

PERSONAL DYNAMIC MEDIA

BY THE LEARNING RESEARCH GROUP



XEROX

PALO ALTO RESEARCH CENTER

PERSONAL DYNAMIC MEDIA
XEROX PALO ALTO RESEARCH CENTER
LEARNING RESEARCH GROUP

© Copyright by Xerox Corporation March, 1976

Table of Contents

I.	Introduction	Page 3
II.	Background	4
	<i>Humans and Media</i>	4
	<i>A Dynamic Medium for Creative Thought: The Dynabook</i>	4
	<i>Design Background</i>	8
	<i>Our Approach</i>	10
III.	An Interim Dynabook	12
	<i>Remembering, Seeing and Hearing</i>	12
	<i>Different Fonts for Different Effects</i>	12
	<i>Editing</i>	15
	<i>Filing</i>	17
	<i>Drawing/Painting</i>	22
	<i>Turtles</i>	23
	<i>Animation and Music</i>	27
	<i>Simulation</i>	27
IV.	Smalltalk: A Communications Medium for Children of All Ages	42
V.	Smalltalk, Dynabooks, and Kids	49
	<i>Example Smalltalk Programs</i>	49
	<i>Outline of Planned Projects</i>	52
1.	Teaching Smalltalk	54
	Tutorial Dialogues with Smalltalk	
	Composing Pictures from Geometric Shapes	
2.	Plans for a Readiness Program	60
	Thinking Games for Primary-aged Children	
	Button Boxes and Turtles	
3.	Experiments with Schools	68
	English Classes for Junior High and High Schools	
	High Schools	
	Spaceship Simulation Project	
VI.	Summary	71
VII.	Acknowledgements	72
VIII.	References	73



PERSONAL DYNAMIC MEDIA

Learning Research
Xerox Palo Alto Research Center

I. Introduction

The Xerox Learning Research group (LRG) is concerned with all aspects of the communication and manipulation of knowledge. We design, build, and use dynamic media which can be made accessible to human beings of all ages. Several years ago, we crystallized our dreams into a design idea for a personal dynamic medium the size of a notebook (the *Dynabook*) which can be owned by everyone and has the power to handle virtually all of its owner's information-related needs. Towards this goal we have designed and built a communications system: the Smalltalk language, implemented on small computers we refer to as interim Dynabooks. We are exploring the use of this system for programming and problem solving; as an interactive memory for the storage and manipulation of data; as a text editor; and as a medium for expression through drawing, painting, animating pictures, and composing and generating music.

We have used our experimental experiences with our interim Dynabooks to guide the design of learning activities and examples for children in different age groups: preschool, primary, intermediate, and high school. Since children differ as to interest and intellectual development, each activity explores the potential of this new medium for these various groups. Our work with adults includes those in fields other than the computer sciences, especially those who are professionally involved with handling knowledge, such as secretaries and librarians.

We offer this report as a perspective on our goals and activities during the past years. In it, we explain the Dynabook idea, describe a variety of systems we have already written in the Smalltalk language, and outline some of our plans for a research program to be carried out in a resource center located near schools and homes.



II. Background

Humans and Media

"Devices" which variously store, retrieve, or manipulate information in the form of messages embedded in a medium have been in existence for thousands of years. People use them to communicate ideas and esthetic feelings both to others and back to themselves. Although thinking goes on in one's head, external media serve to materialize thoughts and, through feedback, to augment the actual paths the thinking follows. Methods discovered in one medium provide metaphors which contribute new ways to think about notions in other media.

For most of recorded history the interactions of humans with their media have been primarily nonconversational and passive in the sense that marks on paper, paint on walls, even "motion" pictures and television, do not change in response to the viewer's wishes. A mathematical formulation--which may symbolize the essence of an entire universe--once put down on paper, remains static and requires the reader to expand its possibilities.

Every message is, in one sense or another, a *simulation* of some idea. It may be representational or abstract, isolated or in context, static or dynamic. The essence of a medium is very much dependent on the way messages are embedded, changed, and viewed. Although digital computers were originally designed to do arithmetic computation, the ability to simulate the details of any descriptive model means that the computer, viewed as a medium itself, can be *all other media* if the embedding and viewing methods are sufficiently well provided. Moreover, this new "metamedium" is *active*--it can respond to queries and experiments--so that the messages may involve the learner in a two-way conversation. This property has never been available before except through the medium of an individual teacher. We think the implications are vast and compelling.

A Dynamic Medium for Creative Thought: The Dynabook

Imagine having your own self-contained knowledge manipulator in a portable package

the size and shape of an ordinary notebook. Suppose it had enough power to outrace your senses of sight and hearing, enough capacity to store for later retrieval thousands of page-equivalents of reference material, poems, letters, recipes, records, drawings, animations, musical scores, waveforms, dynamic simulations, and anything else you would like to remember and change.

We envision a device as small and portable as possible which could both take in and give out information in quantities approaching that of human sensory systems. Visual output should be, at the least, of higher quality than what can be obtained from newsprint. Audio output should adhere to similar high fidelity standards.

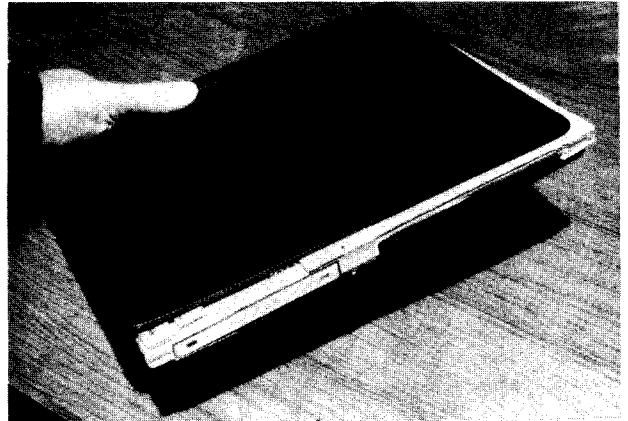
There should be no discernible pause between cause and effect. (One of the metaphors we used when designing such a system was that of a musical instrument, such as a flute, which is owned by its user and responds instantly and consistently to its owner's wishes. Imagine the absurdity of a one second delay between blowing a note and hearing it!)

These "civilized" desires for flexibility, resolution, and response lead to the conclusion that a user of a dynamic personal medium needs several hundred times as much power as the average adult now typically enjoys from timeshared computing. This means that we should either build a new resource several hundred times the capacity of current machines and share it (very difficult and expensive), or we should investigate the possibility of giving each person his own powerful machine.

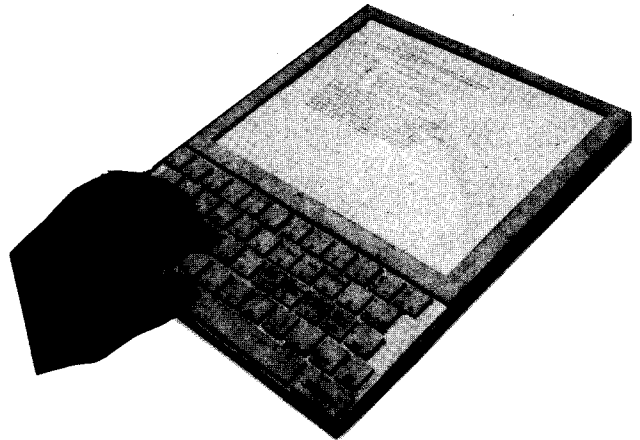
If such a machine is designed in a way that *any* owner can mold and channel its power to his own needs, then a new kind of medium will have been created: a metamedium, whose content is a wide range of already-existing and not-yet-invented media.

An architect may wish to simulate three-dimensional space in order to peruse and edit his current designs which can be conveniently stored and cross-referenced.

A doctor can have all of his patients on file, his business records, a drug reaction system, and so on, all of which can travel with him wherever he goes.



cardboard mockup of a Dynabook



An animator can have a tool which will show him, and allow him to edit, his animation as he is creating it in order to eliminate guesswork. Instead of laboriously having to redraw frame after frame and produce "in-betweens", he would communicate his desires for movement by showing and telling the system what to do.

A composer can hear his composition while it is in progress, particularly if it is more complex than he is able to play. He could also bypass the incredibly tedious chore of redoing the score and producing the parts by hand.

Learning to play music can be aided by being able to capture and hear one's own attempts and compare them against expert renditions. The ability to express music in visual terms which can be filed and played means that the acts of composition and self-evaluation can be learned without having to wait for technical skill in playing.

The experience of playing the great musical instruments of the past, such as baroque organs and harpsichords actually played by Bach, pianos played by Beethoven, and so on, has been only open to a few people. It could be possible to enjoy this experience in one's home through the power of high-resolution simulation.

Home records, accounts, budgets, recipes, reminders, and so forth, can be easily captured and manipulated.

Those in business can have an active briefcase which can travel with them and can contain a working simulation of their company, the last several weeks of correspondence in a structured cross-indexed form, a way to instantly calculate profiles for their futures and help make decisions.

For educators, the Dynabook can be a new world limited only by their imagination and ingenuity. They can use it to show complex historical inter-relationships in ways not possible with static linear books. Mathematics becomes a living language in which children can cause exciting things to happen. Laboratory experiments and simulations too expensive or difficult to prepare can easily be demonstrated. The production of stylish prose and poetry can be greatly aided by being able to easily edit, file and "debug" one's own compositions. Those who believe in a free, open approach to creativity and learning will find a wealth of cross-connection and philosophy of world-view

which promotes synergistic thinking. Those who feel that a more structured approach is appropriate, including those interested in computer-assisted instruction, can easily implement their own version of the Socratic dialogue using dynamic simulation and half-toned graphic animation.

For children, this dynamic notebook, or *Dynabook*, can be an environment in which the natural activities are creative thinking and planning; visualization of effects and their causes; the assumption of varying perspectives towards humans, knowledge, and culture; and the enhancement of their personal style. It can also be a magic gateway to the rich world of the already-known, offering a compelling way to dynamically peruse the great discoveries of the past and present in ways that are just not possible with static media such as books and photos.

These are just a few ways in which we envision using a Dynabook. But, if everyone can have one, is it possible to make the Dynabook generally useful, or will it collapse under the weight of trying to be too many different tools for too many different people? The total range of possible users is so great that any attempt to specifically anticipate their needs in the design of the Dynabook would end in a disastrous feature-laden hodgepodge which would not be really suitable for anyone. We have taken an entirely different approach to this problem, one which involves the notion of providing many degrees of freedom and a way for any user to communicate his or her own wishes for a specific ability.

Some mass items, such as cars and television sets, attempt to anticipate and provide for a variety of applications in a fairly inflexible way; those who wish to do something different will have to put in considerable effort. Other items, such as paper and clay, offer many dimensions of possibility and high resolution; these can be used in an unanticipated way by many, though *tools* need to be made or obtained to stir some of the medium's possibilities while constraining others.

We would like the Dynabook to have the flexibility and generality of this second kind of item, combined with tools which have the power of the first kind. Thus a great deal of effort has been put into providing both endless possibilities and

*easy tool-making through a new medium
for communication called Smalltalk.*

When a house is being designed, the last thing the designer wishes to worry about is the process by which bricks are constructed--unless he really needs a new kind of brick. In general, people who *do things* like to have available a standard set of "building blocks" whose properties they understand, with an "escape" to new tool building when needed.

Our design strategy, then, divides the problem. The burden of system design and specification is transferred to the user (who will generally not be a computer scientist). This approach will only work if we do a very careful and comprehensive job of providing a general medium of communication which will allow ordinary users to casually and easily describe their desires for a specific tool. We must also provide enough already-written general tools so that a user need not start from scratch for most things he or she may wish to do.

Design Background

The first attempt at designing this metamedium (then called the FLEX machine) occurred in 1967-69 [16,17,18]. Much of the hardware and software was successful from the standpoint of computer science state-of-the-art research but lacked sufficient expressive power to be useful to an ordinary user. At that time we became aware of Papert and Feurzeig's pioneering work having to do with teaching kids how to think by giving them an environment in which thinking is fun and rewarding [10,24,25]. They chose a time-shared computer and invented a simple though powerful language called Logo. With Logo, the children (ranging in age from 8-12 years) could write programs to control a number of exciting activities: a robot turtle which can draw, a CRT version of the turtle, and a simple music generator.

The Logo work radiates a compelling excitement when viewed from a number of different perspectives.

First, the children really can program the turtle and the music box to do serious things. The programs use symbols to stand for objects, contain loops and recursions, require a fair

amount of visualization of alternate strategies before a tactic is chosen, and involve interactive discovery and removal of "bugs" in their ideas. As Papert points out, the children are performing real mathematical acts of a kind, scope and a level not achieved by many college graduates.

Second, the kids love it! The interactive nature of the dialogue, the fact that *they* are in control, the feeling that they are doing *real* things rather than playing with toys or working out "school" problems, the pictorial and auditory nature of their results, all contribute a tremendous sense of accomplishment to their experience. Their attention spans are measured in hours rather than minutes.

A number of Feurzeig and Papert's results were particularly interesting to those of us who had designed the FLEX machine. Aside from the potential future for education implied by getting kids to program, we realized that many of the problems involving the design of a metamedium for creative thought, particularly those having to do with expressive communication, were brought strongly into focus when children down to the age of six were seriously considered as users.

Another interesting nugget was that children really needed as much or more computing power than adults were willing to settle for when using a time-sharing system. The best that time-sharing has to offer is slow control of crude wire-frame green-tinted graphics and square-wave musical tones. The kids, on the other hand are used to finger-paints, water colors, color television, real musical instruments, and records. If the "medium is the message" then the message of low bandwidth time-sharing is "blah".

