the Statistics dialogue (see Section 10.2).

11.3 Switching selection mode

In general when Qxw switches between cell and light selection mode as a consequence of one of the above commands the selection is reset. However, the command *Select-Cell mode <> light mode* ('shift-M') switches Qxw between selection modes without clearing the selection: if Qxw is in cell selection mode, it switches to light mode, selecting all lights incident with any selected cell; and if it is in light selection mode, it switches to cell mode, selecting all cells that form part of any selected light.

Chapter 12

Cell and light properties and cell contents

Qxw lets you customise your grid and how it is filled by assigning properties to cells and lights.

In each case there is a set of default properties which can be overridden as required. For example, you would normally have the default cell properties set to give black text on a white background, but you could override this default to to give white text on a red background in cells whose letters spell out a theme word.

Usually you would first set the default properties (using the command *Properties-Default cell properties* or *Properties-Default light properties*); then select the cells or lights where you wish to override the defaults using the selection commands described in Chapter 11; and finally use the *Properties-Selected cell properties* or *Properties-Selected light properties*) commands to set the new properties.

12.1 Cell properties

Figure 12.1 shows the Selected cell properties dialogue. (The Default cell properties dialogue is the same except that it lacks the 'Override default cell properties' option.)

The first section of the dialogue allows you to change the colours used for foreground text, for corner marks, and for the background of the cell, as well as the font style (normal, bold, italic or bold italic) for characters entered in the cell.

Next, for each corner you can specify a (short) string to appear there: you can use this, for example, to distinguish some cells by adding an asterisk or a letter of the alphabet in the top right corner. The characters allowed in corner marks are letters, digits, and normal punctuation.

If the mark is the special sequence consisting of a backslash followed by a hash character, that corner will be used to display the light number (if any).

If the mark is the special sequence consisting of a backslash followed by the letter 'c' or 'C', then (assuming that the cell is normally checked and that there is only a single character in the cell) that corner will be used to display a number that depends on the character in the cell. The mapping from character to number is randomly chosen afresh each time Qxw is run. This facility makes it easy to produce 'codeword'-style puzzles.

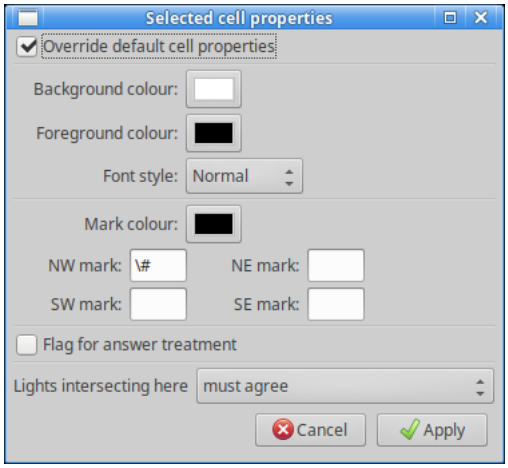

Figure: 12.1: The Selected cell properties dialogue

If the mark is the special sequence consisting of a backslash followed by the letter 'i' or 'I', and the first (or only) free light passes through that cell, then that corner will be used to display a number giving the position of the cell within the free light. This makes it possible to highlight a sequence of letters forming a thematic word in the grid, automatically updating if you change the path of the free light.

If the mark is the special sequence consisting of a backslash followed by the letter 'o' or 'O', then a circle will be drawn around the character or characters entered in that cell. This gives another way to highlight a thematic word hidden in the grid. This mark can be specified as belonging to any corner of the cell, with no difference to the visual effect; and the circle is drawn in a half-intensity version of the specified mark colour. Together, these mean that circles can be combined with other marks with a large degree of freedom. Circles do not appear in versions of the grid exported to pure HTML.

The second section of the dialogue allows the cell to be 'flagged for answer treatment': this means that the cell can be dealt with specially when a light passing through it is subject to answer treatment. For more details see Chapter 14. You can also specify whether lights intersecting in the cell must agree (the normal case) or whether the cell is 'dechecked', in which case they need not agree. If the cell is 'dechecked' the contributions from the lights passing through it are shown separately, either side by side ('horizontal display') or atop one another ('vertical display').

When the dialogue is called up, it shows the properties of the first selected cell in the grid in normal reading order. When you click on the 'Apply' button, the chosen properties are applied to all selected cells.

12.2 Light properties

Figure 12.2 shows the Selected light properties dialogue. (As before, the Default light properties dialogue is the same except that it lacks the 'Override default light properties' option.) When the dialogue is called up, it shows the properties of the first selected light in the grid (in clue order). When you click on 'Apply', the chosen properties are applied to all selected lights.

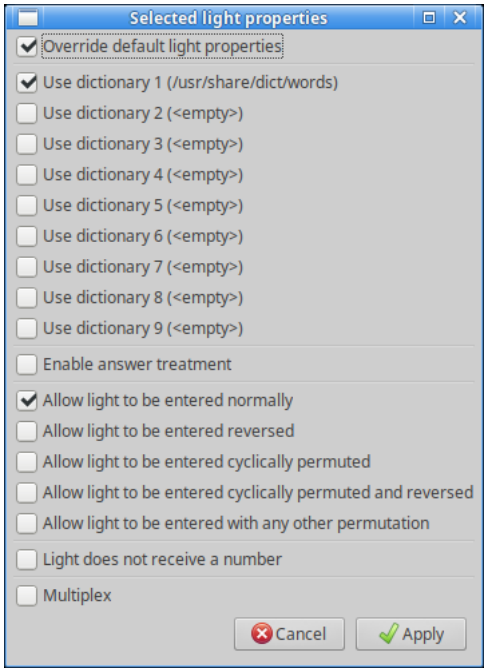

Figure: 12.2: The Selected light properties dialogue

The first section of the dialogue allows you to choose which dictionaries are to be used for filling the light: see Chapter 9. If no dictionaries are selected Qxw will allow any combination of characters in the light.

The second section lets you enable or disable answer treatment for the light: see Chapter 14.

The third section allows you to choose the 'entry method' for the light. Answers can be entered forwards and/or backwards and may also be cyclically permuted. A further option allows for any other permutation of the letters in the light. Allowing arbitrary permutations or cyclic permutations in conjunction with a large dictionary can make filling run slowly and use more of the computer's memory.

If 'with any other permutation' is ticked then an asterisk is appended to all the results shown in the feasible list pane (whether or not the light is in fact jumbled), and clicking on these results does not have any effect.

The fourth section of the Selected light properties dialogue allows a light to be excluded from the usual consecutive numbering, which can be desirable in crosswords with unclued thematic entries. The first cell of a light will still receive a number if a number is required for another light starting in the same cell.

The fifth section of the Selected light properties dialogue allows you to configure the light as a 'multiplex light'. This means that the light can be filled in more than one way, treating cell contents strings that contain more than one letter as alternative single-letter fills rather than as a single multi-letter fill. The number of alternatives is determined by the longest cell contents string over the light.

