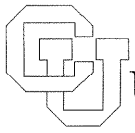
SPREADSHEET-BASED INTERACTIVE GRAPHICS:

FROM PROTOTYPE TO TOOL

Nicholas Wilde and Clayton Lewis

CU-CS-445-89 October 1989



University of Colorado at Boulder

DEPARTMENT OF COMPUTER SCIENCE

* Supported by NSF NYI #CCR-9357740, ONR #N00014-96-1-0720, and a Packard Fellowship in Science and Engineering from the David and Lucile Packard Foundation.

SPREADSHEET-BASED INTERACTIVE GRAPHICS: FROM PROTOTYPE TO TOOL

Nicholas Wilde and Clayton Lewis

CU-CS-445-89 October 1989

Department of Computer Science
Campus Box 430
University of Colorado @ Boulder
Boulder, Colorado 80309-430

This research was done under the auspices of the University of Colorado's Center for Space Construction, NASA Grant NAGW-1388. NoPumpII was created using XVT, a system independent programming toolbox for windowing systems, graciously supplied by API, Inc.

Spreadsheet-based interactive graphics: from prototype to tool

Nicholas Wilde and Clayton Lewis
Dept of Computer Science
University of Colorado, Boulder

Abstract

The NoPumpG prototype (Lewis, 1988, Lewis,in press) suggested that the spreadsheet model of computation could simplify the creation of some types of interactive graphical application when compared with other approaches. We report here experience in developing an enhanced follow-on system, NoPumpII, and describe three applications developed using it. We conclude that (1) the potential advantages of the spreadsheet model are realized in this application experience, (2) revisions to the prototype design have permitted an increase in the complexity and scale of applications, and (3) there remain limitations in the current design which, if redressed, would further enlarge the scope of application. More generally we conclude that alternative computational models are an important area of exploration for HCI research.

Introduction

To what extent are underlying models of computation responsible for the complexity users encounter in dealing with computing systems? While most modern application systems mask the underlying model very effectively, users who wish to create or shape their own applications must still confront and work with a computational model: procedural, functional, or some other. The characteristics of these models may have important impact on the ease with which applications can be expressed, but the usability implications of such models have been little explored in the literature (Lewis 1989).

Following suggestions in Draper (1986) (see also Lewis and Olson,1987) one of us (CL) developed the NoPumpG prototype to explore whether the model of computation underlying the popular spreadsheet paradigm could

be extended to support the creation of interactive graphics applications. This paper reports experience in developing this idea to the point of actual application.

The spreadsheet model

According to many, the spreadsheet has played a major role in the microcomputer revolution. By putting computational power directly in the hands of users, it has allowed many non-programmers to develop their own customized applications.

By providing automatic computation and recomputation, the spreadsheet supplies a computational engine that is fundamentally different to that found in the standard procedural model. Because the spreadsheet is based on the propagation of values, and not the sequential flow of control found in most programming languages, it is a good deal simpler for many small applications. The user does not have to worry about such concepts as flow of control, parameter passing, or recursion, for example.

Another large part of the spreadsheet's appeal is its automatic management of the I/O process. By providing convenient facilities for the input and output of data to the spreadsheet application, the spreadsheet acts as an I/O pumping engine. This frees the user to worry about the specification of their problem, instead of the I/O code. By allowing users to modify data that has already been input, and providing automatic recomputation of the problem at hand, the spreadsheet allows a highly interactive form of "what if" to be played between the user and the computer, encouraging experimentation while providing immediate feedback. By providing direct and immediate access to intermediate values in the computational process (through the use of intermediate cells), the spreadsheet allows rapid and easy debugging of applications, should this experimentation lead to problems.

Applying the spreadsheet model to graphics

The idea behind NoPumpG and NoPumpII is to extend the spreadsheet notion to include interactive graphics. By allowing the user to create graphical primitives, and tying the behavior of those primitives in with a

computational and I/O engine based on the spreadsheet model, NoPumpG and NoPumpII allow the user to quickly and easily build relatively complex applications with a high degree of interactivity in them.

