Mental Models and Interactive Statistics: Design Principles

R.W. Oldford
Department of Statistics & Actuarial Science
University of Waterloo

Abstract

The debate about the appropriate computer human interface – direct manipulation versus command line – is an old one and a false one. That it regularly arises is a consequence of inappropriate software design. An ideally designed system would freely mix the two.

D.A. Norman's (1988) *Design of Everyday Things* is condensed to five essential principles which are applicable to the design of interactive statistical systems. These are described and illustrated by the design of the experimental statistical software system called Quail. While the principles apply types of interface they are illustrated primarily for a direct manipulation one.

Examination of these principles continually points to the value of modelling statistical concepts directly in the base software system. It is argued that such a system would be well positioned to freely mix interface styles to best suit the analysis.

1 A non-debate

Like most debates, that purported to exist between command language and direct manipulation interface proponents is an artificial one, constructed only to provoke discussion about the merits and scope of each position. The position at either extreme is untenable; so the challenge becomes to determine the lay of the middle ground.

Collectively, we have been exploring this middle ground since the invention of the first digital computer. The history of programming languages is the history of increasingly powerful computational abstraction. Where once we manipulated binary digits in registers, we now manipulate objects, procedures, and graphics which, while ultimately still only stored binary digits, are taken to represent all manner of things from architectural designs, to mathematical theory, to small virtual worlds.

The challenge to statistical computing is to determine the appropriate abstractions for statistical design and analysis, to figure out how to implement them computationally, and to develop interactive tools which facilitate their use.

Meeting this challenge requires careful design, both of appropriate programming abstractions and of graphical user interface. The goal is to seamlessly integrate the two approaches.

2 A difficult game

The following rather difficult two person game is based on combinatorics but neatly illustrates a number of useful design principles:

1. Begin with the set of non-zero digits, $\{1, \ldots, 9\}$.
2. The two players take turns selecting a digit from the set.
3. Once selected, a digit is removed from the set of available digits making it unavailable for future selections by either player.
4. The first player to have three digits which sum to 15 is the winner.
5. The game ends in a draw if all of the digits have been selected and neither player has three digits summing to 15.

Imagine playing the game. The goal of the game is simply defined. The moves are relatively straightforward. Yet the game is difficult to play. Why?

Several factors contribute to the difficulty. To begin with, the game is introduced and described in such a way that one has the perception that it is difficult even before playing it. The gratuitous mathematical[1] description especially heightens this impression for non-mathematical readers.

Playing the game reinforces this perception. At each turn at least three sets of information need to be managed: your set of selected digits, your opponent's, and

[1] This is the principal distinction from the game as described in Norman (1988, pp. 126-7)

those remaining. Each turn requires a choice between several competing options. Each option needs to be explored one or two moves further out in order to assess its merits.

The slow, serial nature of conscious thought and the known limitations of short term memory make management of this complexity difficult for most people. We need to introduce some tricks to help us out.

The first thing most people will do is write the set of numbers down and record the selections of both players. Next, writing down all possible triples of digits summing to 15 helps in determining the possible outcomes of different moves.

Further analysis of the possible triples suggests the following remarkable tabular arrangement (from Norman, 1988, page 126):

8	1	6
3	5	7
4	9	2

Not only do the three rows, three columns, and two diagonals each sum to 15 but they actually exhaust the set of triples which do! Moves can be identified directly on the display by, say, circling our selections and crossing out those of our opponent. The digits are now superfluous and the object of the game is to get three circles (or crosses) in a straight line – tic-tac-toe!

By building the structure of the game into the display, the complexity is substantially reduced and the players are free to concentrate more on the play and on developing winning strategies.

Tic-tac-toe is an example of a wide and deep structure – many choices at each turn, many turns in sequence. Figure 1 shows the essential structure of the game when the first player (O) selects the middle square on the first turn. From this initial move, player one can now follow a strategy which ensures that the game ends in either a draw or a win.

3 Interactive statistical analysis

If we imagine that the equivalent of a player's turn in an interactive statistical analysis consists of a single interaction with the computer (a command, a mouse click, or any other communication with the underlying program), then the structure of an interactive statistical analysis is certainly a wide and deep one. Even so, it is a structure which differs fundamentally from that of tic-tac-toe.