If you make changes to the grid after having set some light properties, Qxw will try to make an intelligent decision about which lights should have which properties. Although it does its best,

it is possible that Qxw's view on this will disagree with yours. The general rule is that Qxw attaches light properties to the cell containing the first letter of a light. Check (with the help of *Select-Lights-overriding default properties*) that things are as you expect.

12.3 Cell contents

Normally the contents of a cell can be changed by simply moving the cursor to the cell and either pressing the desired letter on the keyboard or pressing 'Tab' to remove a letter. When a cell contains more than one character, however, the cell contents are changed using a dialogue: see Figure 12.3.

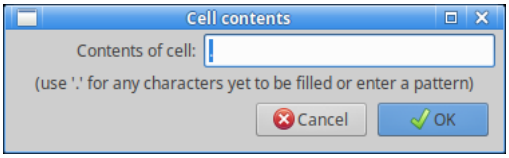

Figure: 12.3: The Cell contents dialogue

This dialogue automatically appears when (by pressing a letter or digit key or 'Tab') you attempt to change the contents of a cell that does not contain exactly one character; it is also available via the menu item *Edit-Cell contents* ('control-I'). If the cell is normally checked you are prompted to enter the characters that are to occupy the cell; if it is dechecked (see under 'Cell properties' above) you are prompted for the (independent) contributions from the cell to lights in each direction.

In each case a full stop ('.') must be used to indicate characters that are not yet decided upon (including in multiplex lights). For clarity the grid display will show these full stops in all cases except where a cell is normally checked and only contains a single character.

Where you have several instances of a cell containing more than one character, it is sometimes easier to use the Light contents dialogue (*Edit-Light contents*, 'control-L') than the Cell contents dialogue.

Furthermore, it is possible to partially restrict the characters the automatic filler is allowed to use in a cell. This is done by entering a pattern rather than a single character or a full stop. For example, entering the pattern `[aeiou]` would force the cell to contain a vowel; the pattern `[^s]` would allow any character except 's'; the pattern `[a-m][a-rt-z]` would allow a sequence of two characters, the first any letter from 'a' to 'm' and the second any letter except 's'; and `@#.` would allow a sequence of three characters, the first being a vowel, the second a consonant, and the third unrestricted.

One use of this facility is to prevent the filler using words ending in 's' at the right-hand or bottom edges of the grid.

Chapter 13

Free lights

'Free light' is the term Qxw uses to refer to a light consisting of an arbitrary sequence of cells in the grid that is constrained to form a word (or treated word) just like the normal lights. For example, this allows you to create lights that run along the diagonals or around the periphery of a square grid or take knight's tours around the grid. As the examples in the first part of this guide show, free lights can also be used in conjunction with specially-constructed dictionaries to make more complicated thematic puzzles.

13.1 Making free lights

To create a new free light move the cursor to the first cell that is to be part of it and select the menu item *Edit-Free light-Start new*. A small orange dot will appear. This is the first cell of the new free light, and it is automatically selected. Move the cursor to the desired second cell and select the menu item *Edit-Free light-Extend selected* ('control-E'). An orange dot will appear in this second cell, joined by a thin orange line to the first. Continue using the menu item *Edit-Free light-Extend selected* ('control-E') to add all the desired cells one by one. The free light appears as a series of small dots linked by thin lines. The first dot in the free light is square, the remainder circular: this is so that you can see in which direction the light runs.

If you make a mistake you can remove the last cell from the free light using the menu item *Edit-Free light-Shorten selected* ('control-D'). or delete the light altogether using *Edit-Free light-Delete selected*.

13.2 Selecting and editing free lights

As mentioned above, the free light under construction is automatically selected (i.e., displayed in orange). For clarity, free lights are not usually displayed in the grid if not selected. However, you can display them by moving the cursor to a cell through which a free light passes and changing the cursor direction (by pressing 'Page Up', 'Page Down' or 'slash', or by clicking with the left mouse button). The cursor will cycle through the usual directions and then through the free lights that visit that cell, displaying them in grey. The feasible word list will update to show you what words can be used in the free light.

When a free light is displayed in grey the cells that form it can be edited in one step using the Light contents dialogue (*Edit-Light contents*, 'control-L').

When a free light is displayed in grey it can be selected using the menu item *Select-Current light* ('shift-L'). Alternatively, it is possible to cycle through the free lights, selecting them in turn, using the menu item *Select-Free light* ('shift-F'). Using the menu item *Select-All* ('shift-A') when any free light is selected will select all the free lights.

When a single free light is selected its path can be modified directly using the menu item *Edit-Free light-Modify selected*. This calls up a dialogue in which the coordinates of the cells visited by the free light are displayed and can be edited. It is also possible to create a coordinate list externally to Qxw and paste it into this dialogue. Each line in the list identifies a single cell in the grid by its horizontal and vertical coordinates (or angular and radial coordinates in the case of circular grids), counting from zero. The elements of the coordinate pair must be separated by a space or comma.

The paths of all the free lights can be written to a file using the menu item *File-Export free light paths*. The format is: one coordinate pair per line, separated by a space, with a blank line between each sequence of coordinates representing a single free light. A command *File-Import free light paths* is available to read in files in this format.

Properties can be set on free lights, just like normal lights, by selecting them and then using the Light properties dialogue: see Section 12.2.

When a free light visits a cell containing more than one character, all characters in the cell contribute to the free light, even if the cell is 'dechecked'. This means that a free light running through such a cell will result in a constraint being applied to that cell's contents.

Chapter 14

Answer treatments

14.1 Built-in answer treatments

Qxw contains a variety of built-in answer treatments. This following sections describe in detail how each behaves. In each case, the treatment is thought of as transforming a word (which will have been obtained from a dictionary file) called the 'answer' into the 'light' for entry in the grid. The answer treatments make use of up to two 'messages', which in a thematic crossword would typically be hidden quotations or further instructions to the solver.

14.1.1 Playfair cipher

This answer treatment uses only characters from the first message, which is used to make the keyword for a Playfair square. All 'J's in the keyword are replaced by 'I's (if 'I' is present in the current alphabet). All but the first occurrence of each character is deleted. The characters of the message that remain, followed by the remaining characters of the alphabet (except 'J', if 'I' is present in the alphabet) in order, are written to the Playfair square in normal reading order until it is full or the alphabet is exhausted. If the alphabet, possibly after reduction by removal of 'J', consists of more than 25 characters, the excess characters are discarded.

All 'J's in the answer are replaced by 'I's if 'I' is present in the current alphabet. The answer is then split into pairs of characters which are encoded using the Playfair square in the usual way. If (a) a pair consists of the same character twice; or (b) either character of the pair is one of the excess characters discarded in the construction of the square; or (c) encoding the pair would involve accessing a cell of the Playfair square that is not filled because there are not enough characters in the alphabet, then the pair is left unencoded.