We felt then that the next time we tried to design a personal metamedium it should be done with children strongly in mind. We decided to expand our horizons to include studies into the nature of the learning and creative processes, visual and auditory perception, how to teach thinking, and how to show children the challenges and excitement of *doing* art and science.

Our Approach

First, we decided to admit that the design of a truly useful dynamic medium was a hard but extremely worthwhile problem which would require many years and a number of complete interim hardware/software systems to be designed, built and tested. Our basic approach is:

1. Conceptualize a "Holy Grail" version of what the eventual Dynabook should be like in the future. This image will provide a rallying point and goal which can be referenced while the sometimes grubby spadework of producing intermediate systems is going on.
2. Do the research in human factors, psychology of perception, physics, and language design which is prerequisite to any serious attempt at an interim system.
3. Design an interim version of the Dynabook and build a considerable number of them.
4. Make the medium of communication as simple and powerful as possible.
 - a. It should be simpler and more powerful than (say) Logo.
 - b. It should be better than the best state-of-the-art "grown-up" programming language for serious systems design.
 - c. It should be as "neutral" as possible to all conceivable simulations.
5. Explore the usefulness of such a system with a large number of short range projects involving more than 100 users, ages 4 to 60, from varying backgrounds and with different needs and goals.
 - a. Develop all manner of simulated media including typography, music, animation, physical simulations, and file systems.
 - b. Develop methods and strategies which aid teaching and learning the system.
 - . Invent projects and produce curriculum materials
 - . Capture all transactions of users with the system
 - . Video-tape much of the activity
 - . Experiment with peer-group teaching (for example, 13-year olds teaching 12-year olds)
6. Re-extend the system in the light of the two-year study and start to think about the next interim system.

7. Set up a community resource center containing several interim Dynabook systems for both open- and closed-shop use near school and playground traffic patterns.

a. Start a series of longitudinal studies, primarily with children, in cooperation with local schools.

b. Study the roles of real-time feedback of sensory impressions and hands-on editing and debugging, cross-filing, and programming in art, music, science, and writing.

. Investigate the use of this medium as a focus for creativity and play

. Study its use with and without teachers for more formal learning

. Continue with project and curriculum development

. Continue with peer-group tutoring

. Get teachers and parents involved in this new way to look at things

. Find ways to ascertain the impact of this new medium on both the quantity and quality of the child's model of the universe.

c. Provide an *open-shop* in the resource center for a large number of hours per day, offering free machine time for casual computing to the community at large, including children, visiting artists, musicians, and educators.

8. Develop the next interim version of the Dynabook, and so on.

We have completed, with considerable help from others at Xerox PARC, parts 1 through 6. The interim Dynabooks now being used are capable of producing high quality real-time video and music synthesis, with local file capacities of several thousand page-equivalents of cross-indexed user-defined simulations. The extensible communications system, called Smalltalk, has been used for building systems by children and adults: professionals, amateurs, and tyros. In order to give a picture of what is possible on this first version of a metamedium, a number of interesting systems which have been produced are described in the following sections.

III. An Interim Dynabook

Although it is not easy to convey the flavor of the extreme freedom, ease, and flexibility of this dynamic medium through a static two-dimensional paper, we will attempt to show some of the kinds of things that can be done with a Dynabook. Then a number of interesting systems developed by various users will be briefly illustrated. Henceforth, we will use the word "Dynabook" to refer to the interim system already designed and built by us. All photographs of computer output in this report are taken from the display screen of this system.

Remembering, Seeing and Hearing

The Dynabook can be used as an interactive memory or file cabinet. The owner's context can be entered through a keyboard and active editor, retained and modified indefinitely, and displayed on demand in a font of publishing quality. Each removable file memory can hold the equivalent of 1500 pages of text.

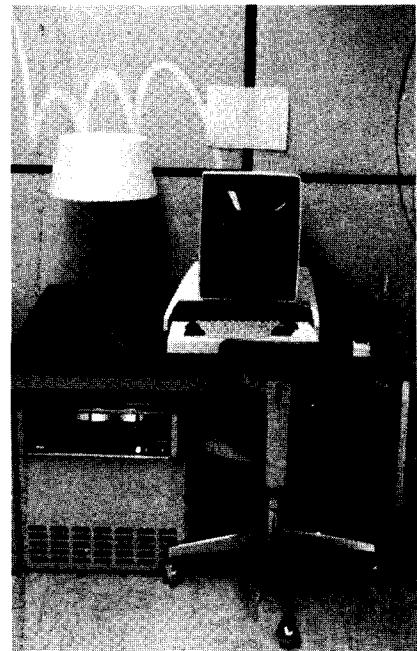
Drawing and painting can also be done using a hand-held pointing device and an iconic editor which allows easy modification of pictures. A picture is thus a manipulable object (just as are characters, words and sentences) and can be animated dynamically by the Dynabook's owner.

Music can be composed, edited, and played either from a score or via a variety of input accessories.

A book can be read through the Dynabook: the memory can be inserted as shown to the right. It need not be treated as a simulated paper book since this is a new medium with new properties. A dynamic search may be made for a particular context. The non-sequential nature of the file medium and dynamic manipulation allow a story to have many accessible points of view; Durrell's *Alexandria Quartet*, for instance, could be *one book* in which the reader may pursue many paths through the narrative.

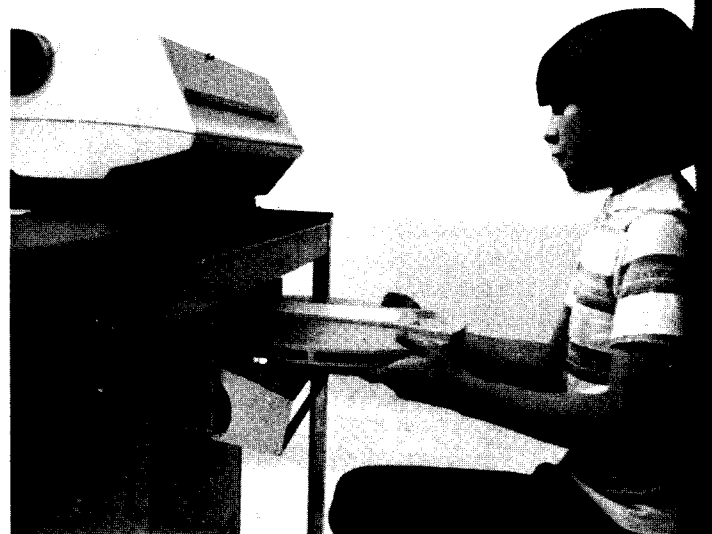
Different Fonts for Different Effects

One of the goals of the Dynabook's design is *not* to be *worse* than paper in any important way. Computer displays of the past have been superior in matters of dynamic writing and



processor, display screen
and keyboard

inserting the file memory



erasure, but have failed in contrast, resolution, or ease of viewing. There is more to the problem than just the display of text in a high quality font. Different fonts create different moods and cast an aura that influences the subjective style of both writing and reading. The Dynabook is supplied with a number of fonts which are contained on the file storage.

CHAPTER I

In Which

We Are Introduced to Winnie-the-Pooh and Some Bees, and the Stories Begin

Here is Edward Bear, coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it. And then he feels that perhaps there isn't. Anyhow, here he is at the bottom, and ready to be introduced to you, Winnie-the-Pooh. When I first heard his name, I said, just as you are going to say, "But I thought he was a boy?" "So did I," said Christopher Robin. "Then you can't call him Winnie?"

Sans Serif

CHAPTER I

In Which

We Are Introduced to Winnie-the-Pooh and Some Bees, and the Stories Begin

Here is Edward Bear, coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it. And then he feels that perhaps there isn't. Anyhow, here he is at the bottom, and ready to be introduced to you, Winnie-the-Pooh. When I first heard his name, I said, just as you are going to say, "But I thought he was a boy?" "So did I," said Christopher Robin. "Then you can't call him Winnie?" "I don't." "But you said-----" "He's Winnie-the-Pooh. Don't you know what 'that' means?"

Serif

CHAPTER I

In Which

We Are Introduced to Winnie-the-Pooh and Some Bees, and the Stories Begin

Here is Edward Bear, coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it. And then he feels that perhaps there isn't. Anyhow, here he is at the bottom, and ready to be introduced to you, Winnie-the-Pooh. When I first heard his name, I said, just as you are going to say, "But I thought he was a boy?"

Alan's Handwriting

wun de when poe beer had nuthig eis too doo, hee ghaut hee woad doo sumphig, see hee went round too piglet's hous too see what piglet wuz dooin. it wuz still sneeg as hee stumpt over the whiet forest track, and hee ekspected to fiend piglet wairning his toes in frunt uv his fier, but too his surprise hee saw that the frunt deer wuz open, and the moer hee lookt insied the moer piglet wuzn't maer.

"hee's out," sed poe sadly. "that's what it is. hee's not in. ie shall hav too gee a fast thinkig wauk bie mie self. bother!"

but first hee ghaut that hee woad nock very loudly just too mack kwiet shur...and whiel hee wated fer piglet not too anser, hee jumpt up and down too keep worm, and a hum cam suddenly into his hed, which seemd too him a good hum, such as is hummd hapfully too others.

ITA

भाग १
उत्सवा

विनो-पू तथा अरि मरिहोर हागिस्ताई
बोलादेन छन, र कथा सुरु भैलछ।

यहां चाहि क्रोटोकर रामको पछारि, डक, डक, डक, टाउकाया आउंदो बालू बहादुर माथिबाट आउंदे रहिछ। अरु भन्दाबाट आउने रितो उत्साई बाहा नभएता पनि, कहिले कहि उत्साई लाग्छ कि यदि यो चाहि डक-डक-डक बलम हुन्छ र उ सोचि सक्छ भने अरु आउने रितो हुने छ। अरि लाग्छ कि छन! तापनि, यहाँ तस आइपुगेर विनो-पू बोलादेन छ।

उस्को नाम पहिल्ला पटक सुनेर तिमिस्ताई भन्न लाग्ने जन्ते भैले भने, "तर मलाई लाग्छो कि उ केता हो रे।"

"हो--मलाई पनि त्यसो लाग्छो," क्रोटोकर रामले भन्थो।

"त्यस कारण उ केता भए उस्को नाम 'विनो' हुन सक्दैन। होइन त ?"

"हो--"

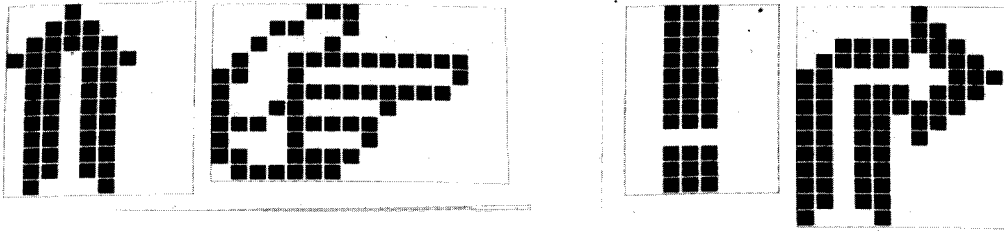
Sanskrit

विनो-पू तथा अरि मरिहोर हागिस्ताई बोलादेन छन, र कथा सुरु भैलछ।

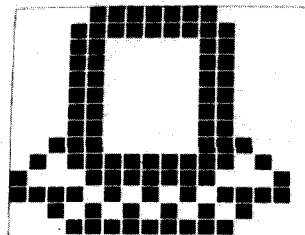
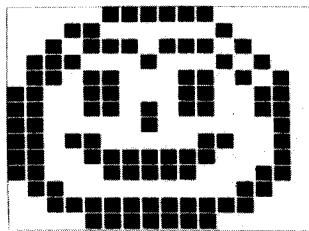
यहां चाहि क्रोटोकर रामको पछारि, डक, डक, डक, टाउकाया आउंदो बालू बहादुर माथिबाट आउंदे रहिछ। अरु भन्दाबाट आउने रितो उत्साई बाहा नभएता पनि, कहिले कहि उत्साई लाग्छ कि यदि यो चाहि डक-डक-डक बलम हुन्छ र उ सोचि सक्छ भने अरु आउने रितो हुने छ। अरि लाग्छ कि छन! तापनि, यहाँ तस आइपुगेर विनो-पू बोलादेन छ।

उस्को नाम पहिल्ला पटक सुनेर तिमिस्ताई भन्न लाग्ने जन्ते भैले भने, "तर मलाई लाग्छो कि उ केता हो रे।"

Fingerspelling



a finished example for a turtle font



```

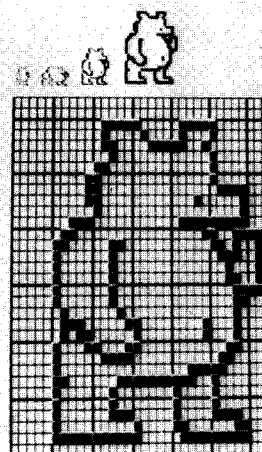
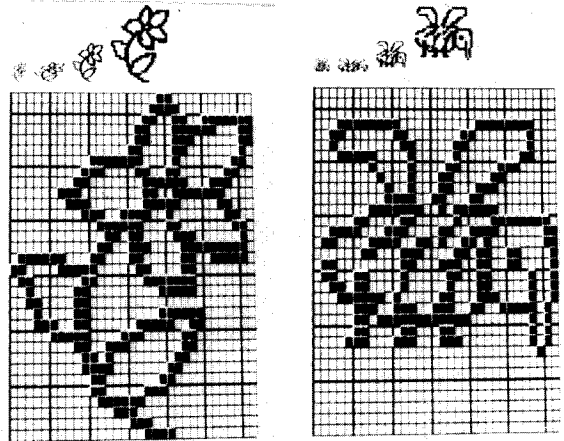
Q @ ← turtle !
Q @ # 100 !
Q @ # 50 !
Q
    
```

telling a story with pictures

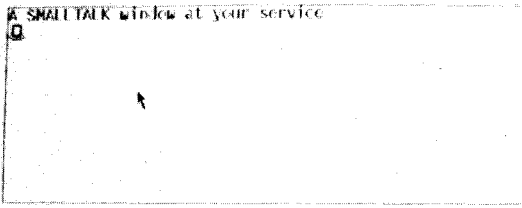
The malleability of this approach is illustrated to the right: this owner has decided to embellish some favorite nouns with their iconic referent. A teacher of early reading might like to help children form correspondences between words and objects by having the visual representations interchange when pointed to by the child.