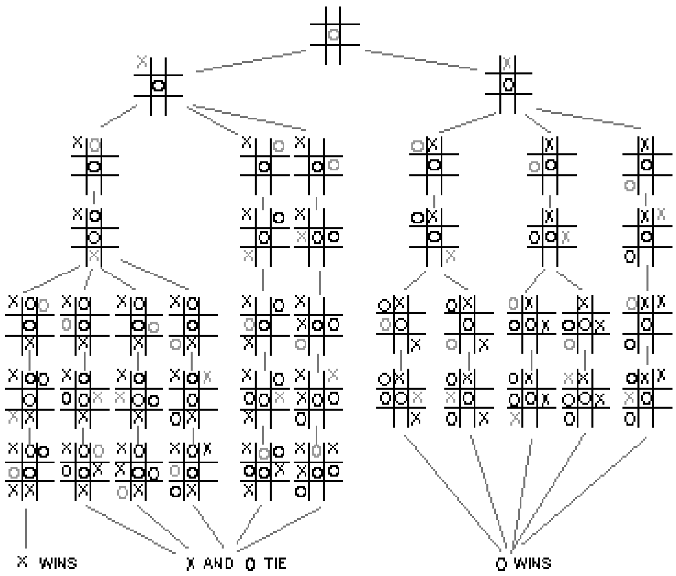

Figure 1: Tic-tac-toe: A wide and deep structure.

The typically exploratory and experimental nature of interactive statistical analysis ensures that its structure is far wider and deeper. It is difficult to imagine that we could a priori determine the length of the longest possible path in an analysis let alone identify all conceivable actions an analyst might wish to take at any given step. This open-ended nature means that any computational environment which presupposes a closed structure (like that of tic-tac-toe) will eventually overly constrain the analyst.

Designers of statistical software often make the error that the closed world approach is sufficient. Examination of the *Interface proceedings* from the 1970s will reveal that command line systems began this way but soon had statisticians hitting the edges of the system's possibilities. To quote but one source:

> "Even the statistician's operating language, context and syntax, became formed from the names of available programs and functions. In order to regain his individuality, it became necessary for the thinking statistician to teach computers to do his wishes ... That is, he had to learn to program or hire a programmer." ... Guthrie (1975, pp 8-9)

In fairly short order, 'macro' capabilities which permitted programming were added to most command language systems.

Similarly, direct manipulation interfaces are often first built for either wide and shallow structures or narrow and deep ones. A simple example of the former would be a pop-up menu from which to choose a singe item. A more complex example would be a stand-alone interactive graphic – such as a brushable scatterplot matrix or a grand tour type dynamic scatterplot. An example of a narrow and deep structure would be the sequential prompting for information preparatory to proper execution of some task.

Direct manipulation interfaces can be made to work very well for either kind of structure (wide and shallow or narrow and deep). But, as with the early command line interfaces, if this is all that the user can do then the software soon becomes overly restrictive.

When it becomes apparent that the interface does not meet perceived needs, or perhaps in response to users requests for certain functionality, designers often give into *creeping featurism*, the first of D.A. Norman's (1988) "two deadly temptations". The result is that 'features' are added to the existing design which, if not carefully done, quickly add to its complexity (Norman, 1988, believes that the complexity grows with the square of the number of features). The resulting more complicated design can be rendered less useful than the original simpler design.

Creeping featurism is to be avoided. If it cannot, the only solution is in a careful organization of the design: to modularize, to divide and conquer. Gratuitous addition of features adds unneeded complexity.

Sometimes complexity is introduced intentionally – as if it were desirable itself. This is Norman's second deadly temptation – the *worshipping of false images*. In a graphical user interface we might see all features made available at once with gratuitous use of multiple colours, buttons, knobs, etc. The false image is that a complex interface implies technical sophistication, when the reality is that complexity leads to confusion. Again the best advice is avoidance – keep things as simple as possible.

Interactive statistical analysis is substantively more difficult than the game presented at the beginning of Section 2. Yet it is hoped that software can be designed which, like the tic-tac-toe representation of Section 2, will considerably simplify the interaction between user and system. Complexity is to be managed by the system so that the user is freed to concentrate on the analysis. Achieving this is no easy task.

A mixed strategy of direct manipulation and keyboard commands is desirable. Moreover, because some tasks will always be outside the existing design, programmatic control needs to be available to the user. Programmatic control can come from either the keyboard or from direct manipulation – an early statistical example of some programming functionality in a graphical user interface can be seen in Desvignes and Oldford (1988) – although the keyboard is likely to remain the programming input device of choice.