If an answer has odd length then its last character is left unencoded.

14.1.2 Substitution cipher

This answer treatment uses only characters from the first message. Each character in the answer is considered in turn. If it is classed as a 'symbol' (see Section 9.9) then it is left unchanged. Otherwise, its position within its character class is determined, and this value is used to index into the message to find a replacement character. If the message is too short to allow this, the

character is left unchanged.

For example, with the 'Roman plus digits A-Z 0-9' alphabet, each 'A' or '0' in the answer is replaced by the first character of the message; each 'B' or '1' by the second character; each 'C' or '2' by the third character; and so on.

14.1.3 Fixed Caesar/Vigenère cipher

This answer treatment uses the first message as a keyword for a Vigenère cipher. Letters are always encoded as letters and digits as digits. Symbols are left unchanged. If the message is empty, the answer is left unchanged.

The positions (counting from zero) of corresponding characters of answer and keyword within their respective character classes are determined, and then added together. The result is converted back to a character by indexing into its character class, wrapping around if necessary.

For example, with the 'Roman plus digits A-Z 0-9' alphabet, we have A $\equiv$ 0, B $\equiv$ 1, C $\equiv$ 2 and so on. Then A + A = A, C + F = H, N + N = A, Y + Z = X, Q + 3 = T and 3 + Q = 9.

If the keyword is shorter than the answer it is repeated as necessary. In particular, if the keyword consists of one character, the result is a fixed Caesar cipher. Using the keyword 'N' with a Roman alphabet results in the `rot13` cipher.

14.1.4 Variable Caesar cipher

This answer treatment assigns characters from the first message to lights. One character is assigned to each light that has answer treatment enabled, in clue order. The character is used to derive an offset for a Caesar cipher, the same offset being used throughout a single light. The encoded character is calculated in the same way as for the fixed Caesar cipher above.

If the message is empty, answers are left unchanged; if the length of the message is less than the total number of lights with answer treatment enabled, the message is repeated as necessary.

For example, with the 'Roman plus digits A-Z 0-9' alphabet, a message character of 'A' or '0' leaves an answer unchanged; a message character of 'B' or '1' advances the characters of an answer by one place (taking e.g. 'LAZY' to 'MBAZ' and '1966' to '2077'); a message character of 'C' or '2' advances the characters of an answer by two places (taking 'LAZY' to 'NCBA' and '1966' to '3188'); and so on.

This answer treatment can take advantage of the discretionary modes of the filler.

14.1.5 Misprint (correct letters specified)

This answer treatment uses only characters from the first message. Each light with answer treatment enabled uses one character from the message. Lights are produced from answers by changing an occurrence in the answer of the character from the message to a different character. This means that a single answer word can give rise to two or more different lights, or, equally, may give rise to no lights at all.

If the message length is less than the total number of lights with answer treatment enabled, excess answers are left unmodified.

This answer treatment can take advantage of the discretionary modes of the filler.

14.1.6 Misprint (incorrect letters specified)

This answer treatment uses only characters from the first message. Each light with answer treatment enabled uses one character from the message. Lights are produced from answers by changing a character in the answer to the (different) character from the message. This means that a single answer word can give rise to two or more different lights, or, equally, may give rise to no lights at all.

If the message length is less than the total number of lights with answer treatment enabled, excess answers are left unmodified.

This answer treatment can take advantage of the discretionary modes of the filler.

14.1.7 Misprint (general, clue order)

This answer treatment uses both messages in their raw form, before punctuation is removed. Each light with answer treatment enabled, in clue order, uses one character from each message. Lights are produced from answers by changing an occurrence of the character from the first message to the character from the second message. This means that a single answer word can give rise to two or more different lights, or, equally, may give rise to no lights at all.

If the character from the first message is '.' then any character in the answer can be changed to the character from the second message to make the light. If the character from the second message is '.' then an occurrence in the answer of the character from the first message can be changed to any character to make the light. If the characters from both messages are '.' the answer is left unchanged.

Other than when specifically instructed (i.e., when the characters from the two messages are the same), Qxw will not 'change' a character to itself.

A message whose length is less than the total number of lights with answer treatment enabled is considered to be extended with '.' characters.

14.1.8 Delete single occurrence of character (clue order)

This answer treatment uses only characters from the first message. Each light with answer treatment enabled, in clue order, uses one character from the message. For each occurrence of that character in a candidate answer word, a light is constructed by deleting that occurrence. A single answer word can thus give rise to two or more different lights.

If the length of the message is less than the number of lights with answer treatment enabled, lights without a specified character to be deleted are left unchanged.

This answer treatment can take advantage of the discretionary modes of the filler.

14.1.9 Letters latent: delete all occurrences of character (clue order)

This answer treatment uses only characters from the first message. Each light with answer treatment enabled, in clue order, uses one character from the message. If a candidate answer word does not contain that character, it is discarded; otherwise a light is constructed by deleting every occurrence of the character from the answer.

If the length of the message is less than the number of lights with answer treatment enabled, lights without a specified character to be deleted are left unchanged.

This answer treatment can take advantage of the discretionary modes of the filler.

14.1.10 Insert single character (clue order)

This answer treatment uses only characters from the first message. Each light with answer treatment enabled, in clue order, uses one character from the message. A series of lights is constructed from each candidate answer word by inserting that character at each possible position within the word, including at either end.

If the length of the message is less than the number of lights with answer treatment enabled, lights without a specified character to be inserted are left unchanged.

This answer treatment can take advantage of the discretionary modes of the filler.

14.2 Filler discretionary modes

Qxw offers three alternative methods by which characters in answer treatment messages can be allocated to the lights that have answer treatment enabled.

The normal case is called 'in clue order, first come first served': the letters are allocated to the lights that have answer treatment enabled in clue order, and this allocation is fixed. If there are more such lights than characters in the message the behaviour depends on the particular treatment in question.

The second alternative is called 'in clue order, at discretion of filler'. If there are as many characters in the message as lights to be treated, this behaves in the same way as 'first come first served' mode. However, if there are fewer, then the filler will distribute them as it sees fit across the lights, preserving their order, and leaving as many lights as required untreated.

The third alternative is called 'in any order, at discretion of filler'. In this mode the filler will distribute the characters in the message as it sees fit among the lights that have answer treatment enabled without necessarily preserving their order. If there are fewer characters in the message than lights to be treated then the filler will leave as many lights as required untreated.

The allocation of characters to lights is subject to the constraint string entered in the Answer treatment dialogue. The string specifies what message characters are acceptable for each light in clue order. The message character can be any character from the current alphabet or the special character dash ('-'), which indicates that the light is not to be treated. The symbol '.' (full stop) stands for any character except dash, and the symbol '?' (question mark) stands for any character including dash. A set of allowable message characters can be specified using a syntax along the lines of '[A-DQ-Z-]', and Qxw will use this syntax if *Autofill-Accept hints* ('control-A') is used on a partially-filled grid: the interactive assistance algorithm will often be able to prove quickly that some allocations are infeasible.