Editing

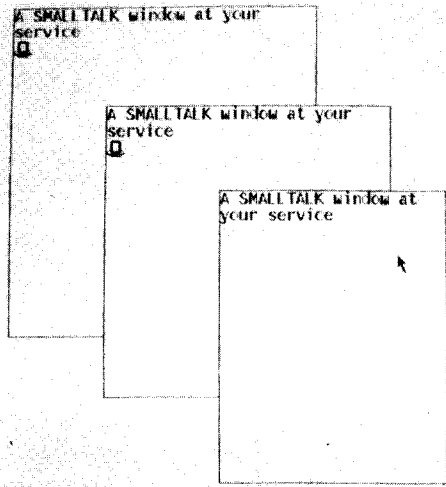
Every description or object in the Dynabook--such as the character fonts described above--can be displayed and edited. Text, both sequential and structured (as in file records), can easily be manipulated by combining pointing and a simple "menu" for commands, thus allowing deletion, transposition and structuring. Multiple windows allow a document (composed of text, pictures, musical notation, and so on) to be created and viewed simultaneously at several levels of refinement. Editing operations on other viewable objects (pictures, fonts) are handled in analogous ways.



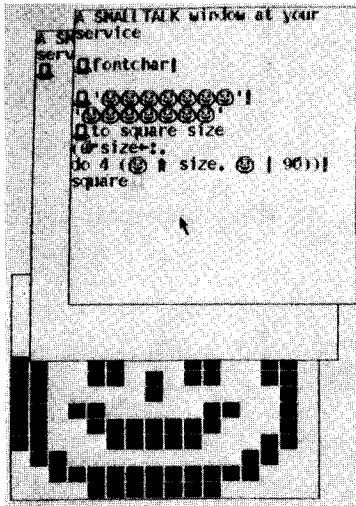
There once was a who got lost ?
 He did not know whether to go or .
 He asked a he saw sitting on a .
 The promptly stung the on the nose .
 Ouch, said the . The said 'I am sorry--I forgot myself. Close one and took your , said the .
 'The way to go, or , is the way you will see.' So the closed one .



Members of the class window are located anywhere on the screen.

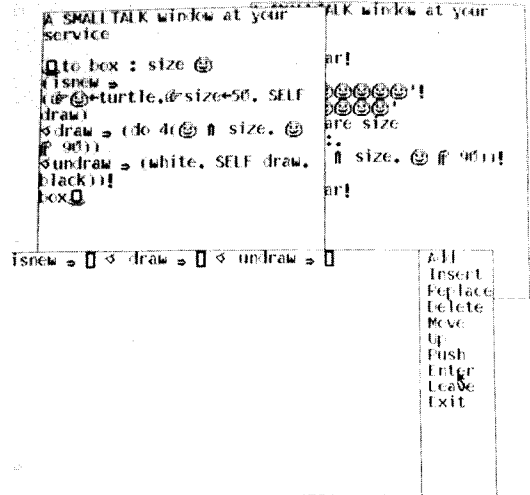
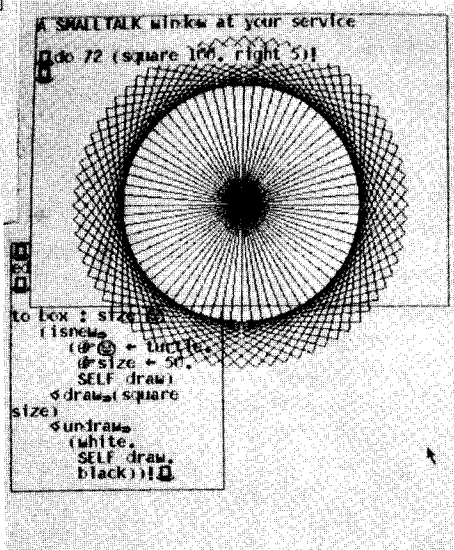


They are different sizes and shapes...

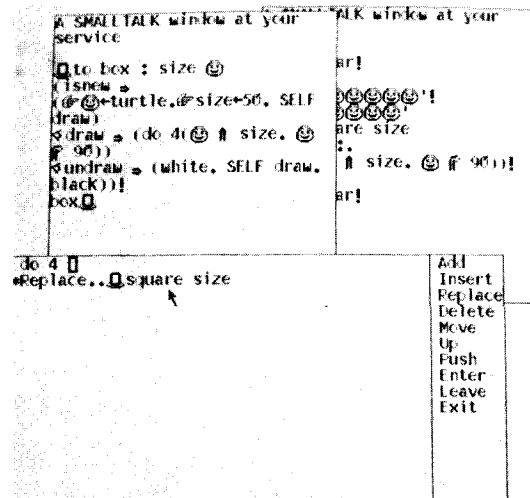


and contain different information.

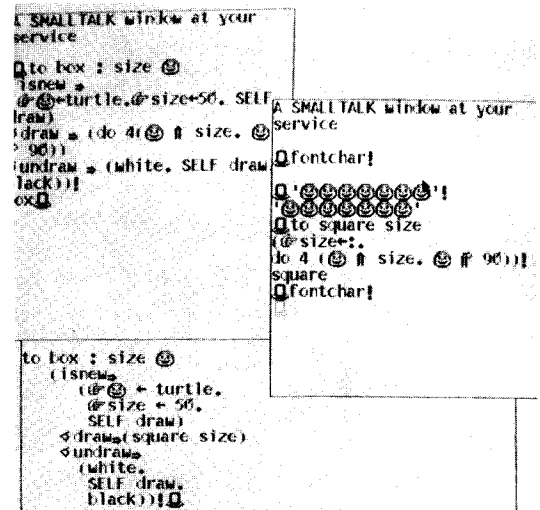
Running the program square



Editing Smalltalk class box to replace the action taken when a box sees the draw message



One window shows the definition before editing, another the edited version.



Filing

The multiple-window display capability of Smalltalk has inspired the notion of a dynamic *document*. A document is a collection of objects that have a sensory display and have something to do with each other; it is a way to store and retrieve related information. Each subpart of the document, or *frame* (text, picture, font, music), has its own editor which is automatically invoked when pointed at by the stylus. These frames may be related sequentially (as with ordinary paper usage) or *inverted* with respect to properties (as in a cross-indexed file system). *Sets* which can automatically map their contents to secondary storage with the ability to form unions, negations, and intersections are part of this system, as is a new "modeless" text editor with automatic right justification.

The current version of the system is able to automatically crossfile several thousand multifield records (with formats chosen by the user), which include ordinary textual documents indexed by content, the Smalltalk system, personal files, books, papers, and so on. The system shows that a personalized retrieval system, integrated with a comprehensive editor and high level language, all running on a personal computer, is a highly attractive package.

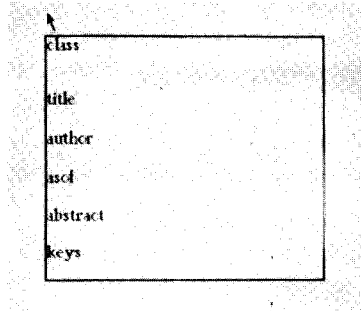
This experimental document system will have a number of important uses. First, it will serve as a test-bed for new ideas in information storage, retrieval, handling, and viewing--its design represents an amalgamation of ideas developed by us and others at PARC. Since it is written in Smalltalk, it is easy to modify.

Second, its brevity and simplicity will allow non-computer-professionals to add features. A new search feature has already been added to the paragraph text editor program by a secretary at PARC .

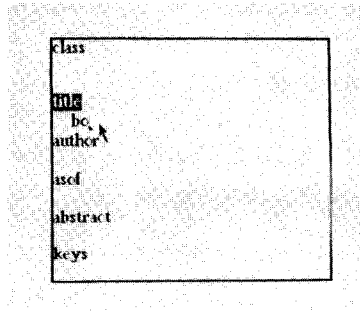
Third, it will be a framework for experiments with children and adults having to do with personal uses of computation that are not primarily related to programming. A secretary or high school student could set up a cross-indexed filing system for organizing data. We expect to learn a lot about the semantics of questions from such studies.

An illustration of the use of the *document* system is provided on the next four pages.

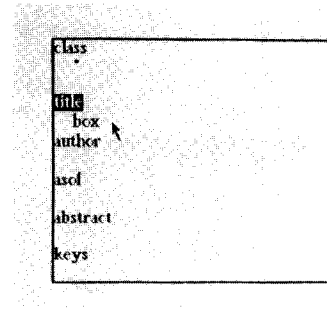
A Smalltalk Document System



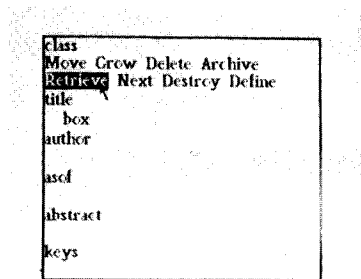
A "template" for a large class of cross-filed documents is displayed



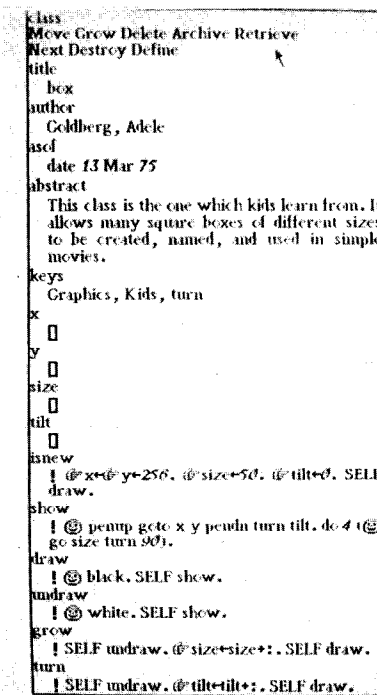
Moving the pointer near a paragraph causes its label to reverse. The word box is typed.



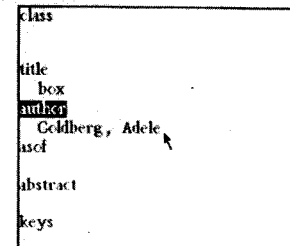
At this point, the other paragraphs could be filled in and a new document archived.



We decide to retrieve by pointing to the command in the documents menu. The system will find every document with the title "box".



Here it is. Note that the paragraphs have been filled in and there are additional fields.



Some other ways this document could have been found: We type Adele Goldberg's name into the author field.

```

class
Move Grow Delete Archive
Retrieve Next Destroy Define
title
  box
author
  Goldberg, Adele
ascl
abstract
keys
  
```

```

class
Move Grow Delete Archive Retrieve
Next Destroy Define
title
  box
author
  Goldberg, Adele
ascl
  date 6 Mar 75 to date 13 Mar 75
abstract
keys
  Graphics
  
```

```

class
Move Grow Delete Archive Retrieve
Next Destroy Define
title
  box,boxes
author
  Kay, Alan Goldberg, Adele
ascl
  before date 1 Jan 75
abstract
keys
  Kids
  
```

Here, the search will be for those versions of "box" having to do with "graphics" and written between 6 Mar 75 and 13 Mar 75.

We indicate a choice of titles and authors, and the keyword: kids.

record name: Q.class!

The search will be only for versions of "box" written by "Adele"

```

class
title
  box
author
  Goldberg, Adele
ascl
  date 13 Mar 75
abstract
  This class is the one which kids learn from. It allows many square boxes of different sizes to be created, named, and used in simple movies.
keys
  Graphics, Kids, turn
x
  □
y
  □
size
  □
tilt
  □
isnew
  ! @x*@y+256. @size+50. @tilt+0. SELF draw.
show
  ! @ penup goto x y pendn turn tilt. dc 4 (@ go size turn 90).
draw
  ! @ black. SELF show.
undraw
  ! @ white. SELF show.
grow
  ! SELF undraw. @size+size+. SELF draw.
turn
  ! SELF undraw. @tilt+tilt+. SELF draw.
  
```

```

class
title
  box
author
  Goldberg, Adele
ascl
  date 13 Mar 75
abstract
  This class is the one which kids learn from. It allows many square boxes of different sizes to be created, named, and used in simple movies.
keys
  Graphics, Kids, turn
x
  □
y
  □
size
  □
tilt
  □
isnew
  ! @x*@y+256. @size+50. @tilt+0. SELF draw.
show
  ! @ penup goto x y pendn turn tilt. dc 4 (@ go size turn 90).
draw
  ! @ black. SELF show.
undraw
  ! @ white. SELF show.
grow
  ! SELF undraw. @size+size+. SELF draw.
turn
  ! SELF undraw. @tilt+tilt+. SELF draw.
  