NoPumpG combines interactive graphics with spreadsheet machinery in the following manner. Several graphical primitives are supported in a "draw program" fashion. Each of these primitives, when created by the user, comes with a number of spreadsheet cells. These associated cells both control and report various attributes of the primitive's behavior and appearance. For a typical primitive (say, a line) one might have one or more pairs of X,Y coordinate spreadsheet cells, and a visibility spreadsheet cell. The linkage between cells and primitives is bidirectional. This means that a user can either drag a particular primitive about the screen, in which case the corresponding spreadsheet cells are updated, or input a number or formula into a spreadsheet cell, in which case the corresponding primitive's behavior or appearance changes accordingly. Both values and formulae are supported in NoPumpG's cells. The operators available for formulae allow one to do operations with other spreadsheet cells, resulting in various behaviors being manifest through the graphics. As an example, a line can be forced to stay vertical by causing the X cells of both ends to be equal. If the user moves one endpoint of the line, it only varies in the Y direction, staying vertical and just varying its length.

The resulting applications have a wide variety of uses. Possible applications include the programming of simulations and modeling for physical scientists and engineers, where a high degree of interactivity with the data and model is desirable to allow the exploration of many different relationships between data sets. Other uses include as a teaching aid for subjects such as physics or geometry, where its interactive nature would allow the programming of interactive simulations for the subject matter at hand. NoPumpG or NoPumpII can also serve easily as a highly dynamic and interactive front end to other applications.

Comparison with other approaches

Borning's ThingLab (1979,1981) allows interactive graphics to be developed using objects specified in SmallTalk. Interactions between graphical primitives are specified as constraints on the objects behavior.

As these constraints are fully bidirectional, they represent of a level of complexity and power above NoPumpG's simple spreadsheet propagation mechanism. The resulting system, while more powerful than NoPumpG, is considerably more complex than NoPumpG.

Smith's Alternate Reality Kit (1986,1987) provides a means of modeling physical systems in a graphical framework. ARK is implemented on top of SmallTalk and provides a means of attaching SmallTalk methods to graphical objects by attaching "pushbuttons" to the appropriate object. Objects called "interactors" are available that allow for influences between and interactions among different graphical primitives.

Both Thinglab and Alternate Reality Kit provide mechanisms that are different from, and in some ways more powerful than, the simple spreadsheet propagation mechanism that NoPumpG and NoPumpII are based on. Both rely on SmallTalk to ultimately specify the relationships between objects. While both systems attempt to shield the user as much as possible from this level of programming, the procedural model of computation is still the final arbiter of interactions. Thinglab and ARK both distinguish the "user" of the system from the programmer. The notion is that "someone else" will provide the necessary SmallTalk objects for the user to manipulate in their simulations.

In the NoPump applications, on the other hand, there is no underlying language based model for the user to deal with. The closest thing to a language statement is a spreadsheet cell formula, which may contain a boolean and/or a real valued arithmetic statement. All interactions are specified directly in the top level of the system, and their is no way for the user to go below this. While this results in a somewhat less powerful system than the others, it also results in a less complicated system, from the user-as-programmer point of view.

Fabrik (Ingalls, et al, 1988) is a visual programming application which provides a toolkit of objects that a user may "wire together". Because it relies on dataflow as its primary model of computation, it is a closer to NoPumpG than both ThingLab and ARK. However, the toolkit of objects provided by Fabrik is a good deal richer (and consequently, more complex) than the simple objects provided in NoPumpG or NoPumpII.

From prototype to tool

Experience with the NoPumpG prototype quickly revealed that the resulting system, while interesting, suffers from some limitations that become overwhelming when using NoPumpG for larger scale applications. For instance, some aspects of the graphical primitives are not controlled/reported by spreadsheet cells, making them impossible to modify in the current system. Collision detection, for example, or reporting when a particular graphical object has been selected (so behaviors other than dragging, such as appearing/disappearing when clicked on, could be implemented) is not supported in the NoPumpG prototype. A number of attempts to add different behaviors to primitives by adding different cell types resulted in a system that was more useful, but conceptually more complex than, the original system.