The constraint string is considered to be padded with question mark characters at the end if necessary.

Note that the allocation of characters to lights is an integral part of the filler's algorithm: it does not simply choose an arbitrary allocation and fix it before embarking on the fill. Allowing the filler discretion in allocating message characters can turn an infeasible fill into a feasible one; on the other hand, giving the filler more freedom can also make it run more slowly.

14.3 Plug-in answer treatments

One of Qxw's most powerful features is its ability to use customised answer treatments in the form of 'plug-ins', which are small external programs that Qxw loads on request.

This section describes the plug-in system in detail. Initially we will consider only the case where the current alphabet comprises the letters 'A' to 'Z' and/or the digits '0' to '9'; the extensions to the plug-in system to cover other alphabets are discussed in Section 14.3.1.

If 'Custom plug-in' has been chosen as the answer treatment method, Qxw will call the plug-in when building the feasible word list for each light with answer treatment enabled. This happens every time the user makes a change to the grid. Qxw calls the plug-in once for each word in the set of dictionaries selected for the light in question, passing in the word to be treated along with other information that the plug-in might need.

As you can see, a plug-in might get called hundreds of thousands or even millions of times whenever the user makes a change to the grid. Although Qxw will interrupt its feasible light list building to respond to a user action, it is nevertheless important that plug-in code execute as quickly as possible.

The plug-in takes the form of a C program. Under Linux, this program can be compiled using `gcc` with its `-shared` and `-fPIC` options. For Windows compilation options see Section 14.4. The resulting object file (which conventionally has a '`.so`' suffix under Linux and a '`.dll`' suffix under Windows) is dynamically loaded and unloaded by Qxw as necessary.

The program must provide a function called `treat()`. The function is passed a candidate answer word (as a `char*`), entirely in capitals and with no punctuation or spaces. It is expected to make a local modified version of this string (the 'treated answer'), again entirely in capitals and with no punctuation or spaces. The function must then call `treatedanswer()` with the modified string, which will be considered as a candidate for addition to the feasible light list. The function `treatedanswer()` returns a non-zero value if an error occurs, and this value should be passed back as the return value of `treat()`.

In many cases your `treat()` function will need to call `treatedanswer()` exactly once, in which case the function can simply end `return treatedanswer(...);`. If the treatment is such that an answer can give rise to several different lights, `treat()` will need to call `treatedanswer()` more than once, and the returned value must be checked each time and returned if non-zero.

In addition, the plug-in may provide a function called `init()`, which is called immediately after the plug-in is loaded, and a function called `finit()`, which is called immediately before the plug-in is unloaded. The plug-in is reloaded whenever the user clicks 'Apply' in the Answer treatment dialogue or in the Alphabet dialogue, and so the `init()` function is a suitable place to do any relatively time-consuming pre-processing (such as building encoding tables) that the plug-in requires.

By including the header file `qxwplugin.h` a plug-in can access the following variables.

`int clueorderindex`, the sequence number of the current light in clue order, counting from zero. Only lights with answer treatment enabled are counted.

`int lightlength`, the length of the current light in characters.

`int lightx`, the x-coordinate of the start position of the current light (or angular position for circular grids), counting from zero.

`int lighty`, the y-coordinate of the start position of the current light (or radial position for circular grids, measured inwards from the perimeter), counting from zero.

`int lightdir`, the direction of the current light. For plain rectangular grids, the directions (in

order counting from zero) are 'Across' and 'Down'; for hex grids with vertical lights 'Northeast', 'Southeast' and 'South'; for hex grids with horizontal lights 'East', 'Southeast' and 'Southwest'; and for circular grids 'Ring' and 'Radial'. Free lights are assigned `lightdir` values from 100 upwards.

`int*checking`, a pointer to an array of `lightlength ints`, one for each character in the light. Each entry is the degree of checking experienced by that character. An 'unchecked' character has a checking value of 1, and a normally checked character has a checking value of 2. Higher checking values can arise when using merged cells, hexagonal grids or free lights.

`int*gridorderindex`, a pointer to an array of `lightlength ints`, one for each character in the light. If the cell containing that character is not flagged for answer treatment in the Cell properties dialogue then the value is set to -1. Otherwise the value is set to the index, in normal reading order and counting from zero, of that cell among all flagged cells in the grid. For example, suppose a vertical light is five cells long and has its second and last cells flagged, and that these two cells are respectively the fourth and ninth flagged cells in the grid in reading order. The array will then read $-1, 4, -1, -1, 9$.

`char*treatmessage[]`, an array of two strings containing the messages as specified by the user in the Answer treatment dialogue.

`char*treatmessageAZ[]`, an array of two strings containing the messages specified by the user in the Answer treatment dialogue with only letters preserved, converted to capitals 'A' to 'Z'.

`char*treatmessageAZ09[]`, an array of two strings containing the messages specified by the user in the Answer treatment dialogue with only letters and digits preserved and all letters converted to capitals 'A' to 'Z'.

`char msgchar[]`, an array of two characters, one from each of the two messages specified by the user in the Answer treatment dialogue.

`char msgcharAZ[]` is analogous to `msgchar[]`, but based on `treatmessageAZ[]` rather than `treatmessage[]`. Only letters and the character '-' can appear in `msgcharAZ[]`.

`char msgcharAZ09[]` is analogous to `msgchar[]`, but based on `treatmessageAZ09[]` rather than `treatmessage[]`. Only letters, digits and the character '-' can appear in `msgcharAZ09[]`. If the allocation order for a message is 'in clue order, first come first served' then `msgcharAZ09[i]` is the same as `treatmessageAZ09[i][clueorderindex]`, or the character '-' if there are not enough characters in `treatmessageAZ09[i]`. If one of the other allocation orders is used, the situation is more complicated: see below.

`char light[]`, a temporary storage area with enough space for the longest possible light (`MXLE` characters) plus a zero termination byte. Plug-ins can use this area to construct the treated answer before passing it back to Qxw.

Plug-ins also have access to a function `int isword(const char*light)` which checks whether a given string (entirely capitals and digits, with no punctuation or spaces) is a word in any of the dictionaries selected for that light.

Caution: if a plug-in crashes, for example as a result of accessing storage before the beginning or after the end of the string passed to it, Qxw will also crash. Save your work before experimenting with plug-ins!

14.3.1 Writing a plug-in for a non-Roman alphabet

If your plug-in only has to deal with (at most) the letters 'A' to 'Z' and/or the digits '0' to '9' then you can write it in terms of the variables and functions listed above. If, however, it has to

deal with accented or other characters (more specifically, any character not in the '7-bit ASCII' set) then it will need to make use of the additional variables and functions described below; and unfortunately your program may end up a little more complicated.