Having the user able to concentrate on the analysis implies that interaction with the system needs to be in terms which are familiar to an analyst as opposed to a programmer. Statistical analysis concepts must form the fabric by which direct manipulation interfaces and text-based programming are seamlessly integrated. This is the fundamental design challenge for interactive statistical systems.

4 Design Principles

Good design follows identifiable principles – whether it is the design of a door handle, a video cassette player, or a teapot. These principles apply no less to the design of interactive statistical analysis systems.

In his book, *The Design of Everyday Things*, D.A. Norman (1988) develops several principles of good (and bad) design which he illustrates using objects from every day life. These can be reduced to the following five principles of design:

- M. Match mental models
- S. Simplify structure
- C. Constrain
- E. Expect error
- F. Failure? Fix on a standard.

The principles are discussed in turn below and illustrated with the design choices of the interactive statistical system called *Quail* (see Oldford *et al* below to access the software, and Oldford, 1998, for some further detail on Quail). Figure 3 shows the screen of a typical session in Quail.

M. Match mental models

The designer has a certain model for interactive statistical analysis in mind which he/she tries to capture in the design. This design is then implemented as a working system (see Figure 2). The user too has a model for an interactive statistical analysis in mind. Working with a particular system forces the user to construct a working model of that system, which can be quite different from both his/her general mental model and that of the designer. The only communication between the mental models of the user and the designer is through the implemented system.

Ideally these two models should match and the system implementation should follow them closely. The system

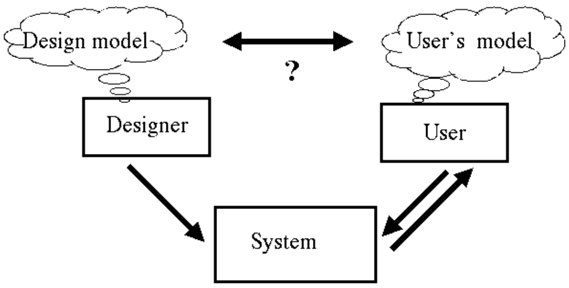

Figure 2: Matching mental models.

would then be more immediately useful to the analyst. Moreover, it would be more easily extended to new areas by the user; the step from user to designer would be small.

To better match models, Norman (1988) recommends the designer:

1. Use existing common knowledge,
2. Communicate the model,
3. Use the model, and
4. Reinforce the model.

There are two types of existing knowledge common between the designer and the user of a statistical analysis system. The first is the everyday sort such as switches, push-buttons, dials, and gauges which can be mimicked to good effect in a visual display. The familiarity of the visual representation makes its use obvious to the user.

The second is the knowledge about statistical analysis which, although specialized, is shared by both user and designer. In the early command line systems this was evident primarily in the names of commands. In early direct manipulation interfaces, it is first evident through common interactive statistical graphics.

Strictly speaking, it is not necessary to make maximal use of the existing common knowledge. If the designer is successful in communicating, using, and reinforcing the model then the user can be trained to adopt it. In the extreme, a user who learned about statistical analysis solely by interacting with a single statistical system might very well have a mental model essentially coincident with that of the designer. When computational resources are scarce this is particularly desirable.

Ever faster processors and cheaper memory means that we can now afford to devote more resources to system software which better models existing statistical knowledge. Matching fundamental system components directly to the basic structures of statistical analysis maximizes use of the specialized knowledge common between designer and user.

This approach has important consequences. First, communicating the system model to a statistically trained user should be straightforward. Second, the designer writing code in terms of these fundamental components is actively exercising the model and providing the user with easily understood means to tailor the system to his/her needs. Third, such use reinforces the model.

Not surprisingly, there is abundant structure in statistical knowledge relevant to interactive analysis. Many statistical concepts are quite naturally represented as data structures, for example:

- Datasets, variates, cases, ...
- Random variates, parameters, likelihoods, ...
- Response models, linear models, smooths, ...
- Fitted models, estimates, ...
- Borel Sets, measures, probability measures, ...
- Mathematical functions, survivor functions, state transition intensity functions ...