As the number of desired behaviors increased, so did the number of types of cells. As each new cell type was added to each of the already existing primitive types, the resulting matrix quickly became unwieldy. This also meant that the instantiation of a particular primitive brought with it a lot of "cell baggage", regardless of whether the user was interested in the particular attributes those cells controlled. To avoid this, certain cell types were added only to those primitives where it was thought they would be most useful. This had the effect of decreasing the consistency, across primitive types, of the attributes available for inspection and control.

New primitive types were also envisioned that would perhaps have completely different attributes to control. These new types of primitives would add to the increased complexity of the system.

In addition, NoPumpG suffers from a malaise common among spreadsheet applications. Without some levels of abstraction available, as the size of the problem grows, so does the size and complexity of the resulting spreadsheet. It is not uncommon to create spreadsheets with a great deal of inter-connected cells for what seems like a small application.

The primary issues we wished to address in the follow up system, then, were:

Cell explosion

There is a tendency in the prototype for anything but trivial applications to require large numbers of cells. This results from two primary factors: the decision to limit spreadsheet formulae to a single binary operator each, and the tendency to create unneeded cells every time one a new primitive was created.

Control cell explosion

As the desired number of attributes we wished to control increased, so did the number of control cells per primitive. This only served to exacerbate the first problem above.

The need to allow users to follow propagation / control paths easily

When the user must manage a large number of primitives / cells, it often becomes difficult to quickly track which cell is controlling which primitive, and what the propagation path is between cells.

The NoPumpII design

To address these issues, several steps were taken in the design of NoPumpII:

Allow arbitrary formula complexity

The desire here was to limit severely the number of cells which were created as simple holding places for intermediate variables, thus reducing overall the number of cells needed for any application. This is a trade off because part of the spreadsheet's appeal as a model of computation is its automatic management of the I/O process and the ready availability of all intermediate values in the computational process.

Allow arbitrary cell type to be hooked to arbitrary primitive type

Instead of automatically creating certain control cells when a primitive is

created, NoPumpII allows the user to select control cells from a menu and attach them to a given primitive. This has the desired affect of reducing the number of unneeded control cells that get created, and by allowing any type of control cell to be attached to any type of primitive, retains regularity in the primitive/cell relationships.

There are four types of primitives available in NoPumpII: Line primitives, Sketch primitives (small bitmaps), Text primitives (strings), and display primitives (which simply allow the display of a particular cells value without the attendant cell). The are eight types of control cells available: X and Y control cells, Visibility control cells, Pen control cells (turns the attached primitive into a pen that leaves a trail of ink on the screen when moved), Button and Toggle cells (respond to clicks on the attached primitives), and Collision detection cells. In addition, there are two types of cells available that don't control primitives directly, but can be used indirectly through cell formulae: ordinary cells (available for holding intermediate values), and clock cells (which are tied in to a system clock and allow animation to be created easily by making X and Y control cells dependent upon the clock's value)

A side effect of this decision is that an arbitrary number of primitives can be attached to the same cell. This makes for some surprisingly easy and natural behaviors to be prescribed: for instance, it is trivial to create a line that remains vertical no matter where it is dragged on the screen: simply create a line primitive, and attach the same X control cell to both ends of the line.

Use visual clues to tell who is attached to whom, but also give the user active methods of following propagation and control paths.

When a particular cell is selected in NoPumpII, two things may occur if the user elects: 1) both the cell, and all primitives it is attached to that are visible will become highlighted. 2) lines will be drawn explicitly connecting the cell to those primitives it is attached to. The same things may occur in reverse if a user clicks on a given primitive. Both of these have the effect of telling the user about connections already in place.

In addition, a dialogue allows the user to control the visibility/invisibility

of all control cells in a given application. This allows the user to focus in on only those cells that are important at the moment. The same dialogue has tools available to allow chasing up or down a dependency graph, going from cell to its dependents, or to those cells that depend upon it, and making only what the user desires to see visible on the screen.