```

```

class
title
  box
author
  Goldberg, Adele
ascl
  date 13 Mar 75
abstract
  This class is the one which kids learn from. It allows many square boxes of different sizes to be created, named, and used in simple movies.
keys
  Graphics, Kids, turn
x
  □
y
  □
size
  □
tilt
  □
isnew
  ! @x*@y+256. @size+50. @tilt+0. SELF draw.
show
  ! @ penup goto x y pendn turn tilt. dc 4 (@ go size turn 90).
draw
  ! @ black. SELF show.
undraw
  ! @ white. SELF show.
grow
  ! SELF undraw. @size+size+. SELF draw.
turn
  ! SELF undraw. @tilt+tilt+. SELF draw.
  
```

Returning to the retrieved document: we do not want to end the sentence with a preposition, so we draw...

...with the pointer...

...to grab the word "from".

```

class
title
  box
author
  Goldberg, Adele
asof
  date 13 Mar 75
abstract
  This class is the one which kids learn. It
  allows many square boxes of different sizes
  to be created, named, and used in simple
  movies.
keys
  Graphics, Kids, turn
x
  □
y
  □
size
  □
tilt
  □
isnew
  | @x=@y=256. @size=50. @tilt=0. SELF
  draw.
show
  | @ penup goto x y pendn turn tilt. do 4 (@
  go size turn 90).
draw
  | @ black. SELF show.
undraw
  | @ white. SELF show.
grow
  | SELF undraw. @size=size+1. SELF draw.
turn
  | SELF undraw. @tilt=tilt+1. SELF draw.

```

We excise "from"...

```

class
title
  box
author
  Goldberg, Adele
asof
  date 13 Mar 75
abstract
  This class is the one which kids learn. It
  allows many square boxes of different sizes
  to be created, named, and used in simple
  movies.
keys
  Graphics, Kids, turn
x
  □
y
  □
size
  □
tilt
  □
isnew
  | @x=@y=256. @size=50. @tilt=0. SELF
  draw.
show
  | @ penup goto x y pendn turn tilt. do 4 (@
  go size turn 90).
draw
  | @ black. SELF show.
undraw
  | @ white. SELF show.
grow
  | SELF undraw. @size=size+1. SELF draw.
turn
  | SELF undraw. @tilt=tilt+1. SELF draw.

```

...point where we want it to go...

```

class
title
  box
author
  Goldberg, Adele
asof
  date 13 Mar 75
abstract
  This class is the one from which kids learn. It
  allows many square boxes of different sizes
  to be created, named, and used in simple
  movies.
keys
  Graphics, Kids, turn
x
  □
y
  □
size
  □
tilt
  □
isnew
  | @x=@y=256. @size=50. @tilt=0. SELF
  draw.
show
  | @ penup goto x y pendn turn tilt. do 4 (@
  go size turn 90).
draw
  | @ black. SELF show.
undraw
  | @ white. SELF show.
grow
  | SELF undraw. @size=size+1. SELF draw.
turn
  | SELF undraw. @tilt=tilt+1. SELF draw.

```

...and place it back in the correct place.

```

class
title
  box
author
  Goldberg, Adele
asof
  date 13 Mar 75
abstract
  This class is the one from which kids learn. It
  allows many square boxes of different sizes
  to be created, named, and used in simple
  movies.
keys
  Graphics, Kids, turn, show
x
  □
y
  □
size
  □
tilt
  □
isnew
  | @x=@y=256. @size=50. @tilt=0. SELF
  draw.
show
  | @ penup goto x y pendn turn tilt. do 4 (@
  go size turn 90).
draw
  | @ black. SELF show.
undraw
  | @ white. SELF show.
grow
  | SELF undraw. @size=size+1. SELF draw.
turn
  | SELF undraw. @tilt=tilt+1. SELF draw.

```

A new key word is added.

```

class
title
  box
author
  Goldberg, Adele
asof
  date 13 Mar 75
abstract
  This class is the one from which kids learn. It
  allows many square boxes of different sizes
  to be created, named, and used in simple
  movies.
keys
  Graphics, Kids, turn, show
x
  □
y
  □
size
  □
tilt
  □
isnew
  | @x=@y=256. @size=50. @tilt=0. SELF
  draw.
show
  | @ penup goto x y pendn turn tilt. do 4 (@
  go size turn 90).
draw
  | @ black. SELF show.
undraw
  | @ white. SELF show.
grow
  | SELF undraw. @size=size+1. SELF draw.
turn
  | SELF undraw. @tilt=tilt+1. SELF draw.

```

We decided to replace "named"...

```

class
title
  box
author
  Goldberg, Adele
asof
  date 13 Mar 75
abstract
  This class is the one from which kids learn. It
  allows many square boxes of different sizes
  to be created, given names, and used in
  simple movies.
keys
  Graphics, Kids, turn, show
x
  □
y
  □
size
  □
tilt
  □
isnew
  | @x=@y=256. @size=50. @tilt=0. SELF
  draw.
show
  | @ penup goto x y pendn turn tilt. do 4 (@
  go size turn 90).
draw
  | @ black. SELF show.
undraw
  | @ white. SELF show.
grow
  | SELF undraw. @size=size+1. SELF draw.
turn
  | SELF undraw. @tilt=tilt+1. SELF draw.

```

...by "given names".

```

class
Move Grow Delete Archive Retrieve
Next Destroy Define
title
  box
  author
    Goldberg, Adele
  asof
    date 13 Mar 75
  abstract
    This class is the one from which kids learn. It
    allows many square boxes of different sizes
    to be created, given names, and used in
    simple movies.
  keys
    Graphics, Kids, turn, show
  x
    □
  y
    □
  size
    □
  tilt
    □
  isnew
    ! @x=@y+25@. @size+5@. @tilt+@. SELF
    draw.
  show
    ! @ penup goto x y pendu turn tilt. do 4 (@
    go size turn 90).
  draw
    ! @ black. SELF show.
  undraw
    ! @ white. SELF show.
  grow
    ! SELF undraw. @size+size$. SELF draw.
  turn
    ! SELF undraw. @tilt+tilt$. SELF draw.

```

```

picture
Move Grow Delete Archive Retrieve
Next Destroy Define
Title
  polygons
  a
  Intro
  b
  PostLog
  pict
  picedit
  [edit para]
  [namewidth=64]
  [record picture]

```

```

picture
Move Grow Delete Archive Retrieve
Next Destroy Define
Title
  polygons
  a
  Intro
  b
  PostLog
  pict
  picedit
  [edit para]
  [namewidth=64]
  [record picture]
total number of items is 1

```

The document is rearchived. You may see what this Smalltalk program does in the next section.

Here we retrieve a document...

...called "polygons" which...

```

picture
Title
  polygons
  a
  Intro
  b
  PostLog
  pict
  picedit
  [edit para]
  [namewidth=64]
  [record picture]
total number of items is 1

```

```

picture
Move Grow Delete Archive Retrieve
Next Destroy Define
Title
  my very first picture
  a
  Intro
  b
  PostLog
  pict
  picedit
  [edit para]
  [namewidth=64]
  [record picture]
total number of items is 1

```

```

picture
Title
  my very first picture
  a
  Intro
  b
  PostLog
  pict
  picedit
  [edit para]
  [namewidth=64]
  [record picture]
total number of items is 1

```

...has some pictures in it, as well as text.

Another picture document

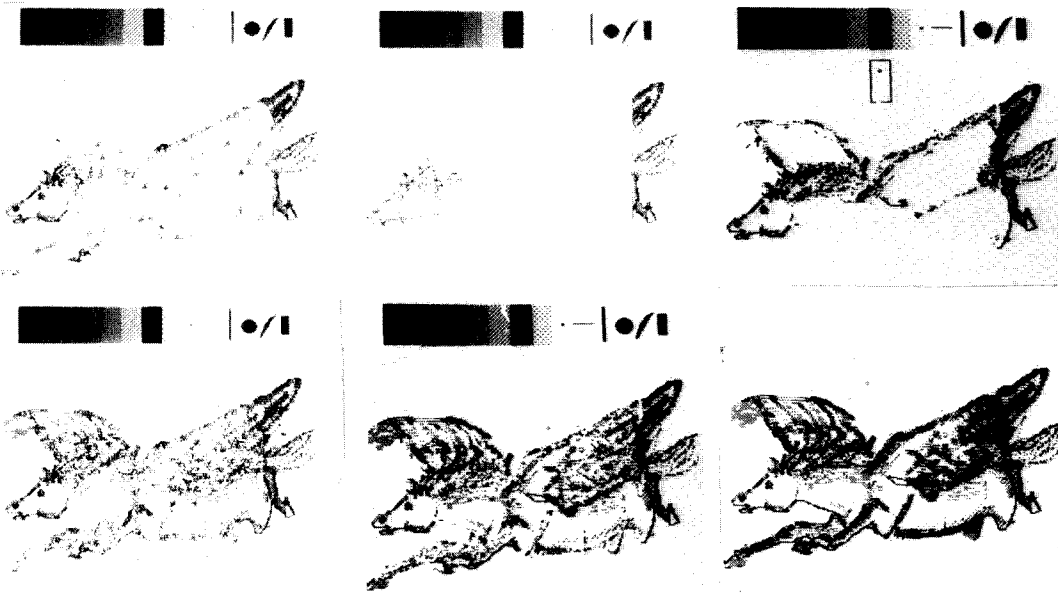
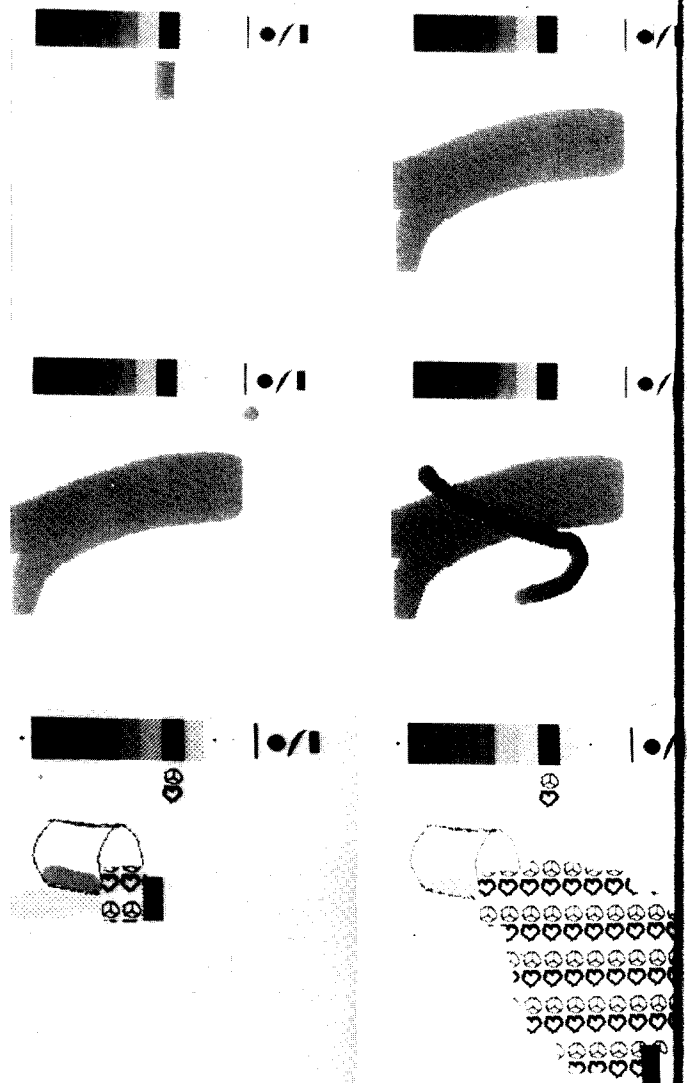
Drawing/Painting

The many small dots required to display high quality characters (about 500,000 for 8-1/2" x 11" sized display) also allows sketching quality drawing, "halftone painting", and animation.

The sequence below shows a sketch being planned, scrubbed out, edited, and then finished off to the owner's pleasure. The dots on the display are either black or white, as are the dots in newspaper photos. The subjective effect of grey scale is caused by the eye fusing an area containing a mixture of small black and white dots. The pictures to the right show a palette of paint dots (shown as toned patterns) and some brushes. A brush can be grabbed with the pointing device, dipped into a paint pot, and then the half-tone can be swabbed on as a function of the size, shape and velocity of the brush. The fourth picture shows a darker tone applied with a circular brush...and so on, editing as one goes, until done.

As already seen, brushes can be 0-dimensional (like a pencil point), 1-dimensional (like a hoe or rake), or 2-dimensional. The set of pictures shows a rather large rectangular brush after swabbing on some tone. Then a circular brush is grabbed. The last pair of pictures shows a heart-peace symbol shaped brush used to give the effect of painting wallpaper.

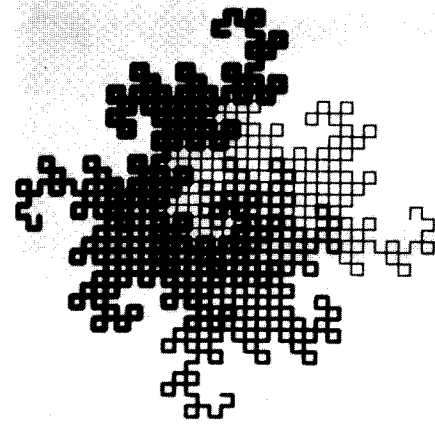
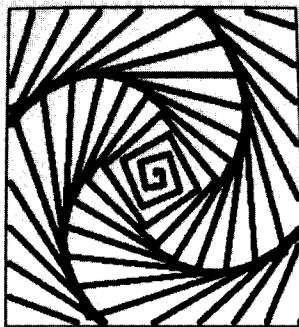
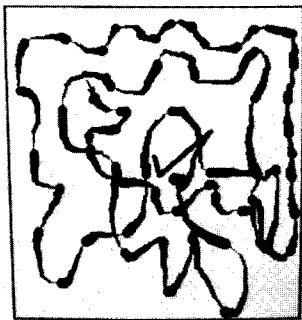
Pictures are manipulable objects to the Dynabook and may be stored in the same way as character fonts and texts, and printed on hard copy.