Object-oriented programming seems to be particularly well-suited to modelling these concepts. Classes are used for each of these structures with inheritance hierarchies which follow the so-called 'IS-A' relations as in a cauchy distribution IS A student distribution with one degree of freedom and hence the class *cauchy-distribution* appears as a sub-class of the *student-distribution* class. The reasoning is that a user who has an instance of a *cauchy-distribution* should expect the same functionality from it as any other instance of a *student-distribution* with some behaviour specialized (e.g. moment calculations). In this way the known relations between statistical concepts is modelled by the definition of the classes and class hierarchies.

Generic functions and specialized methods are also useful in modelling statistical concepts which might not be represented in a hierarchy. In *Quail* for example, the generic function *random-value* applies equally to any instance of a distribution and to any dataset. This is so that the user may regard the dataset as an empirical distribution, as in resampling procedures, without changing its class to some kind of distribution.

Nothing about matching mental models is directed exclusively at either programming language interfaces or at direct manipulation interfaces but rather is directed at both. The fact that so many statistical concepts can be directly modelled as data structures has an important consequence for this approach. Namely, graphical user interfaces can be built by laying out visual displays of the statistical objects themselves – direct manipulation

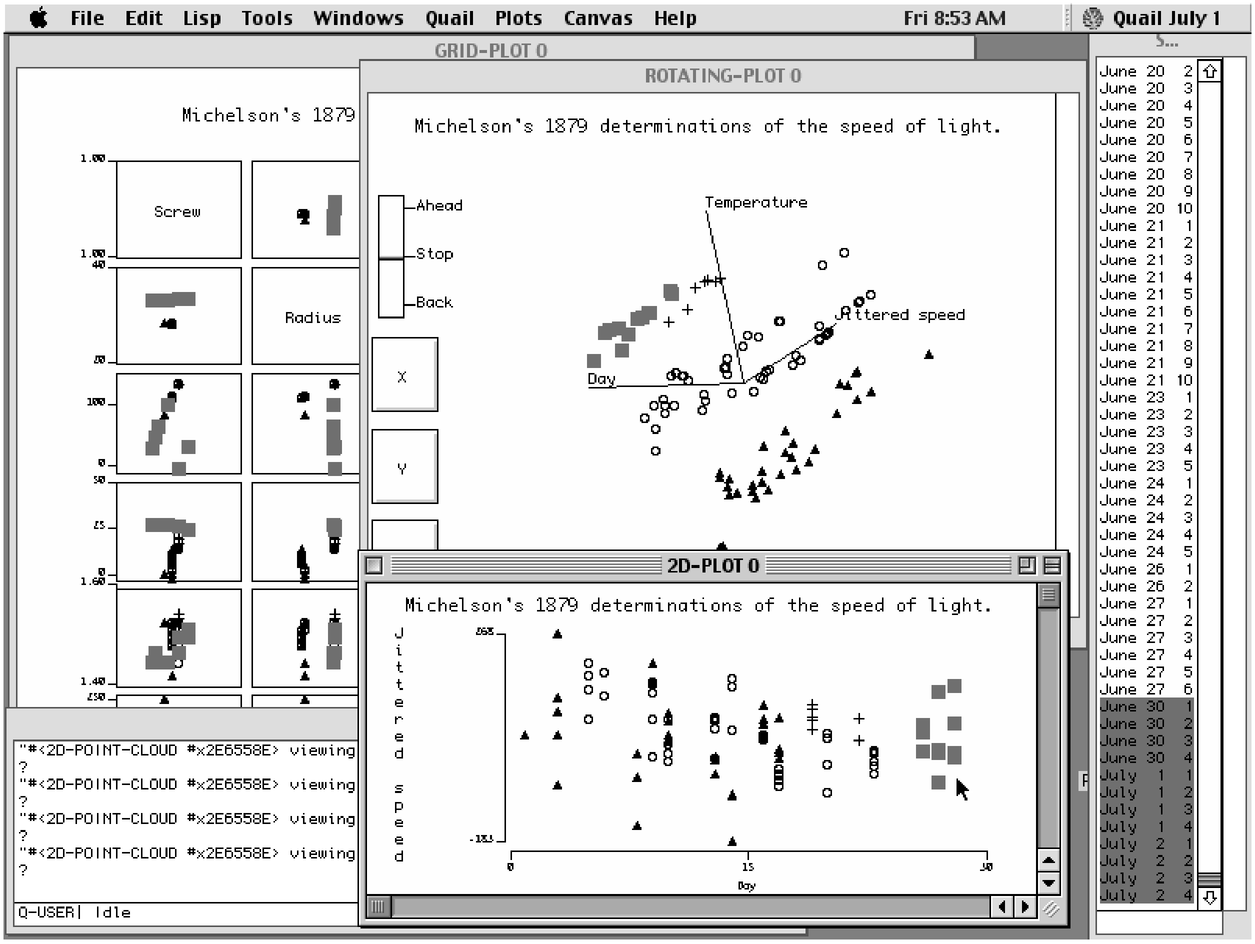

Figure 3: A typical screen shot from Quail.

and programmatic or command line interfaces use the same structures as arguments. The information relevant to both is stored within the object so that the transition between the two interface styles is relatively straightforward.