The extra functions deal in two different representations of characters, and you can choose whichever is the more convenient for your purposes. In the first, each character is represented by an ordinary unsigned integer which is its Unicode code point. The functions and variables that use this representation have the letter 'U' in their names. For convenience, the header file `qxwplugin.h` includes a `typedef` that lets you write the type as `uchar` instead of `unsigned int`.

The other representation uses 'internal character codes', or ICCs, which are stored using the `char` type, and can take on values from 1 to 60. These are the numbers shown to the left of each row in the Alphabet dialogue. Observe that the value zero is not used; moreover, when Qxw manipulates strings of internal character codes it appends a zero termination byte. The consequence of this is that you can use the standard C string manipulation functions on them, and this can considerably simplify the writing of a plug-in. The functions and variables that use this representation have the string 'ICC' in their names.

In most cases you will find it easiest to write a plug-in in terms of internal character codes. This is also usually the most computationally efficient approach, and all the built-in treatments are written in this way.

For backwards compatibility with previous versions of Qxw, the argument to the `treat()` function is represented as UTF-8, as are the strings in the array `treatmessage[]`. In this encoding strings comprising exclusively 7-bit ASCII characters are unchanged; but other characters are represented in a slightly more complex way. To save you from having to deal with UTF-8 conversions, Qxw also makes the untreated answer available in the variables `const char*answerICC` (an array of bytes using internal character codes) and `const uchar*answerU` (an array of unsigned integers using Unicode code points, also referred to as `uchars`), in both cases zero-terminated.

You can only use `int treatedanswer(const char*light)` to submit the result of your answer treatment if it contains only characters that are 7-bit ASCII, or you can pass your non-7-bit-ASCII result encoded in UTF-8. More simply, you can call either `int treatedanswerICC(const char*light)`, which requires a zero-terminated string of internal character codes, or, if you have a zero-terminated string of Unicode code points, `int treatedanswerU(const uchar*lightU)`.

14.3.2 Treatment messages when using a non-Roman alphabet

The array `char*treatmessageICC[]` contains the same information as `char*treatmessage[]`, but using internal character codes and with any characters not representable in the current alphabet deleted.

The array `uchar*treatmessageU[]` contains the same information as `char*treatmessageICC[]` but in the form of `uchars` and with characters are mapped to their 'Entry' representative specified in the Answer treatment dialogue.

Likewise, the array `char*treatmessageICCAZ[]` corresponds to `char*treatmessageICC[]`, but with everything that is not a letter (as defined by its character classification: see Section 9.9) deleted.

The array `uchar*treatmessageUAZ[]` corresponds to `char*treatmessageICCAZ[]` with characters mapped to their 'Entry' representative specified in the Answer treatment dialogue.

The array `char*treatmessageAZ[]` corresponds to `uchar*treatmessageUAZ[]`, but with any

characters not representable using 7-bit ASCII deleted.

The array `char*treatmessageICCAZ09[]` corresponds to `char*treatmessageICC[]`, but with everything that is not a letter or digit (as defined by its character classification) deleted.

The array `uchar*treatmessageUAZ09[]` corresponds to `char*treatmessageICCAZ09[]`, with characters are mapped to their 'Entry' representative specified in the Answer treatment dialogue.

The array `char*treatmessageAZ09[]` corresponds to `uchar*treatmessageUAZ09[]`, but with any characters not representable using 7-bit ASCII deleted.

The arrays `char msgchar[]`, `char msgcharAZ[]` and `char msgcharAZ09[]` are generally not useful when writing plug-ins that deal with non-Roman alphabets. Instead, use the array `char msgcharICC[]`, which contains a character extracted from `char*treatmessageICC[]`; or `uchar msgcharU[]`, which contains a character extracted from `uchar*treatmessageU[]`. Similarly, instead of `char msgcharAZ[]` use `char msgcharICCAZ[]` or `uchar msgcharUAZ[]`; and instead of `char msgcharAZ09[]` use `char msgcharICCAZ09[]` or `uchar msgcharUAZ09[]`.

`uchar lightU[]` is a temporary storage area with enough space for the longest possible light (`MXLE` characters) represented using `uchars`, plus zero termination. Plug-ins can use this area to construct the treated answer before passing it back using `int treatedanswerU(lightU)`.

The function `isword(const char*light)` takes an argument encoded in UTF-8 (and hence works for strings that contain exclusively 7-bit ASCII). And, as you might expect, the functions `int iswordU(const uchar*lightU)` and `int iswordICC(const char*light)` are also available.

The function `int ICCtoclass(char c)` takes an ICC as argument and returns 0 if Qxw classifies that character as alphabetic, 1 if Qxw classifies that character as numeric, 2 if Qxw classifies that character as a symbol, and -1 otherwise.

The function `char uchartoICC(int c)` converts a `uchar` to its corresponding internal character code, returning zero if there is no representative in the current alphabet. Conversely, `uchar ICCtouchar(char c)` converts an internal character code to the `uchar` value for its 'Entry' representative specified in the Answer treatment dialogue, or zero if no representative is specified. `ICCtouchar(ICC_DASH)` returns the Unicode for '-'.

The function `void printICC(char c)` prints out a single character represented as an ICC. The function `void printICCs(const char*s)` prints out a string of characters represented as ICCs. The functions `void printU(uchar c)` and `void printUs(const uchar*s)` do the analogous thing for `uchars`. These functions are only intended to be used when debugging a plug-in.

14.3.3 Writing a plug-in to work with discretionary fill modes

If your plug-in is to be used in conjunction with discretionary fill modes, you must write it to depend on the variable `char msgcharAZ09[]`, or, if you are using a non-Roman alphabet, on `char msgcharICC[]` or `uchar msgcharU[]`. The treatment code must not depend on the `treatmessage[]` strings, and the other variables are not guaranteed to contain useful information.

For each light with answer treatment enabled the plug-in will be called once with each feasible combination of characters from the current alphabet and '-' characters in `msgcharAZ09[]`, `char msgcharICC[]` and `uchar msgcharU[]`.

If the length of the treatment message is less than the number of lights with answer treatment enabled then it is (in effect) padded to the correct length using '-' characters. Your plug-in

should therefore normally interpret this character to mean 'this answer is not to be treated', although this is not compulsory. The internal character code for '-' is defined in `qxwplugin.h` as `ICC_DASH`.

Speed of execution of the plug-in is particularly critical if there are many feasible combinations of message characters; also, long lists of feasible words can result, consuming large amounts of the computer's memory.

If both messages are used in an answer treatment, their character allocation modes can be set independently. Furthermore, the filler will in general choose different allocations for the two messages.