Application experience with NoPumpII

By incorporating the above refinements into a new application based on the prototype, the application has been scaled up to a level where real applications are a possibility. The following details some applications we have developed using the system:

Curve generation

Figure 1 shows a screen image from the use of NoPumpII to investigate a theoretical argument in tropical ecology. Janzen (1970) presents a model in which the distribution of tree species in tropical forests is tied to patterns of predation that selectively kill seeds or seedlings near parent trees. Janzen presented rough sketch graphs of the relationships which seemed inaccurate in some respects. NoPumpII was used to produce quantitatively accurate graphs based on Janzen's drawings.

The spreadsheet model worked effectively in this application, allowing the user to experiment easily with different formulae, looking for curves that matched Janzen's curves. It was a simple matter to plot the product of two curves, as required in the application. It also proved easy to plot families of related curves, and to rescale and shift plots as needed. While all of these functions would be provided by a mathematical plotting package, with much more sophistication in labeling, varying line styles, and the like, the NoPumpII solution was quick and effective. The application works from first principles of Cartesian geometry and does not require the user to learn conventions of curve specification and formatting that would be imposed by a special-purpose plotting application.

NoPumpII lacks one facility that would have been useful in this application, the ability to draw a curve freehand and capture X-Y pairs

from the drawing. A version of such a facility could be written within NoPumpII, in which the system would automatically construct a piecewise-linear approximation to a freehand curve, but this was not done.

A simple weather model

For many physical applications, the actual calculation of data is a small part of the problem. The rest lies in formatting and presenting the data in a suitable format for easy understanding. Such is the case illustrated in Figure 2. The actual problem is quite simple. It consists of a simple radiation balance model of the the lower kilometer of the earths atmosphere, and attempts to predict the diurnal cycle of temperature that will occur given a initial set of conditions such as latitude, longitude, initial temperature, and sky condition (amount of low, middle, and high cloud). By placing the appropriate formulae for the model in ordinary cells, and creating a sketch with an attached pen that is time dependent on the X position and temperature dependent on the Y position, graphical output from such a model is almost trivial to achieve. To make the model more interactive, three slide controllers were built into the application (using NoPumpII primitives) and their output values used as the values for low, middle and high clouds. Figure 2 illustrates a one day cycle of temperature with no clouds followed by a one day cycle with a 50 % cloud coverage at the middle layer.

Software simulation of a robotics ensemble

A final application is illustrated in Figure 3. Several researchers at the University of Colorado - Boulder are interested in the problem of scheduling several independent robots to perform a given task. A model often used in this application is the robot as finite automaton. By creating an ensemble of automata, and restricting the possible states of the ensemble as a whole, forbidden interactions between the different robots can be represented. We have been charged by this group to create a graphical means of representing these automata, and the interactions between them. Using NoPumpII to create the automata, with several clock cells to drive the individual automata and several button objects to control the clock cells, we have been able to create a software simulation of the particular robotics ensemble in question. The resulting application

can be used to test sequences of elementary moves to determine if the result in a legal configuration for the robotics ensemble, or not. Figure 3 is a software simulation of a ensemble consisting of a cuvette rack (that can be moved up or down), a pipette that can be aimed at a particular cuvette or at a fluid reservoir, and a solution reader / assayer that can be aimed at a particular cuvette and "turned on".

Conclusions

The prototype NoPumpG was successful enough to encourage us to develop a follow up system. By building a new system which capitalized on the experiences gained with the prototype, we were able to create a system that is both more usable and more powerful for real applications.
Our experience with the NoPumpII allows us to conclude the following:

The potential advantages of the spreadsheet model can be realized in an interactive graphics application

The automatic management of I/O in a spreadsheet application can be a boon in many areas where "quick and dirty", or interactive problem solving is useful. By allowing an experimental style of application building, the spreadsheet model allows relatively complex applications to be built in short amounts of time. By managing the automatic computation and recomputation of values by dependency propagation, the spreadsheet model eliminates much of the bothersome "one step at a time" approach that new users may find artificial in procedural or functional models, and allows "what if" capabilities to be built into applications without additional difficulty.