In *Quail* this is accomplished by having every graphic component maintain a pointer to the relevant statistical object being displayed – this is its *viewed object* (e.g. see Hurley and Oldford, 1991). A fitted line, for example, would have a pointer to the fit-object which it visually represents in a plot. Conversely, every statistical object in *Quail* returns a graphics object when asked for its display (see Oldford, 1998, 1997 for more discussion).

S. Simplify structure

The second principle is to simplify structure wherever possible. Here Norman (1988) recommends that we

1. Hide complexity,
2. Provide appropriate feedback,
3. Automate where possible; where not, change the nature of the task, and
4. Allow the user to add complexity.

It is the nature of interactive statistical analysis that it be complicated and the typical screen-shot from *Quail* (see Figure 3) seems to reinforce the point. There we see five separate windows. Four of these windows are graphic windows – a 2d scatterplot, a 3d rotating scatterplot, a scatterplot matrix, and an interactive list of cases. Each of these provides some facility for direct manipulation. The fifth window (in the bottom left corner of the display) is a type-in command line compiler/interpreter where arbitrarily complex programs can be written and subsequently used (the question mark is the prompt). The commands which produced the remaining four windows could, for example, have been typed in at this window and executed (although the plots for this session were produced entirely without typed input).

The complexity seen here is, in a certain respect, unavoidable, as it represents complexity that the user chooses to see simultaneously. Otherwise some of the windows could have been closed by the user.

Even so, much complexity is hidden away. The dataset being examined, for example has 15 different variates measured on each of 100 observations. Each plot maintains a pointer to the entire dataset. Moreover, each plot is itself made up of several pieces – axes, labels, point-clouds, titles – each one of which maintains a pointer to its own appropriate viewed-object. This complexity is hidden until accessed by the user.

At the top of the screen in Figure 3 is the Quail menubar. Each word in the bar ('File', 'Edit', etc.) is the title a pull-down hierarchical menu. Figure 4 shows part of the 'Quail' pull-down menu which provides gen-

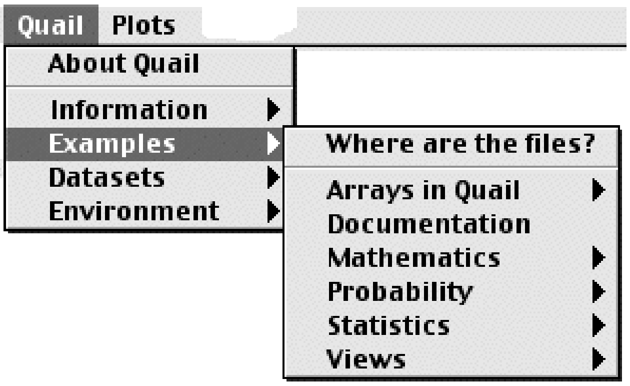

Figure 4: The pull down Quail menu.

eral access to information – help, tutorial examples, etc.

The 'Plots' menu illustrates the use of both recommendations 2 and 3. If, for example, the 'Scatterplot' menu item was selected as shown in Figure 5 then one of three things would automatically occur depending upon the display context.

The top-most window is always displayed slightly differently than the other windows; in Figure 3 the top-most window is titled '2D-PLOT-0'. If the top-most window is the type-in window (as would be the case at the beginning of any session) then the user is first prompted for the dataset to be displayed in the scatterplot and then asked to select two variates from those associated with that dataset. If on the other hand, the top-most window is already a plot of some sort, then it is assumed that the user intends the new scatterplot to be of the same data and so he/she is prompted only for the variates to be used. Finally if, as is the case in Figure 3, the top-most window is a plot with a subset of the data selected (shown as gray square boxes near the cursor in