14.4 Compiling plug-ins under Windows

This guide contains a number of example plug-ins, and shows how they can be compiled under Linux using the `gcc` C compiler. Unfortunately, Windows does not come with a ready-made C compiler. There are several cost-free options, but none is entirely straightforward to use.

One option is to download and install Cygwin or MinGW, both of which provide a command line `gcc` C compiler that can be used in almost the same way as the Linux version. The key difference is that the shared object file under Windows will be a dynamic-linked library and will have a `.dll` file extension.

Alternatively, Microsoft provide a comprehensive range of programming and development tools called Visual Studio, the Community version of which is free to download and use. The rest of this chapter runs through the steps needed to create a Qxw plug-in using Visual Studio 2019. If you have Visual Studio 2017 already installed, then the steps are similar.

14.4.1 Using Microsoft Visual Studio

The first stage is to download and install Visual Studio 2019 Community from the Microsoft Visual Studio website. The first time you launch Visual Studio 2019 Community, you will need to sign in with a Microsoft account: you can create a new one if needed.

Launch Visual Studio and select *Create a new project*. Search for and select the 'Empty project' template. Give your project a suitable name and click the 'Create' button. This will create the project in your `C:\Users\`*username*`\source\repos` folder.

The next stage is to configure your project, and to make sure it can find and link to the Qxw code library. Select *Project-Properties* from the menu, and select 'All Configurations' from the drop-down menu. You will need to make a number of changes to the configuration. First, under *General-Configuration Type* select 'Dynamic Library (.dll)'. Next select *VC++ Directories-Edit Include Directories* and add in the Qxw application folder (normally something like `C:\Program Files (x86)\Qxw`); then *Edit Library Directories* and again add the Qxw application folder. Finally, select *Linker-Input*, edit 'Additional Dependencies' and enter `Qxw.lib`.

Now you are ready to write your code. Right-click on *Source Files* in Solution Explorer, and *Add New Item*. In the window that opens, select *C++ File* but make sure you give it a name that has a `.c` file extension. This will ensure that the project is compiled as C rather than C++ code.

You can now type in your code. A good starting point is simply to copy and paste one of the examples from this guide.

The final step is to compile and build your project. Click on *Build* and *Build Solution* from the

menu and, if all is well, your project will compile and produce a `.dll` file. This will sit in either the project `Debug` or `Release` folder, depending on which mode was selected on the menu bar (usually `Debug` to start with). Either mode will produce a `.dll` file that will run with Qxw; the 'Release' version will run faster.

You should unload the plug-in from Qxw before recompiling, for example by temporarily setting the answer treatment to something other than 'Custom Plug-in'.

14.4.2 Debugging a plug-in under Microsoft Visual Studio

You can debug a plug-in using the following steps:

1. Start Qxw and load or create your crossword.

2. In Visual Studio use the 'Debug' configuration, select *Debug-Attach To Process...* from the menu and attach to `Qxw.exe`.

3. Set a breakpoint at the beginning of the `treat()` function in your plug-in.

4. Set the answer treatment to 'Custom Plug-in' and click 'Apply'.

The feasible light generation process should then halt in your code each time it calls `treat()` with a candidate word.

Chapter 15

Decks

15.1 Introduction to decks

You can run Qxw in 'batch mode' (i.e., non-interactively) from the command line, supplying it with a problem for its filler to solve. In this mode Qxw does not start up its graphical user interface. The file specifying the problem is called a 'deck', after the name used for a set of punched cards used to hold data or programs for early computers. A Qxw deck is a plain text file, conventionally with extension `.qxd`.

To run Qxw in batch mode on a deck, use the `-b` option on the command line. Under Linux the command

```
qxw -b cube.qxd
```

will cause Qxw to load the file `cube.qxd`, run its filler on the problem specified on that file, and, if a fill is found, output the result. Qxw sets a return code of 0 if a fill is found, 4 if no fill is found, and 16 if an error is encountered when processing the deck: this, coupled with the straightforward file format of the deck, makes is easy to use Qxw as part of a script to automate the exploration of grid designs.

If Qxw is run from a Windows command prompt the system will by default launch it in such a way that all output is discarded. To capture the output it is necessary to 'redirect' it to a file. From the command prompt, change to the directory containing your deck file and issue the following command.

```
"C:\Program Files (x86)\Qxw\Qxw.exe" -b cube.qxd >output.txt 2>errors.txt
```

That will cause the results to be written to the file `output.txt` and any error messages to `errors.txt`, also in the current directory.

You can add `-a` and `-d` options to the command line to set the initial alphabet and dictionaries in just the same way as when running Qxw interactively, although these can be overridden by directives in the deck (see Section 15.3.1). When running in batch mode Qxw does not load a 'preferences' file: again, you can configure relevant preferences using directives in the deck.

Blank lines in a deck, and lines whose first non-space character is hash ('#') are ignored. The latter can be used to introduce comments into a deck should you be so moved.

15.2 Entries and words

At its heart, Qxw's filler works on *entries* and *words*. An entry can hold a single character from the current alphabet. In a deck each entry is given a name, which can be any sequence of letters, digits, underscore ('_') and the dollar sign ('$'). Entry names are case sensitive and may consist of up to 31 characters.

A word consists of a sequence of entries, and is specified as a single line in the deck. In the simplest case, the filler's job is to assign a character from the current alphabet to each entry such that each sequence of entries comprising a word is a string found in the available dictionaries.

Consider the following two-line deck.

```
ent0 ent1 ent2
ent2 ent1 ent0
```

Here `ent0`, `ent1` and `ent2` are the names of three entries, and so the filler will be looking for a set of three characters. The first line of the deck specifies a word comprising those three entries in order; the second line of the deck specifies a word comprising those three entries in reverse order. So the problem the filler has to solve is to find a three-letter word in the available dictionaries whose reverse is also in the available dictionaries. Run Qxw on this deck and a typical result would be as follows.

```
W0 ERA
# era
W1 ARE
# are
```

This report means that Qxw's filler has found that the assignment of 'E' to entry `ent0`, 'R' to entry `ent1` and 'A' to entry `ent2` results in both the forward and reverse sequences being words, labelled as 'W0' and 'W1' in the output. The lines starting '#' give the forms of the words found in the dictionaries (like the entries in the feasible word list display when using Qxw interactively).