Pegasus

Turtles

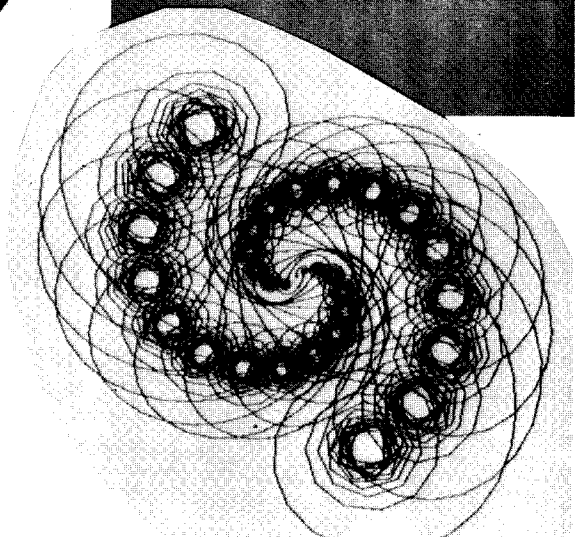
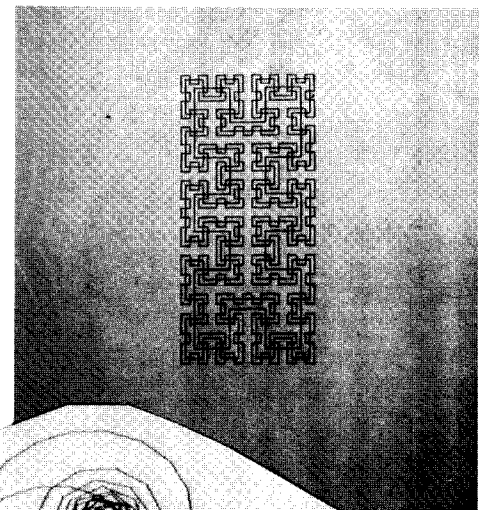
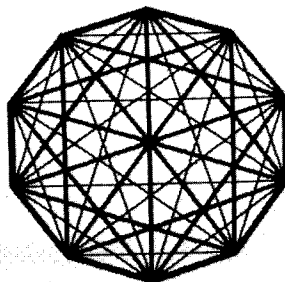
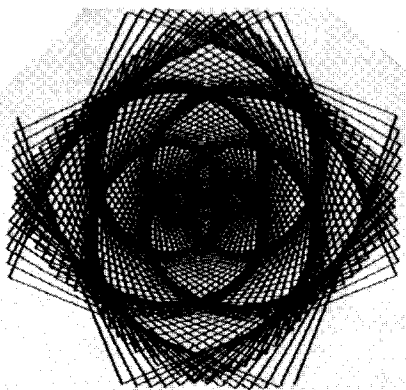
Curves are drawn by a *turtle* which crawls about on the screen leaving a track or not depending on whether its *pen* is up or down. (Straight lines are curves with zero curvature.) In the Dynabook, *turtles* are members of a class that can selectively draw with black or white ink, and change the thickness of the track. Each turtle lives in its own *window*, careful not to traverse its window boundaries, and adjusting as its window changes size and position. In color-Smalltalk, *turtles* can select from 16 different color inks. A number of simple and elegant "turtle geometry" examples are shown on this and on the following page.



```

for i=1 to 4 @ home up turn 90*1
  @s width = i*2 dragon 8!

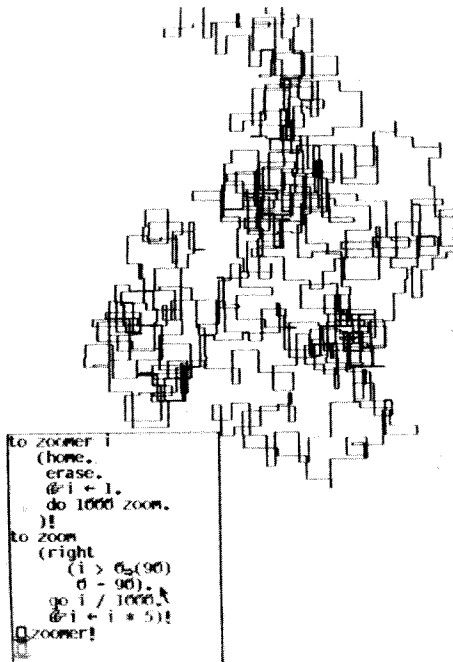
```



```

@home. @sinc=1. @ainc=0.
squirrel 1 89 256.
home. squirrel 1 91 256.1

```

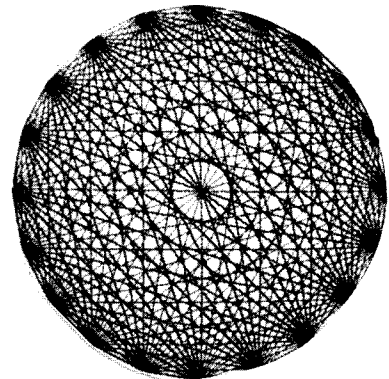



```

to zoomer
  (home.
   erase.
   @i = 1.
   do 1000 zoom.
  )!
to zoom
  (right
   (i > 60(90)
    d = 90).
   go i / 1000.
   @i = i + 5)!
zoomer!
  
```

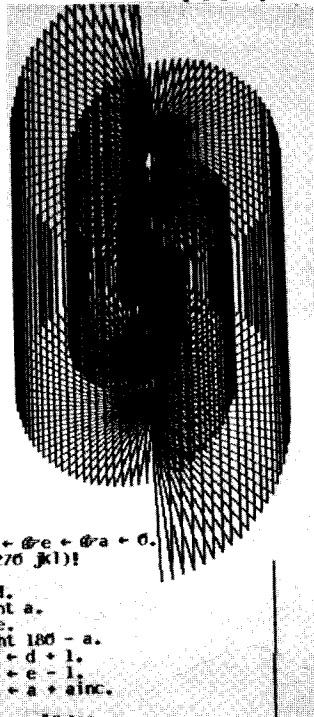
```

to pretty n side a i j xv yv
  (@a = turtle.
   :n.
   :side.
   @xv = vector n.
   @yv = vector n.
   do n
    (a knows
     (xv[n] + x.
      yv[n] + y).
  )
  
```



```

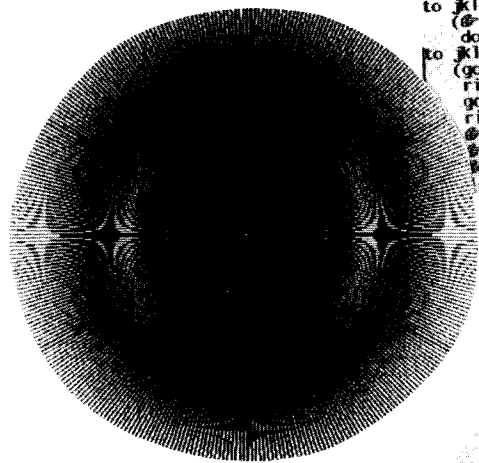
n by 2
i].
j].
yv(j + 1)!!!
  
```



```

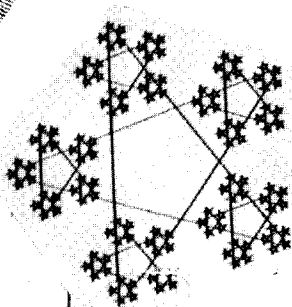
to k11
  (@d = @e = @a = 0.
   do 270 k1)!
to k1
  (go d.
   right a.
   go e.
   right 180 - a.
   @d = d + 1.
   @e = e - 1.
   @a = a + ainc.
  )
  
```

Turtle Geometry Designs



```

to sun
  (home erase go 200 right 180 do 300 burst).
to burst
  (go 400.
   right 179)!
sun!
  
```

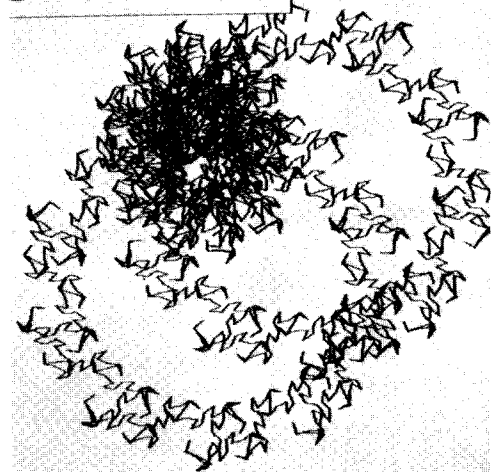


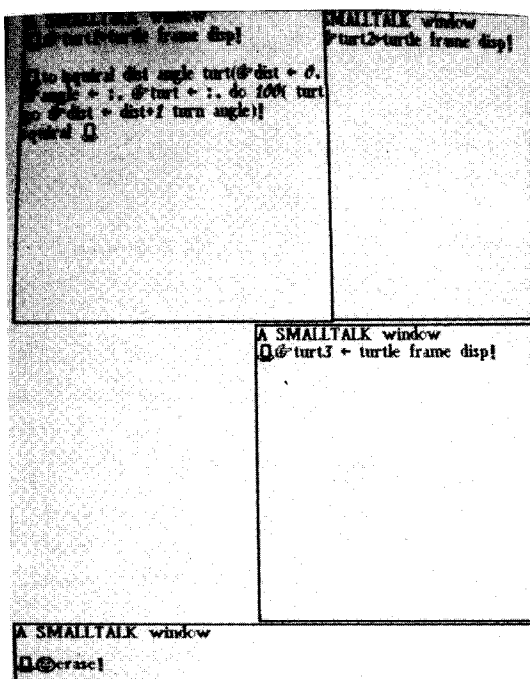
```

to stars i
  (i > 40(do points
   (go 1.
    right angle.
    stars i / 3)!!!
   @angle=144. @points=5.
   stars 200!
  )
  
```

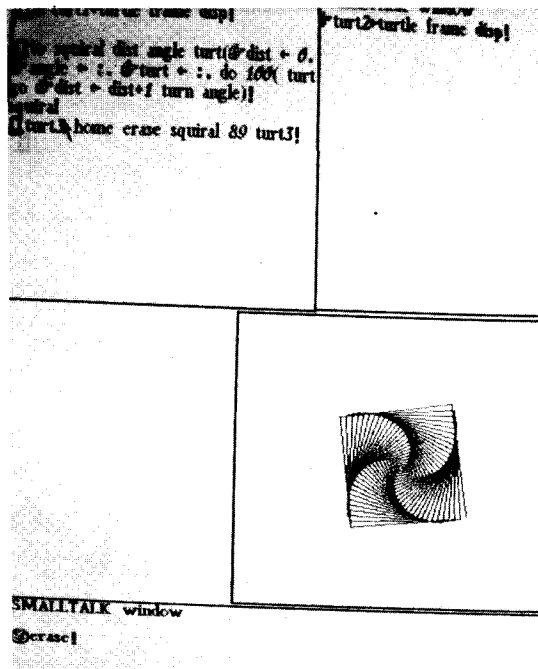
```

to boomer i
  (home.
   erase.
   @i = 0.
   do 3131
    (go 10.
     right @i + i + 95)!!!
  )boomer!
  
```

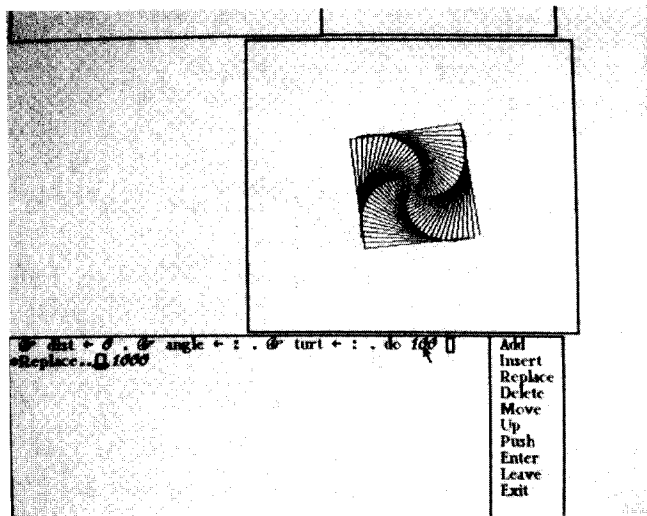




We created three windows. A turtle lives in each window. Their names are "turt1", "turt2", and "turt3".