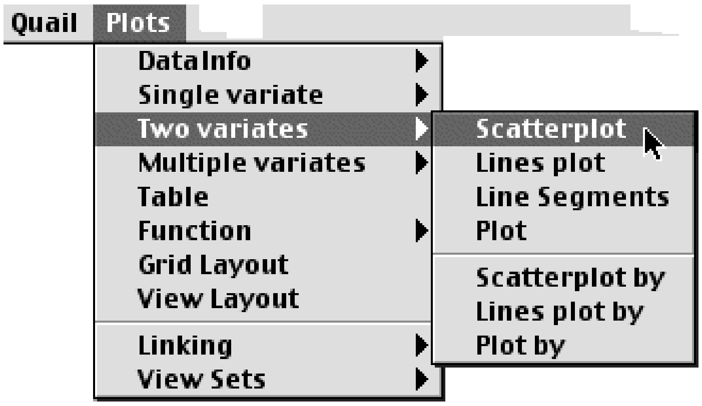

Figure 5: The pull down 'Plots' menu.

Figure 3), then after the user is prompted to select variates a scatterplot is produced for the *selected subset* of the data.

Highlit points in a scatterplot mean the user has focused attention on them and this is assumed to be the case for the automated functionality. When the new scatterplot of the subset comes up, all points will be highlighted since they were the ones selected in the original plot. This clear and immediate feedback to the user should help them understand the system model which the automation has presupposed. A considerable pay off occurs for example, when scatterplots of a subset of the data are desired separated and properly arranged according to the value of one or more categorical variates (including point colour). Simply highlighting the desired subset and selecting 'Scatterplot by' from the 'Plots' menu produces the desired result.

Considerable complexity can be added by the user either programmatically (using the fairly general program structures available in Quail) or interactively via direct manipulation. For the latter case, the variety of point symbol shapes in the displays of Figure 3 were determined by direct interaction with the displays. Besides changing styles (e.g. colour, shape) of the graphics on display, it is also possible to re-arrange their positioning, add other display components and so on.

A good deal of this simplification has been made easy to handle by having the underlying system code match the statistical concepts shared by designer and user alike. The models are reinforced directly by the display organization and by interaction with the display (e.g. point-symbols represent cases, so selected point-symbols represent selected cases).

C. Constrain

We saw the power of this principle at work in the game introduced in the second section. The general admonition here is to exploit all known constraints whether natural or artificial. This will often lead to considerable simplification in the user's model.

A simple natural mapping which is promoted in the display is that between the push-button and speed bar controls of the 3d rotating scatterplot and the 3d point-cloud. The spatial proximity of the controls to the point-cloud strongly suggests that the cloud is indeed the target of the controls. This is preferable to placing the controls at a spatially distant location such as the Quail menubar. Items appear there principally because they apply generally to all statistical graphics at any point in the analysis. More complex examples of controls laid out near their targets are given in Oldford (1997).

Action items appearing on the menubar are typically spatially distant from their target and so something like the selection constraint described in the last section becomes necessary. This constraint is made less artificial by consistently applying it. Graphics which represent the same viewed object show themselves highlighted in all displays when they are highlighted in one. For example, all point-symbols representing the cases selected in the scatterplot show themselves highlighted in all other visible displays in Figure 3. Similarly, they share other display style properties such as size and shape. This linking between graphics also occurs between graphics of different types provided they have the same viewed object (e.g. point-symbols and the case-labels of the case-list appearing on the right in Figure 3).

Another means of reinforcing the rule that user selection determines the focus or target is through pop-up menus associated with each graphical component in a display. This is achieved by employing a 'three-button' mouse. Here left-button mouse click over a component selects that graphic and consequently all information (viewed-object, etc.) attached to it. Middle-button selection pops a menu up directly at the mouse position thus associating it spatially with the underlying selected graphic. Figure 6 shows the menu which pops up over a 2D point cloud such as that found in the scatterplot of Figure 3.

The imposed constraints make it clear that selections from Figure 6 apply to the point-cloud beneath the menu. Consequently, specialized menus can be constructed to correspond with each type of graphical component in any display. Middle-button selection over an axis for example produces the pop-up menu of Figure 7. Here actions that are appropriate to be taken only on an axis are gathered for direct application to the selected axis.