Revisions to the prototype design permit an increase in the complexity and scale of applications

By allowing the user to specify what information they wish to see controlled (and not automatically providing it for them), NoPumpII still retains many of the virtues of the spreadsheet model without causing the user to be overwhelmed in detail. The tendency to get lost in a

proliferation of cells has been curbed to a large extent.

Redressing limitations in the current design could further enlarge the scope of application.

Many applications require the same assemblage of primitives and cells to be created more than once (i.e. the slide controllers in the weather model or the button objects in the robotics ensemble). We would like to allow the creation of user macros for assemblages of primitives and cells.

In addition discussions with users in the case of the robotics ensemble have pointed out some additional capabilities we would like to include: the ability to read in a sequence of robotic moves as a file script for running the software model over. A general capability for allowing models to be run under control of a previously created script could be useful for much more than just the robotics ensemble application. Combining the capability of being able to replay simulations with the "what if" aspects of spreadsheets would provide a powerful experimental capability for many different applications. For instance, the weather model could be run and rerun with different parameters over several days worth of simulated cloud cover progressions.

A more general lesson we draw from our experience with NoPumpG and NoPumpII is that alternatives to the familiar procedural model of computation are worth further serious exploration. The spreadsheet model offers a new look at computation that has considerable generality, certainly beyond that of the very successful commercial spreadsheet systems. Analogously, other models little explored in HCI research, such as functional programming or equational programming (Sethi, 1989, O'Donnell, 1985) may permit us to deliver more power at less conceptual cost for important areas of application.

Acknowledgements.

This research was done under the auspices of the University of Colorado's Center for Space Construction, NASA Grant NAGW-1388. NoPumpII was created using XVT (the Extensible Virtual Toolkit), a system independent programming toolbox for windowing systems, graciously supplied by API,

Inc. Brent Reeves created the parser that NoPumpII uses for expression evaluation. The NoPump team from CL's User Interface course (Dennis Colarelli, Brent Reeves, Chase Turner, and Diane Stockton) contributed to the initial conception and design of NoPumpII. John Blanco and other members of the Construction Operations Branch of the Center for Space Construction were instrumental in evaluating the NoPumpG prototype.

References

Borning, A., 1979, "ThingLab - A constraint oriented simulation laboratory", *Xerox PARC Technical Report SSL-79-3*, Xerox Palo Alto Research Center.

Borning, A., 1981, "The programming language aspects of ThingLab, a constraint oriented simulation laboratory", *ACM Transactions on Programming Languages and Systems*, Vol. 3, p. 353.

Draper, S.W., 1986, "Display managers as the basis for user-machine communication", In D.A. Norman and S.W. Draper (Eds.) *User Centered System Design: New perspectives on human-computer interaction* , Erlbaum, Hillsdale, NJ, p 339.

Ingalls, D., S. Wallace, Y. Chow, F. Ludolph, and K. Doyle, 1988, "Fabrik, A visual programming environment", *Proc OOPSLA '88*, ACM, New York, NY, p. 176.

Janzen, D.H., 1970, "Herbivores and the number of tree species in tropical forests", *The American Naturalist*, Vol 104, No 940, pp 501-528.

Lewis, C.H., 1988, "Making interactive graphics accessible", Panel Discussion in *Proc. CHI'88 Human Factors in Computing Systems*, ACM, New York, NY.

Lewis, C.H., 1989, "New approaches to programming", *Memorias del 14 Symposium Internacional de Sistemas Computacionales*, Instituto Tecnologico y de Estudios Superiores de Monterrey. Monterrey, N.L., Mexico.

Lewis, C.H., in press, "NoPumpG: Creating Interactive Graphics with Spreadsheet Machinery", In Glinert, E.P. (Ed), *Visual Programming Environments,* IEEE Computer Society Press, Los Angeles, CA (to appear early 1990)

Lewis, C.H. and G. M. Olson, 1987, "Can principles of cognition lower the barriers to programming ?", In G. Olson (Ed.) *Empirical Studies of*

Programmers 2. Ablex, Norwood, NJ.

O' Donnell, M.J., 1985, *Equational logic as a programming language*, MIT Press, Cambridge, MA.

Sethi, R., 1989, *Programming Languages,* Addison-Wesley, Reading, MA.