In the example deck shown in Figure 15.1 the sixteen entries are labelled from '00' to '15'. If you look at the pattern of entries forming the words you will see that the problem specified by this deck is to find a four-by-four word square. Typical output from running Qxw on this deck is shown in Figure 15.2.

```
00 01 02 03
04 05 06 07
08 09 10 11
12 13 14 15
00 04 08 12
01 05 09 13
02 06 10 14
03 07 11 15
```

Figure 15.1: Deck to generate a four-by-four word square

Now suppose you wanted to create a four-by-four word square where the words running across and down are the same. Although you can set this up in Qxw interactively using free lights, it is more straightforward in this instance to create a deck. Instead of sixteen independent entries

```
W0 HULL
# Hull, hull
W1 HALT
# halt
W2 ASIA
# Asia
W3 USER
# user
W4 LEEK
# leek
W5 LIEU
# lieu
W6 TRUE
# true
W7 LAKE
# lake
```

Figure 15.2: Typical result from word square deck

there are now only ten, as those below the leading diagonal of the square must be the same as their counterparts reflected in that diagonal; and of course there are now only four words to be found. A suitable deck is shown in Figure 15.3.

```
00 01 02 03
01 04 05 06
02 05 07 08
03 06 08 09
```

Figure 15.3: Deck to generate a symmetrical four-by-four word square

15.2.1 Initialising entries

You can constrain which characters are allowed in certain entries by adding an 'initialiser'. An initialiser takes the form of an equals sign ('=') followed by a series of characters (with no intervening space), and can appear after any entry in a word. If an initialiser consists of a single character, then it specifies that the entry before it must be set to that character. So, if we modify our simplest example above as follows:

```
ent0 =J ent1 ent2
ent2 ent1 ent0
```

then the first letter in the first word is forced to 'J', and the filler might find the following solution.

```
W0 JAR
W0 # jar
W1 RAJ
W1 # raj
```

If an initialiser consists of more than one character, then it constrains more than one of the

previous entries in that word. For example, if the first line of the word square example were changed to read

```
00 01 02 03 =MARK
```

then Qxw would look for word squares reading 'MARK' across the top.

Initialisers can also contain letter choices using the same syntax as in the Cell contents dialogue: see Section 12.3. So for example you could write

```
00 01 02 03 =[LM]@RK
```

and then, depending on your available dictionaries, the top row of the word square might read LARK, LURK, MARK, MORK or MURK.

15.3 Directives

Directives are used in a deck to control various aspects of how the filler goes about its job. They cover many of the same functions as the various dialogues available in interactive mode. Each directive in a Qxw deck occupies a line by itself, and always starts with a full stop ('.') character; directives—indeed, any line in a Qxw deck—can be indented by spaces or tabs if desired. Each directive has two forms: a fully spelled out form and a two-character abbreviated form. Both are case-insensitive. For clarity we will use the full forms in capitals in the examples here.

Directives are divided into two types: *global* directives, which take effect over the whole of the deck and which must be specified at the beginning of the deck; and *local* directives, which only take effect over a subset of words in the file.

15.3.1 Global directives

The `.ALPHABET` directive (short form `.AL`) specifies the alphabet to be used, using one of the abbreviations listed in Section 9.6. For example,

```
.ALPHABET AZ09
```

sets the alphabet to be the Roman letters A–Z plus the digits 0–9.

The `.DICTIONARY` directive (short form `.DI`) specifies a dictionary to be loaded. It is followed by a dictionary slot number (a digit from 1 to 9) and then the rest of the line gives the filename. So for example

```
.DICTIONARY 3 /usr/share/dict/words
```

will cause dictionary slot 3 to use the file `/usr/share/dict/words`. This has equivalent effect to supplying that filename under 'File' in the Dictionaries dialogue: see Chapter 9. The 'File filter' and 'Answer filter' strings can be set using the `.FILEFILTER` (short form `.FF`) and `.ANSWERFILTER` (short form `.AF`) directives, which have the same syntax. So you might say

```
.DICTIONARY 3 /usr/share/dict/words
.ANSWERFILTER 3 a.*a
```

to select only words that contain at least two 'a's. Single-entry dictionaries can also be created using the `.ANSWERFILTER` filter directive without supplying a dictionary filename or file filter.

```
.DICTIONARY 1
.FILEFILTER 1
```

If the deck does not contain any `.DICTIONARY`, `.FILEFILTER` or `.ANSWERFILTER` directives then

Qxw will attempt to find a suitable dictionary to populate slot 1 in the same way as it does in interactive mode.

The `.RANDOM` directive (short form `.RA`) sets the degree of randomness used in the filling algorithm. It is equivalent to the setting under 'Autofill preferences' in the Preferences dialogue. The directive

```
.RANDOM 0
```

forces a deterministic fill, so that supplying the same problem to the filler will always yield the same result;

```
.RANDOM 1
```

introduces a small amount of randomness; and

```
.RANDOM 2
```

introduces a high degree of randomness. The default in batch mode is that fills are deterministic.

The `.UNIQUE` directive (short form `.UN`) specifies whether duplicate answers and lights are allowed in the fill. It is equivalent to the setting under 'Autofill preferences' in the Preferences dialogue. The directive

```
.UNIQUE 0
```

allows duplicates, while the directive

```
.UNIQUE 1
```

prevents duplicates. The default in batch mode is that duplicates are not allowed.

The remaining global directives are concerned with configuring an answer treatment. You may find it helpful to experiment first with using the answer treatment facilities interactively and to compare the directives with the options provided in the Answer treatment dialogue (menu item *: Autofill-Answer treatment*). The behaviour of answer treatments is described in detail in Chapter 14.

The `.TREATMENT` directive (short form `.TR`) is followed by a number that specifies which answer treatment is to be used according to the following table.

Treatment number	Description
0	None (default)
1	Playfair cipher
2	Substitution cipher
3	Fixed Caesar/Vigenère cipher
4	Variable Caesar cipher
5	Misprint (general, clue order)
6	Delete single occurrence of character
7	Letters latent: delete all occurrences of character
8	Insert single character
9	Custom plug-in
10	Misprint (correct letters specified)
11	Misprint (incorrect letters specified)

The `.MESSAGE` directive (short form `.ME`) allows you to set one of the two messages that are passed to the answer treatment. It is followed by a digit, which must be '0' or '1', and then the remainder of the line is taken as the message text. So, for example, the directive

```
.MESSAGE 1 Oh, what a tangled web we weave
```

would set message 1 to the given text.

The .MESSAGEALLOCATE directive (short form .MA) allows you to control the discretionary modes of the filler: see Chapter 6 for more information. It is followed by a digit, which must be '0' or '1', specifying a message number; and then a further digit, which must be '0', '1' or '2', which specifies the message letter allocation mode according to the following table.

Allocation mode	Description
0	in clue order, first come first served (default)
1	in clue order, at discretion of filler
2	in any order, at discretion of filler

The .MESSAGECONSTRAINTS directive (short form .MC) applies constraints to the discretionary modes of the filler. It is followed by a digit, which must be '0' or '1', specifying a message number; and then the remainder of the line is taken as the constraint string. See Chapter 6 for more information.

The .TREATEDANSWERMUSTBEAWORD directive (short form .TW) determines whether the result of the answer treatment process must appear in the dictionaries selected for the word in question. It is followed by a single digit, which must be '0' (the default) to allow arbitrary strings as the result of answer treatment or '1' to restrict to strings that appear in the dictionaries.