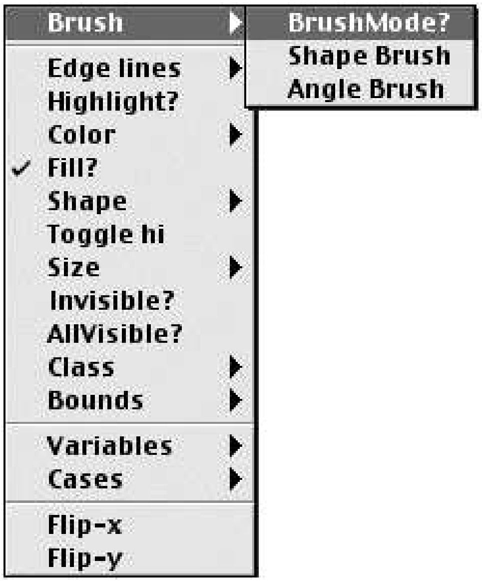

Figure 6: The middle button menu for point clouds.

In Quail, pop-up menus exist which are tailored to each kind of graphical component provided by the system. These are always available allowing the user to interact with every display as suits their fancy.

This kind of specialization is feasible because the software components of the graphic directly model the corresponding statistical graphical concepts. As with statistical concepts more generally, those constraints which naturally exist between concepts are encoded within the software representing those concepts typically, though not always, through a class inheritance hierarchy and method definition.

E. Expect error

This obvious principle is often forgotten, perhaps because it is so difficult to handle well. To the best of their ability, designers need to anticipate the inevitable errors that a user will make. Of these, those that can be prevented by design should be. Those that remain should be handled gracefully with feedback to the user so that they are at least aware of the error and have some chance for recovery.

In Quail, the previous principles have been applied to the design with the hope that error is minimized. Commonly used functions (e.g. *scatterplot*) will prompt the user for missing arguments. Example files providing tutorial instruction and an interactive help system help educate the user on proper use of at least the programmatic interface to Quail. When all else fails, errors at execu-

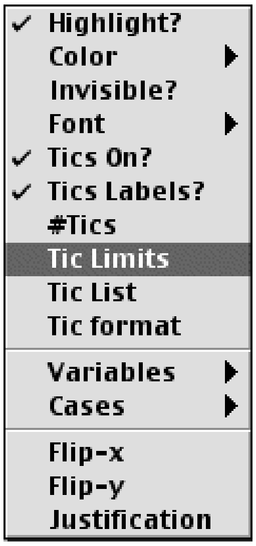

Figure 7: The middle button menu for an axis.

tion time bail out to the default error handling facility of the underlying Common Lisp system – not always the most helpful place to leave a user.

F. Failure? Fix on a standard.

If after exploring all of the above design principles, no good natural design emerges, it may be time to fix on a standard. Even if the standard appears arbitrary, deciding on one and sticking to it will at least result in a clear and consistent model that can be learned.

Examples of such arbitrary but exceedingly useful standards abound in everyday life: the QWERTY keyboard, stop on red go on green, the arrangement of the digits on a calculator (which for some strange reason are different from their arrangement on a telephone), and so on.

In the above discussion, we have seen at least one completely arbitrary choice in the interface design of Quail, namely associating selection with a left mouse button click over the display and a pop-up menu with a middle-button click. Beyond associating the common gesture (left-button mouse) with the most common interest (selecting or highlighting points), the assignment of actions to gestures is arbitrary.

Largely because of design considerations (hiding complexity and constraining actions to be spatially associated with their target), it was clear early in the design of Quail (ca. 1988) that a three button mouse with two modifier keys (Shift and CTRL) be taken to be part of the standard design. This provides us with nine mouse 'gestures' potentially applicable to any display.

To be at all useful, some rather arbitrary organization had to be applied ot these gestures. We took them to naturally be laid out in a grid as in

	Mouse button		
Modifier	Left	Middle	Right
None	select	change style	change type
Shift	multiple	-	-
CTRL	viewed-object	viewed-object	viewed-object

This shows the present arrangement where the two cells with only a dash are unassigned and the last row does the same thing for all three mouse buttons. The hope is to provide roughly orthogonal behaviours associated with button choice that can be crossed in a meaningful way with modifier key choice. So, for example, no modifier key means that the target of all operations will be the selected graphic while the CTRL key modifier means that the target of all operations will be the viewed object of the graphic. Similarly, left should mean selection, middle changing features of the target, and right making fundamental changes in type to the target. While these behaviours have not yet been worked out for viewed-objects, they are held in reserve for that possibility.