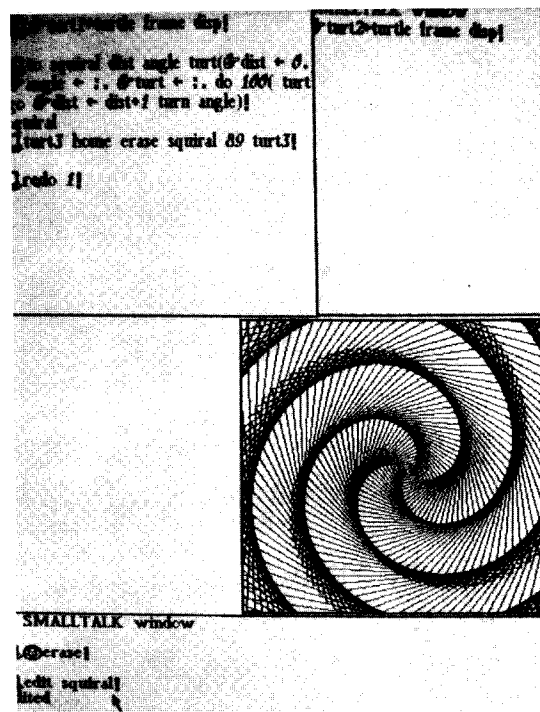


We define a squiral program that expects two messages: the angle the turtle will turn each time it draws a line, and the name of the turtle who will do the work.

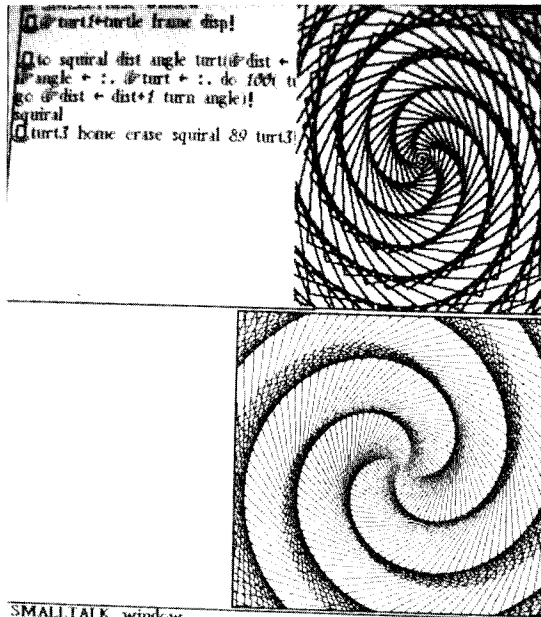


Turtle "turt3" goes to the center of its window, clears the window, and draws a squiral with angle 89.

The squiral was too small. We point at another window and edit the program to iterate 1000, rather than 100, times.



We go back to the first window and redo the previous commands. Now the squiral is bigger than the window, but the design is clipped on the window boundaries.

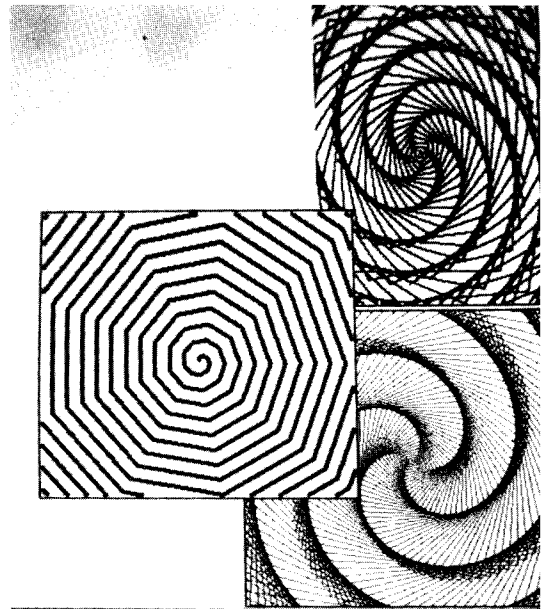


SMALLTALK window

@erase!

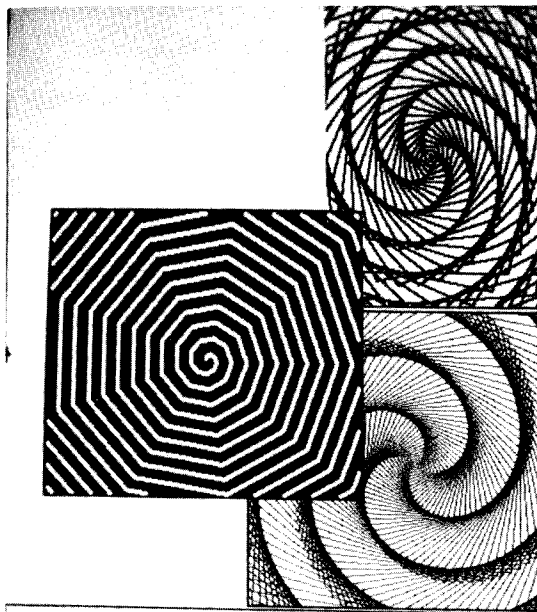
edit smallt

We send a message to turtle "turt2" to draw lines that are doubly thick; "turt2" then draws a squiral with angle equal to 72.



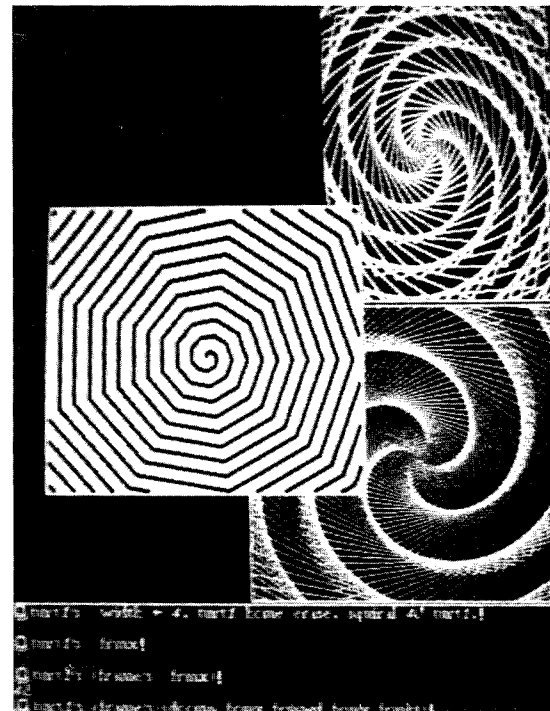
@turt1's width + 4. turt1 home erase. squiral 40 turt1.

After moving the third window lower on the display screen, we set the width of "turt1" to 4; "turt4" draws a squiral with angle equal to 40.



@turt1's width + 4. turt1 home erase. squiral 40 turt1!
 @turt1's frmx!
 @turt1's (frames frmx!)

We complement black and white in turt1's window.



@turt1's width + 4. turt1 home erase. squiral 40 turt1!
 @turt1's frmx!
 @turt1's (frames frmx!)
 @turt1's (frames frmx!)

Now we complement the entire display screen.

Animation and Music

Animation, music, and programming can be thought of as different *sensory views* of dynamic processes. The structural similarities among them are apparent in Smalltalk, which provides a common framework for expressing these ideas.

All of the systems are equally controllable by hand or by program. Thus, drawing and painting can be done using a pointing device or in conjunction with programs which draw straight lines or curves, fill in areas with tone, show perspectives of 3-dimensional models, and so on. Any graphic expression can be animated, either reflecting a simulation (such as bouncing objects in free space) or by example (giving an "animator" program a sample trace or a route to follow).

Music is controlled in a completely analogous manner. The Dynabook can act as a "super synthesizer" getting direction either from a keyboard or from a "score" (a sequence of actions over time). The keystrokes can be captured, edited and played back. Children can both learn to play (coordinating their minds and bodies) and compose at the same time because they do not have to spend several years becoming good enough technically to play their own compositions.

Timbres are the "fonts" of musical expression as they contain the quality and mood which different instruments bring to an orchestration. They may be captured, edited and used dynamically.

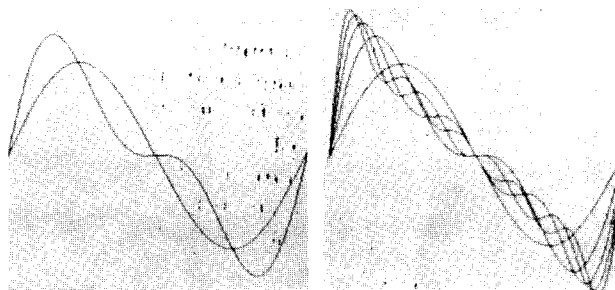
Simulation

In a very real sense, simulation is the central notion of the Dynabook. Each of the previous examples has shown a simulation of visual or auditory media. Descriptions and the ability to manipulate them are the content of the Dynabook. The range of possible simulations extends far beyond media. Here are a number of examples of interesting simulations done by a variety of users in the last eighteen months.

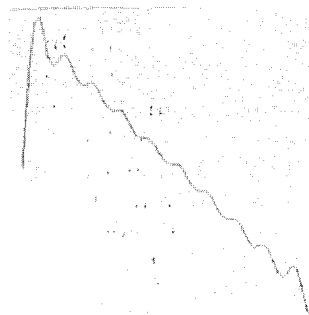
An Animation System Programmed by Animators. Several professional animators visited us with a long-held dream for a magic system which would allow them to create high-quality animations by simply (and



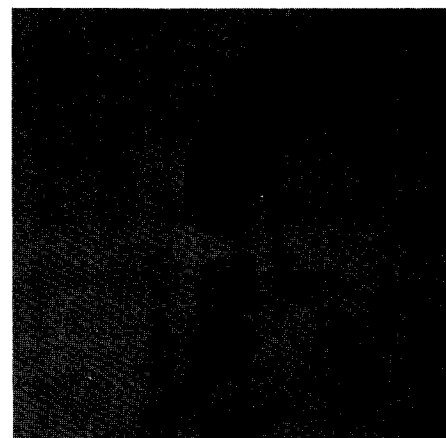
a "captured" score



generation of a timbre



accumulation
of harmonics



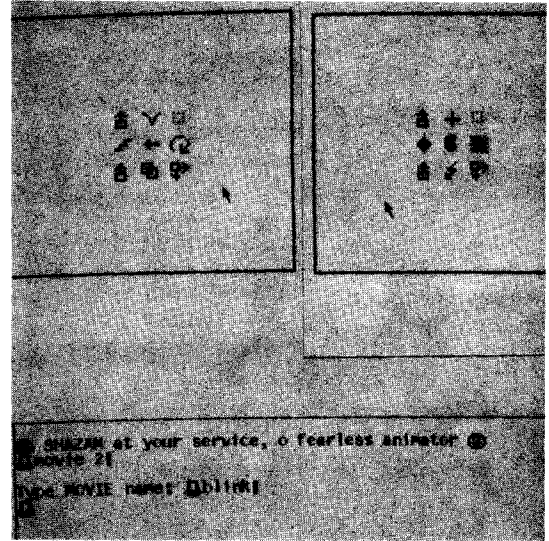
capturing three frames of
a dripping faucet movie

literally) "waving their hands". They wanted to be able to draw and paint pictures which could then be animated in real-time by simply showing the system roughly what was wanted. Desired changes would be made by iconically editing the animation sequences. After a fair amount of design thinking and several tries, their ideas yielded SHAZAM, a system written in Smalltalk, which met many of their goals.

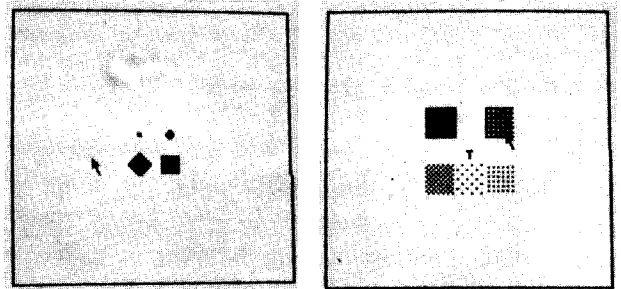
Much of the design of SHAZAM is an automation of the media with which animators are familiar: *movies* consisting of sequences of *frames* which are a composition of transparent *cels* containing foreground and background drawings. Besides retaining these basic concepts of conventional animation, SHAZAM incorporates some creative supplementary capabilities.

Animators know that the main action of animation is due not to an individual frame, but to the change from one frame to the next. It is therefore much easier to plan an animation if it can be seen moving as it is being created. SHAZAM allows any cel of any frame in an animation to be edited while the animation is in progress. A library of already created cels is maintained. The animation can be single-stepped; individual cels can be repositioned, reframed, and redrawn; newframes can be inserted; and a frame sequence can be created at any time by attaching the cel to the pointing device, then "showing" the system what kind of movement is desired. The cels can be stacked for background parallax; *holes* and *windows* are made with *transparent* paint. Animation objects can be painted by programs as well as by hand. The control of the animation can also be easily done from a Smalltalk simulation. For example, an animation of objects bouncing in a room is most easily accomplished by a few lines of Smalltalk which expresses the class of bouncing objects in physical terms.

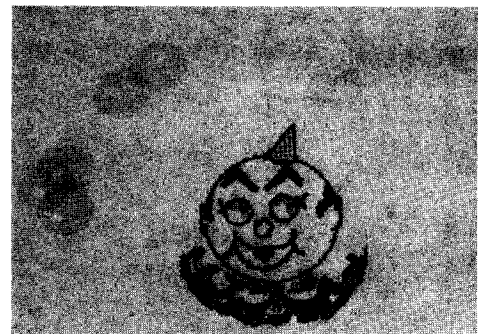
Several sequences depicting *frames* of animated *movies* are provided on this and on the next four pages.