Smith, R.B., 1986, "The alternate reality kit", *Proc. 1986 IEEE Workshop on Visual Languages*, IEEE, Washington, DC, p. 99.

Smith, R.B., 1987, "Experiences with the alternate reality kit: An example of the tension between literalism and magic", *IEEE Computer Graphics and Applications* , pp 42-50.

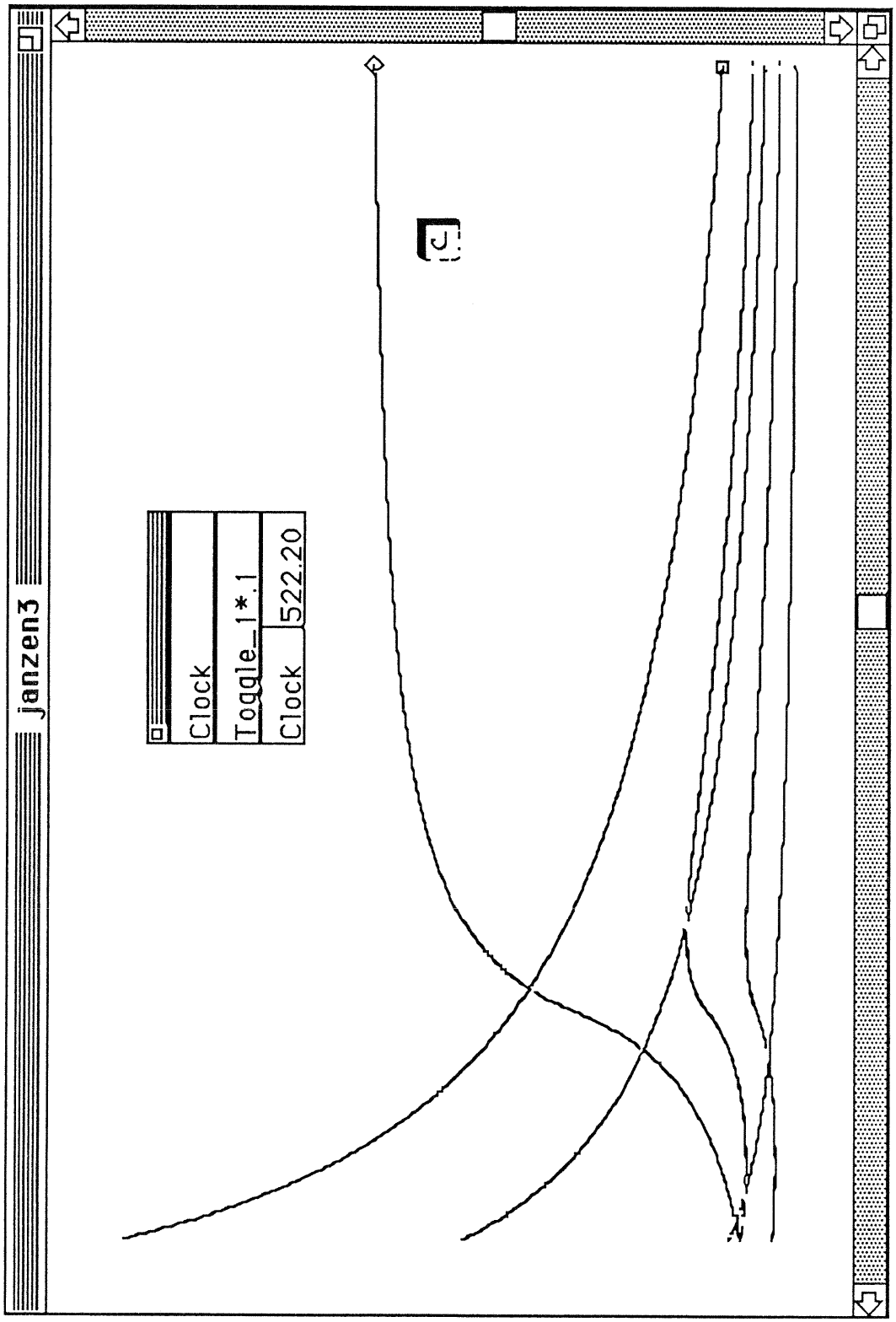


Figure 1. A series of curves, drawn to illustrate an argument in tropical ecology, generated using NoPumpII