At present, any CTRL mouse-button selection pops the menu of Figure 8 at the current mouse position. This

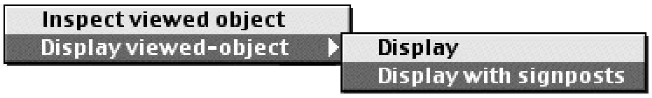

Figure 8: The ctrl middle button menu for a point-symbol.

important menu allows the user to produce an interactive graphic display tailored to the particular viewed-object. For more discussion of these displays and the strategic importance of signposts see Oldford (1997).

5 Concluding remarks

Recall the game with which we began and its ultimate representation as the more familiar tic-tac-toe. This design of the game was successful because it followed good design principles. It coded the constraints into the representation following a natural mapping of constraints to layouts which took advantage of our natural perceptual abilities. It considerably reduced the mental burden on the players and forced a reduction in the errors a player could make. The players were now free to focus on the essence of the game.

Interactive statistical analysis is also hard, considerably harder than the game of tic-tac-toe (whatever the representation). It is desirable to develop and to implement a representation which, like the game, allowed the user more time to concentrate on achieving the objectives of the analysis.

This requires attention to the design and implementation of software models of the statistical concepts which form our common knowledge. Both direct manipulation and programmatic interfaces would benefit substantially from such modelling and the common modelling would allow graphical user interfaces and text-based interfaces to be freely mixed.

To some degree, Quail is designed to provide a programming environment where ideas on modelling statistical concepts and direct manipulation interfaces can be explored. It provides one common foundation for this exploration and has been used with some success by senior undergraduate and graduate students in the development of models and interfaces (e.g. see Oldford, 1997).

So the debate is not one between command language and direct manipulation interfaces, but rather one of appropriate design of the underlying software. The seeming gulf between the two approaches is there because of the lack of a common foundation.

The goal should now be to determine the best models for the selected statistical concepts. Because design is evolutionary this will likely require much debate over the relative merits of different models. The more researchers there are involved in this, the more likely that good models will result.

Prototyping each design will be essential and we will need environments which permit rapid prototyping (Common Lisp and Quail provide one such environment). Nevertheless the model designs will need to be communicated and assessed in an implementation independent way.

The well known principles of design illustrated above provide useful starting points by which designs can be assessed but are not sufficient. Assessment will also need to depend on how well the model represents the statistical concept and so far there is little direct experience in making that assessment – largely because it has not often been a stated goal.

Acknowledgements

Quail has been developed, and continues to be developed by many individuals including R.W. Oldford, C.B. Hurley, D.G. Anglin, M.E. Lewis, and G.W. Bennett. Quail runs on Macintosh and on Windows operating systems via different commercial vendors (see references). Research has been supported by the Natural Science and Engineering Research Council of Canada.

References

Allegro Common Lisp (1997), PC and Unix based Common Lisp from Franz Lisp Inc, Berkeley California.

Desvignes, G.D. and R.W. Oldford (1988). “Graphical Programming” 26 minute video available from the *ASA Statistical Graphics Section's Video Lending Library* presently located at *http://www.research.att.com/ dfs/videolibrary.*

Guthrie, D. (1975). “Dangers in Interactive Statistical Systems” *Proc. of Comp. Sci. and Stats.: 8th Ann. Symp. on the Interface* (ed. J.W. Frane), pp. 8-10, UCLA, USA.

Hurley, C.B. and R.W. Oldford (1991). “A software model for statistical graphics” pp 77-94 of *Computing and Graphics in Statistics* edited by Andreas Buja and Paul A. Tukey, IMA Series on Mathematics and its Applications, Volume 36.

Oldford, R.W. (1997). “ Computational thinking for statisticians: Training by implementing statistical strategy”, pp. 88-97,*Proc. Comp. Sci. & Stat.: Interface*, Houston, TX

Oldford, R.W. (1998). “ The Quail Project: A Current Overview”, *Proc. Comp. Sci. & Stat.: Interface*, Minneapolis, MN.

Oldford, R.W., C.B. Hurley, D.G. Anglin, M.E. Lewis, and G.W. Bennett (1988-1999) *Quail: Quantitative Analysis in Lisp.* A statistical programming environment available free of charge from http://www.stats.uwaterloo.ca/Quail.

Macintosh Common Lisp (1997), Digitool Inc., Cambridge Massachusetts.