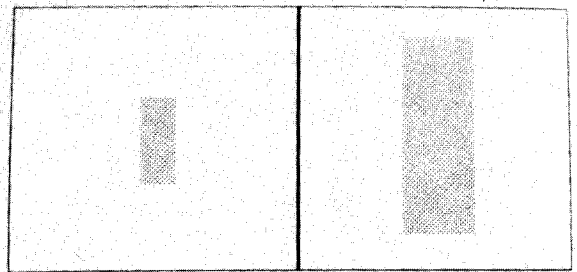
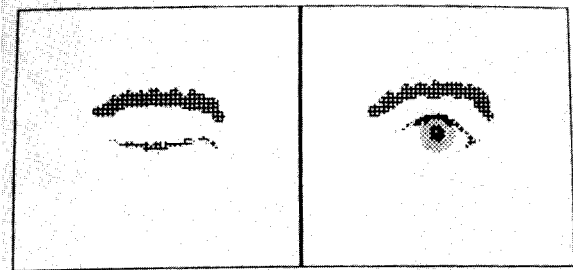
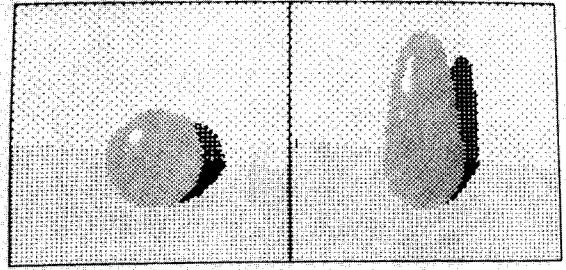
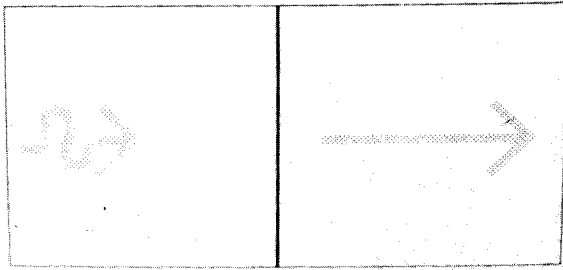


a menu of commands for playing a movie, altering the movie window, and selecting new pictures and positions

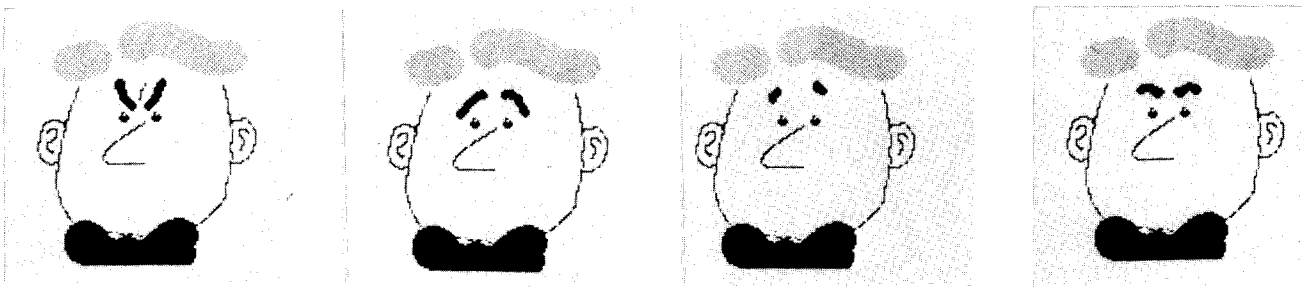
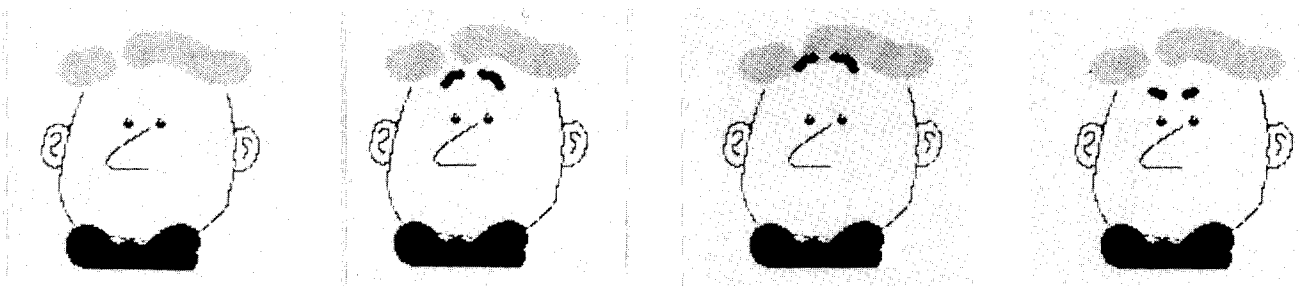


a menu of commands for painting a movie frame, lets the user select a paint tone or a paint brush.

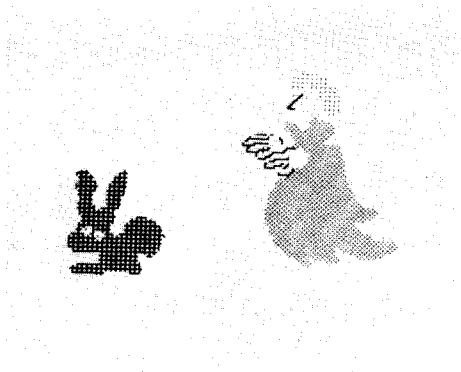
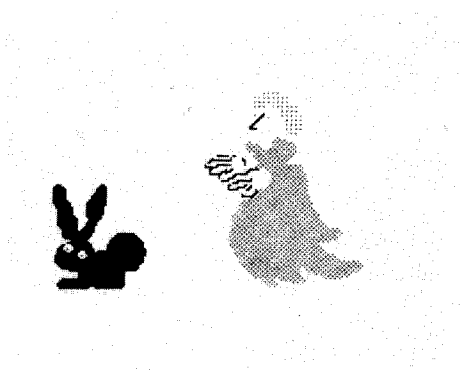
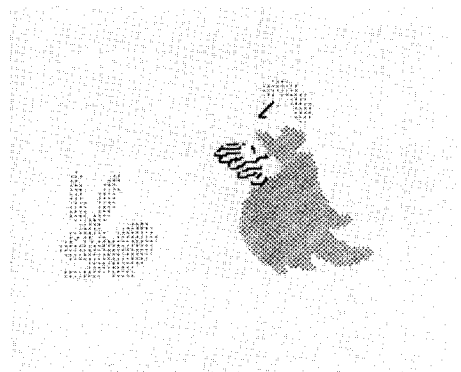
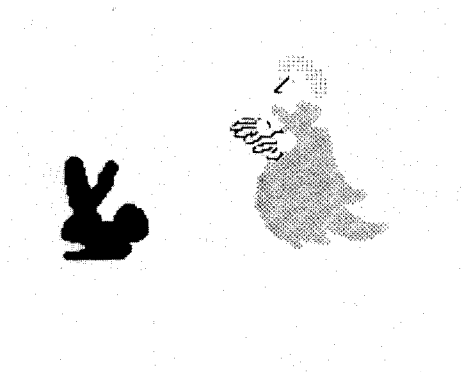
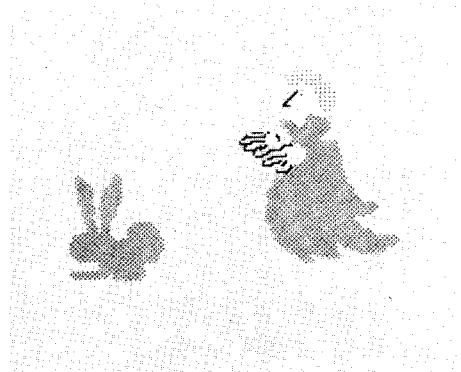
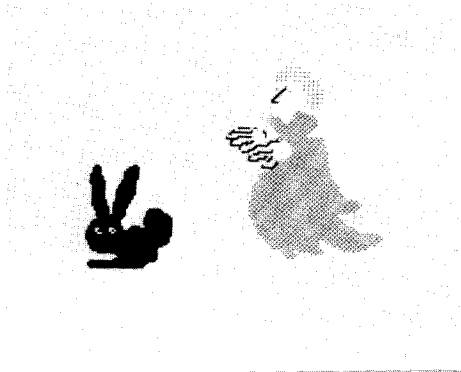




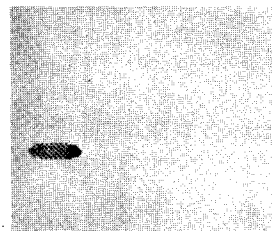
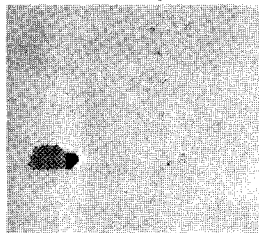
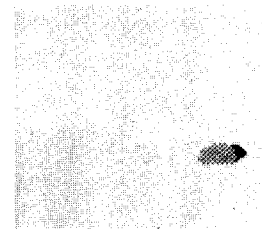
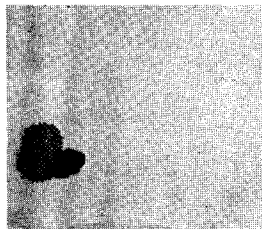
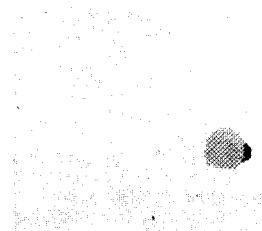
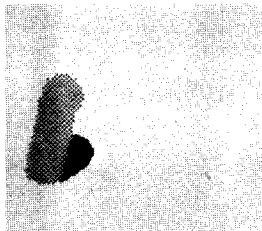
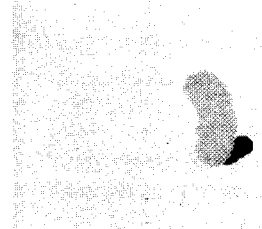
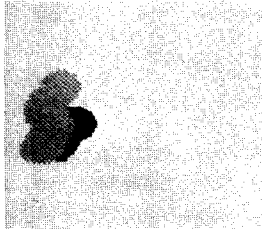
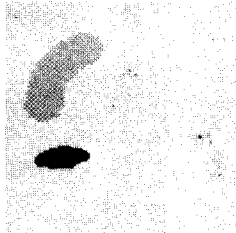
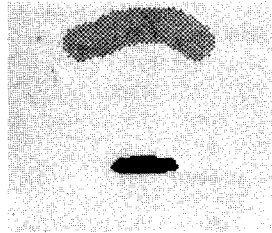
blinking pairs--two frame movies



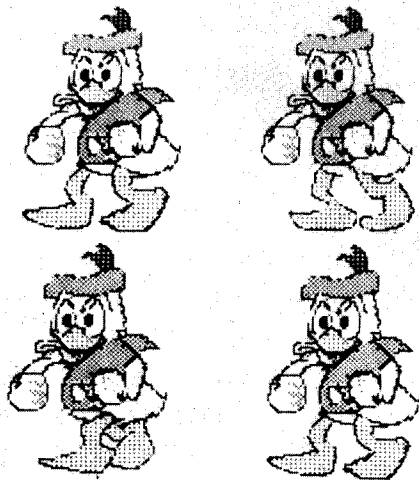
a movie of eyebrows superimposed on a man's face



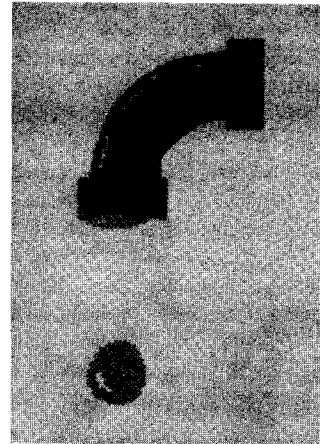
the Great Martino makes the Wabbit disappear



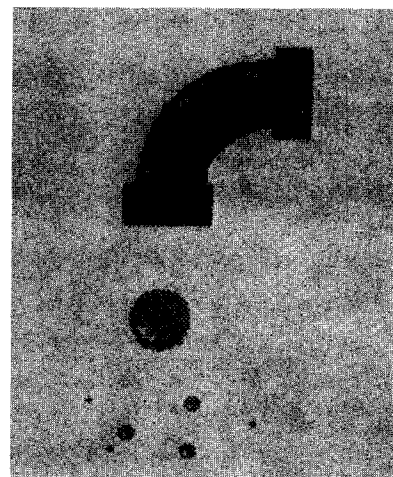
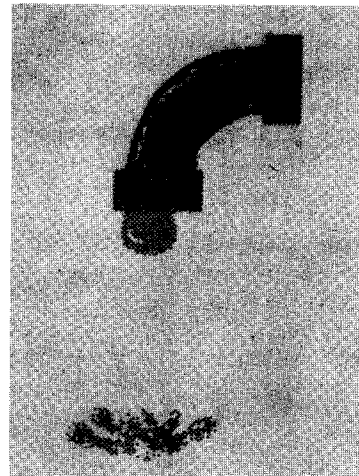
the worm Olga Korbutt



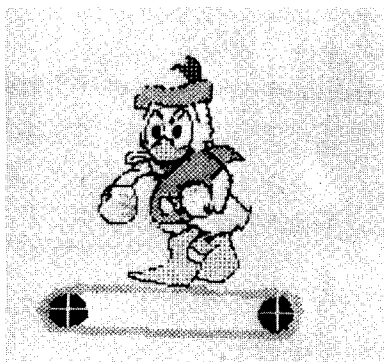
Donald Duck movie...