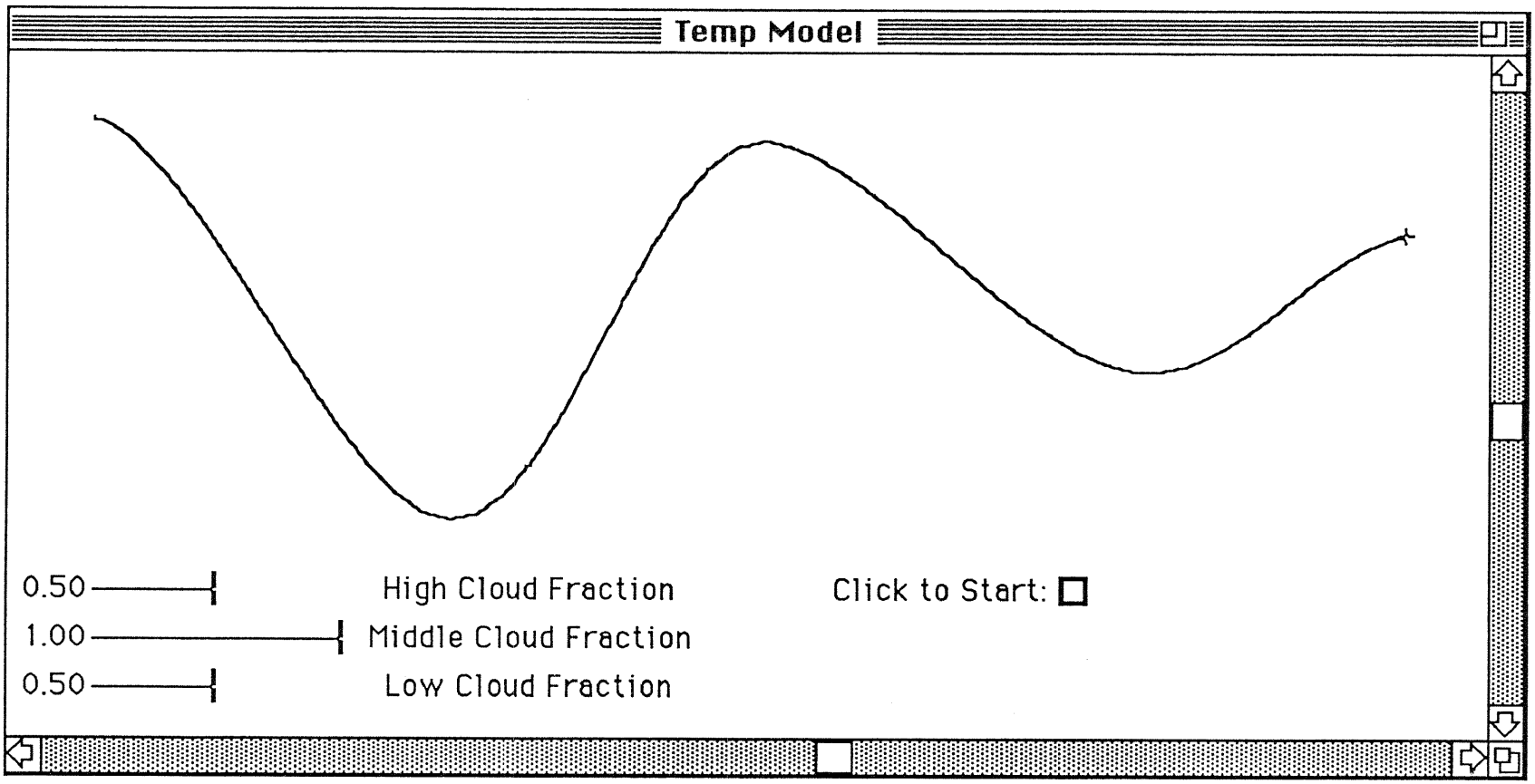


Figure 2. An interactive weather model