The .PLUGIN directive (short form .PI) specifies the filename of a custom answer treatment plug-in. The text from after the directive to the end of the line is taken as the filename, which must include the full path to the desired file.

15.3.2 Local directives

The directives described in this section can occur anywhere in a deck. They affect only words that appear after them in the deck. They correspond to settings available in the Light properties dialogue (see Chapter 12).

The .USEDICTIONARY directive (short form .UD) gives the list of dictionaries that can be used for a word. It is followed by a list of dictionary numbers without spaces. So, for example,

```
.USEDICTIONARY 134
```

allows dictionaries 1, 3 and 4 to be used for filling subsequent words in the deck. At the beginning of the deck only dictionary 1 is selected.

The .ENTRYMETHOD directive (short form .EM) gives the allowable 'entry methods', that is, the ways that letters from dictionary words can be assigned to entries in a word. It is followed by a string of characters without spaces specifying entry methods according to the following table. You can use the letter codes or the symbol codes as you prefer.

Entry method	Letter code	Symbol code
Normal (forwards)	f	>
Reversed	r	<
Cyclically permuted ('clockwise')	c	)
Cyclically permuted and reversed ('anticlockwise')	a	(
Any other permutation ('jumble')	j	@

So, for example,

```
.ENTRYMETHOD frcaj
```

would allow subsequent words to be filled with any permutation of strings from the available dictionaries; and

```
.ENTRYMETHOD )(
```

would allow cyclic permutations and reversed cyclic permutations only. At the beginning of the deck only normal forwards entry is selected.

The `.TREATMENTENABLE` directive (short form `.TE`) enables answer treatment for subsequent words. Similarly, `.TREATMENTDISABLE` (short form `.TD`) disables it. At the beginning of the deck treatment is enabled, but the default answer treatment is 'None' so this has no effect.

15.3.3 Blocks and scope rules

You can further restrict the extent of the effect of a local directive by grouping a set of consecutive words into a 'block'.

A block begins with an open-brace character ('{') on a line on its own, and ends with a matching close-brace character ('}'), also on a line on its own. Blocks may be nested within one another, and you can use indentation to make the block structure clearer if you wish. A local directive only has force until the end of the block in which it appears, including within any sub-blocks nested inside. When Qxw reaches the end of a block as it reads in the deck, any parameters modified by local directives are restored to the values they had at the beginning of that block.

Figure 15.4 shows a more complicated example of a deck. The deck describes a five-by-five word square where the across words are drawn from an English dictionary while the down words are drawn from a French dictionary. The central across and down words are entered reversed, while all other words are entered normally. This is achieved by enclosing the affected words within a block: the effect of each block is to ensure that each `.ENTRYMETHOD` directive only affects one word.

```
.DICTIONARY 1 mydictionaries/english.txt
.DICTIONARY 2 mydictionaries/french.txt
.USEDICTIONARY 1
00 01 02 03 04
05 06 07 08 09
   {
   .ENTRYMETHOD <
   10 11 12 13 14
   }
15 16 17 18 19
20 21 22 23 24
.USEDICTIONARY 2
00 05 10 15 20
01 06 11 16 21
   {
   .ENTRYMETHOD <
   02 07 12 17 22
   }
03 08 13 18 23
04 09 14 19 24
```

Figure 15.4: Deck to generate a mixed-language five-by-five word square

Chapter 16

Keyboard and mouse command summary

16.1 Keyboard commands

Keystroke	**Menu item**	**Effect**
A...Z, 0...9		enter character in grid (varies depending on current alphabet)
Space		advance cursor one position in current direction
Backspace		retreat cursor one position in current direction
Home		move cursor to start of light
End		move cursor to end of light
Tab		delete letter from cell and advance cursor one position in current direction
Return	*Edit-Bar before*	add bar before cursor position in current direction
Insert, ','	*Edit-Solid block*	add block at cursor position
Delete, '.'	*Edit-Empty*	make empty cell at cursor position
PageUp		change current direction one step anticlockwise
PageDown, '/'		change current direction one step clockwise
$\leftarrow, \rightarrow, \uparrow, \downarrow$		move cursor left, right, up, down
control-A	*Autofill-Accept hints*	accept hints (enter suggested letters in grey into grid)
control-C	*Edit-Cutout*	make cutout in grid
control-D	*Edit-Free light-Shorten selected*	remove last cell from selected free light

Keystroke	**Menu item**	**Effect**
control-E	*Edit-Free light-Extend selected*	add cursor position as new cell to selected free light
control-G ('go')	*Autofill-Autofill*	run automatic filler
control-I ('in')	*Edit-Cell contents*	change contents of cell
control-L	*Edit-Light contents*	change contents of light
control-M	*Edit-Merge with next*	merge current cell with next cell in current direction
control-N	*File-New-Current shape and size*	start new grid
control-O	*File-Open*	open a previously-saved file
control-Q	*File-Quit*	quit program
control-S	*File-Save*	save grid
control-X ('expunge')	*Edit-Clear all cells*	clear all cells
control-Y	*Edit-Redo*	redo last undo
control-Z	*Edit-Undo*	undo last change
control-Minus	*Edit-Zoom-Out*	zoom out
control-1	*Edit-Zoom-50%*	zoom to 50%
control-2	*Edit-Zoom-71%*	zoom to 71%
control-3	*Edit-Zoom-100%*	zoom to 100%
control-4	*Edit-Zoom-141%*	zoom to 141%
control-5	*Edit-Zoom-200%*	zoom to 200%
control-Plus	*Edit-Zoom-In*	zoom in
shift-A	*Select-All*	select all lights or cells
shift-C	*Select-Current cell*	add cell under cursor to selection
shift-F	*Select-Free light*	select first or next free light
shift-I	*Select-Invert*	invert current selection
shift-L	*Select-Current light*	add light going through cursor in current direction to selection
shift-M	*Select-Cell mode <> light mode*	switch between selecting cells and lights
shift-N	*Select-Nothing*	reset selection
shift-control-G	*Autofill-Autofill selected cells*	run automatic filler on selected cells only
shift-control-X	*Edit-Clear selected cells*	clear selected cells

16.2 Mouse commands

Action	Effect
left-click on cell edge	add/remove bar*
left-click on cell corner	add/remove block*
left-click on cursor	change current direction
other left-click on grid	move cursor
left-click in feasible word list	enter word in grid at cursor position
shift left-click	select/deselect cell
left-click and drag, holding shift	continue selecting/deselecting cells
shift right-click	select light in current direction
right-click and drag, holding shift	continue selecting/deselecting lights in current direction (not available in all Windows versions)
right-click in feasible word list	open context menu (banishment, copy to clipboard, lookups)
scroll wheel	scroll grid view
scroll wheel, holding control	zoom

* If function is enabled: see Section 10.1.

Index