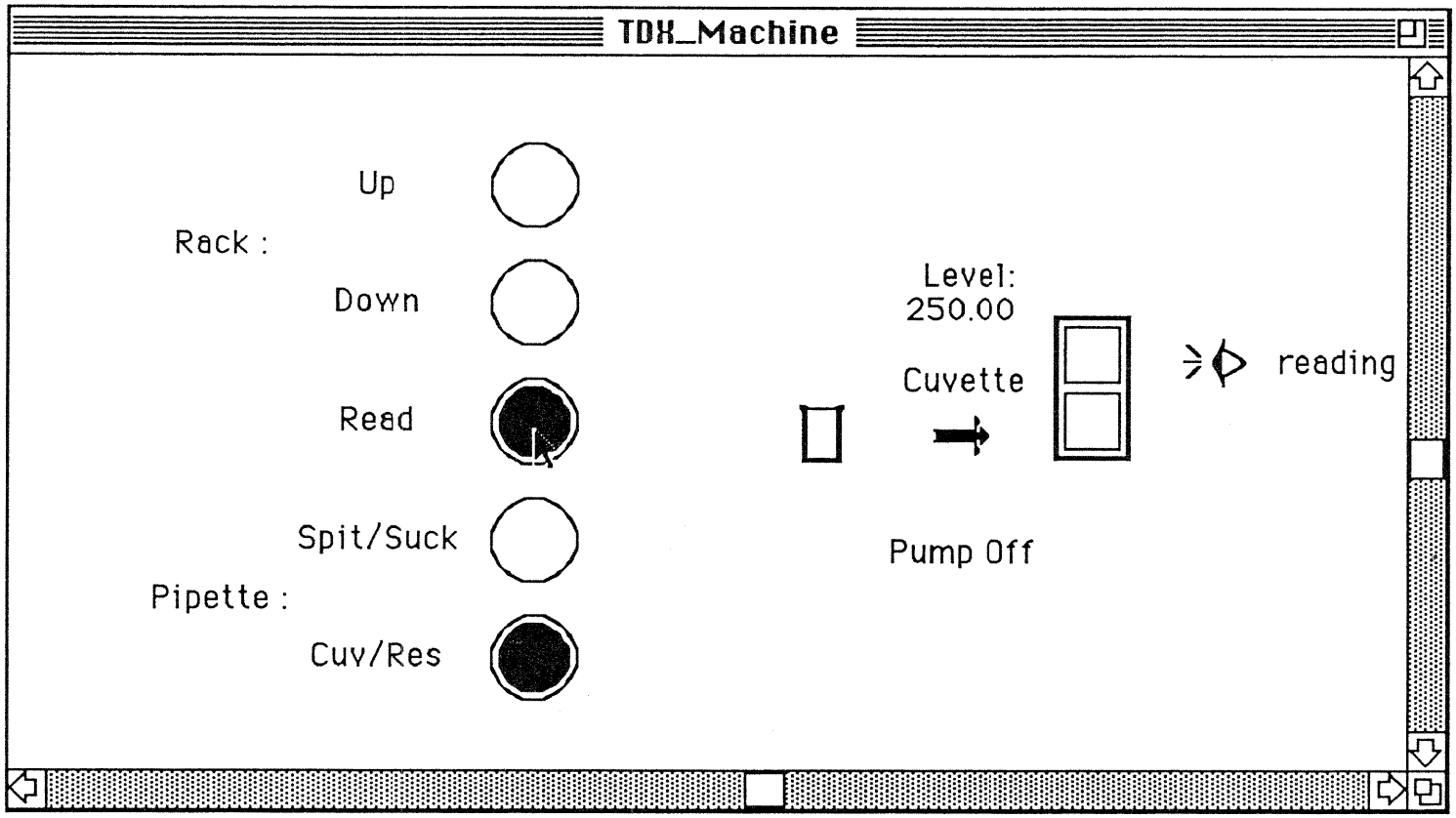


Figure 3. A robotics ensemble simulated in software.