...superimposed on
a movie of a
conveyer belt

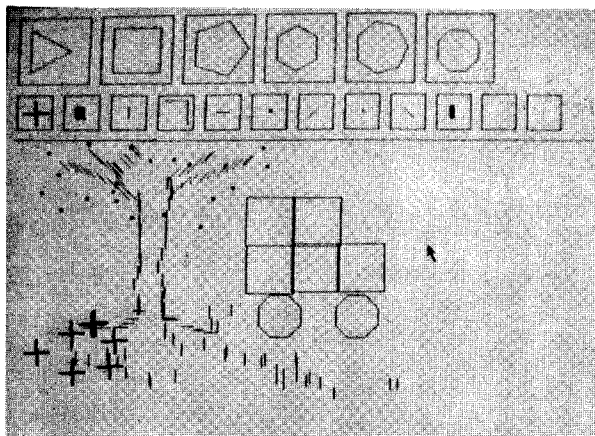
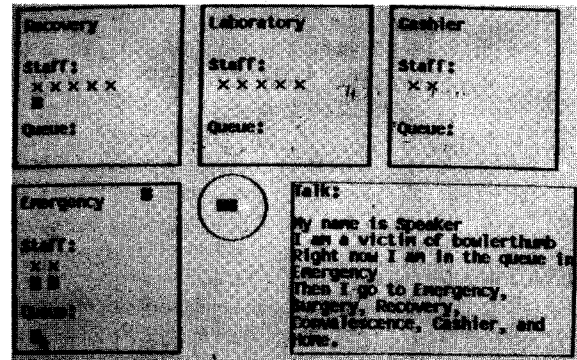
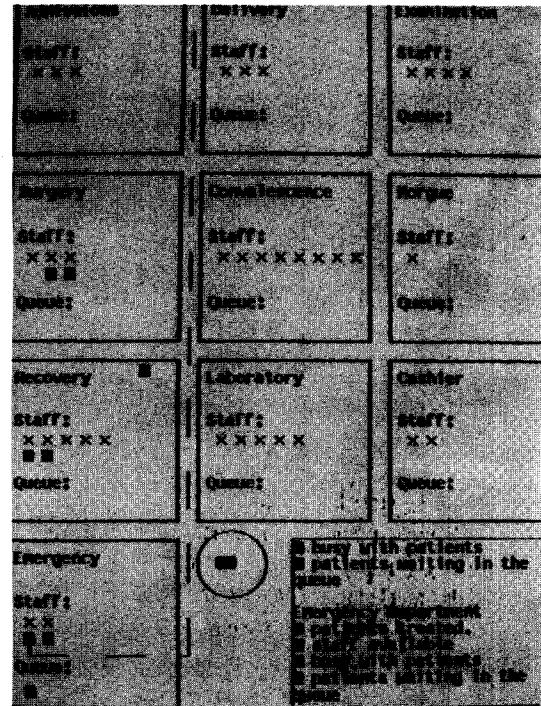


a dripping faucet

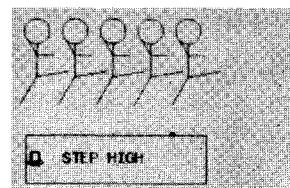
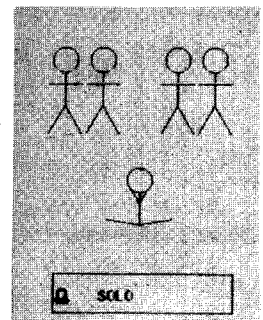
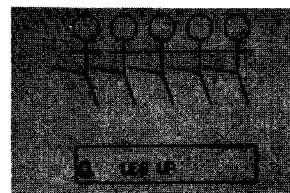
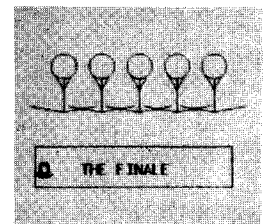
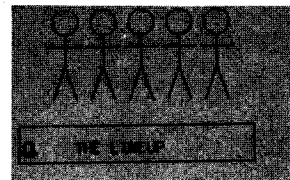


A Hospital Simulation Programmed by a Decision-Theorist. The second simulation represents a hospital in which every department has resources which are used by patients for some duration of time. Each patient has a schedule of departments to visit; if there are no resources (doctors, beds) available, the patient must wait in line for service. The Smalltalk description of this situation involves one description for the class of patients and one for the class of departments. The generalization to any hospital configuration with any number of patients is part of the simulation. Again, the dynamic state of the simulation in progress is displayed by "peering" into the hospital "world" in order to abstract its contents. The particular example captured in the pictures to the right shows patients lining up for service in emergency. It indicates that there is insufficient staff available in that important area.

A Drawing and Painting System Programmed by a Child. We feel successful in providing a tool building system because one young girl, who had never programmed before, decided that a pointing device ought to let her draw on the screen. She then built a sketching tool without ever seeing ours--a Smalltalk class definition for paint brushes. She constantly embellished it with new features including a menu for brushes selected by pointing. Then she turned her attention to animating stick figure drawings and has demonstrated her first "drawing" system for multiple figures. This girl is currently teaching her own Smalltalk class; her students are seventh-graders from her junior high school.



Marian's painting system



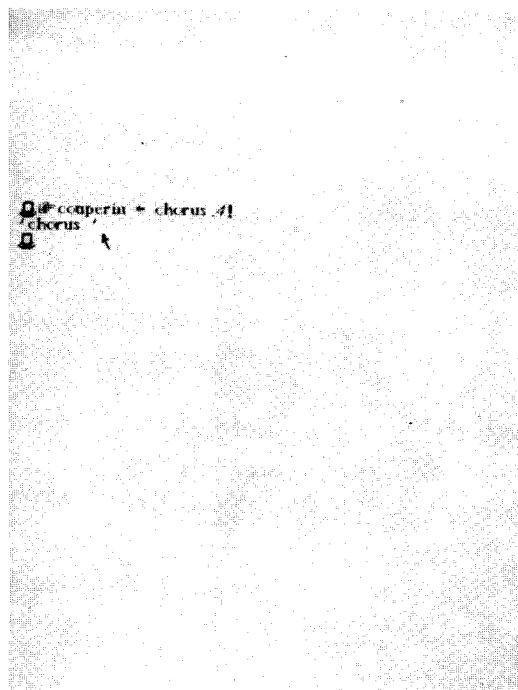
An Audio Animation System Programmed by Musicians. Animation can be considered to be the coordinated parallel control through time of images conceived by an animator. Likewise, a system for representing and controlling musical images can be imagined which has very strong analogies to the visual world. Music is the design and control of images (pitch and duration changes) which can be *painted* different *colors* (timbre choices); it has synchronization and coordination, and a very close relationship between audio and spatial visualization. Music is another rich world where the benefits of contact go far beyond learning a skill, art, or trade.

The system called TWANG was designed by taking many yearnings and dreams which musicians have had for centuries and turning them into reality using Smalltalk and the Dynabook.

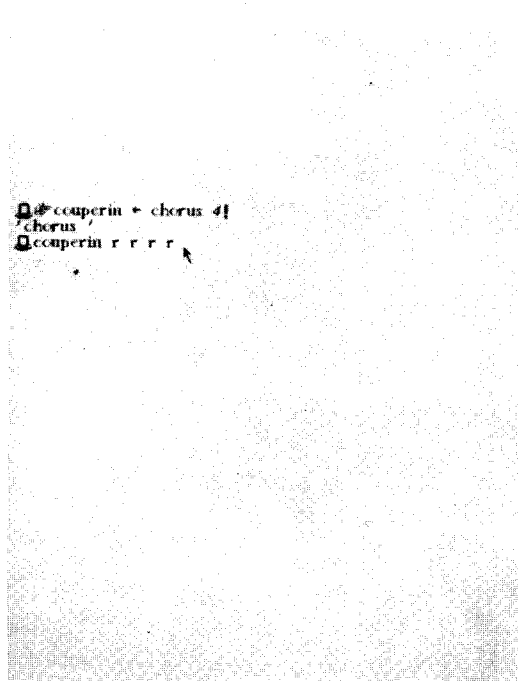
One of these dreams was to be able to *capture*, edit, and replay musical events.

The Smalltalk model created by the musicians has the notion of a *chorus* which, as with an animation *movie*, contains the main control directions for an overall piece. A chorus is a kind of *rug* with a warp of parallel sequences of "pitch, duration, and articulation" commands, and a woof of synchronizations and global directives. As in SHAZAM, the control and the *player* are separate; a given animation sequence can animate many drawings. In TWANG, a given chorus can tell many different kinds of instrumentalists what in general should be played; these *voices* can be synthetic timbres or timbres captured from real instruments; other musical effects such as vibrato, portamento, and diminuation are also available.

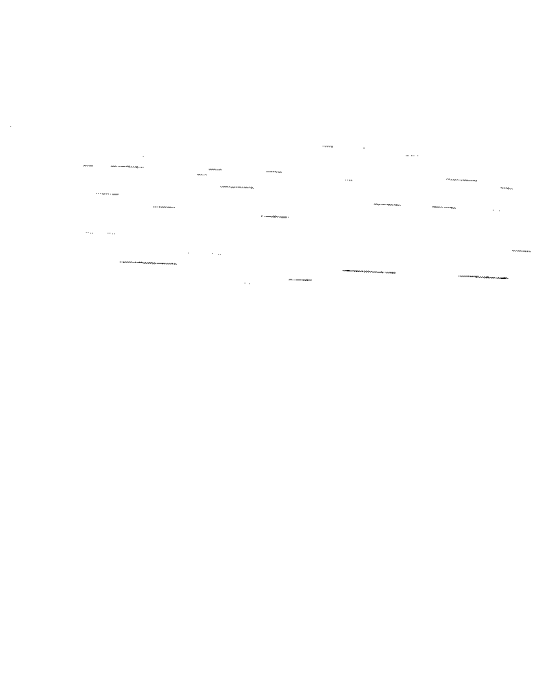
A chorus can be *drawn* using the pointing device, or it can be *captured* by playing it on a keyboard. It can be played back in real-time and dynamically edited in a manner very similar to the animation system. The following set of pictures traces a user through a sequence of playing, editing, and replaying a piece.



We would like to create and manipulate a musical animation. The name "couperin" is given to a new instance of class chorus with four voices.



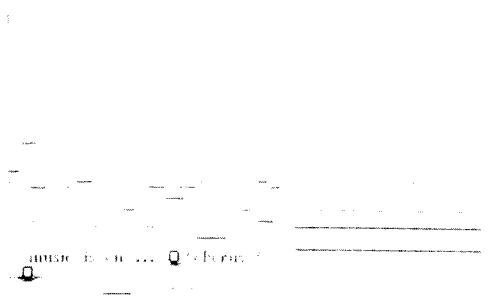
We now tell "couperin" that we wish to record each of its four voices. We could say "s" for silence, and later "p" for play while others are recorded.



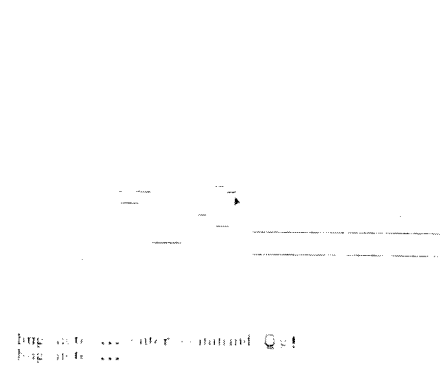
The music is now being played on the keyboard, captured by "couperin", and displayed on the screen. The simplified format (where vertical placement represents pitch as in



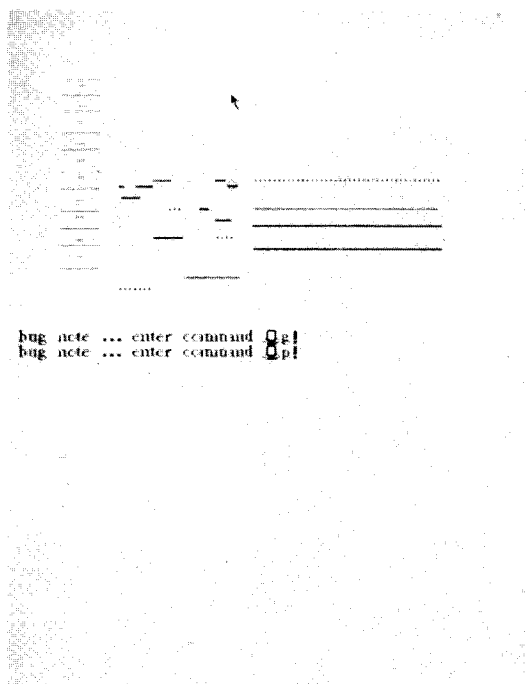
normal use, but duration is shown as a horizontal length rather than a flag) is much easier to read for a beginner than standard notation.



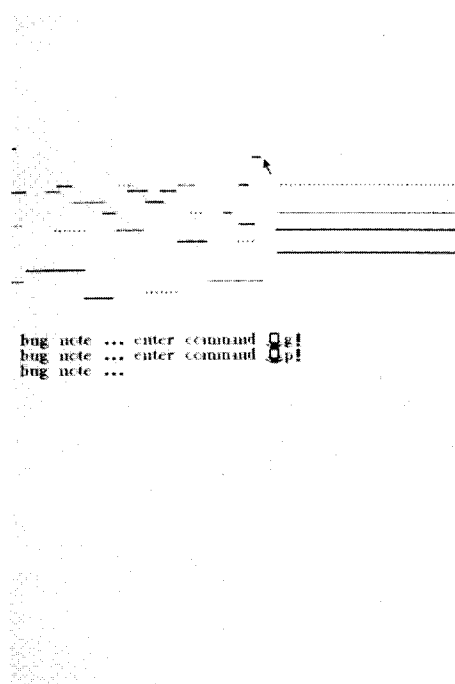
The piece ends with a long held chord.



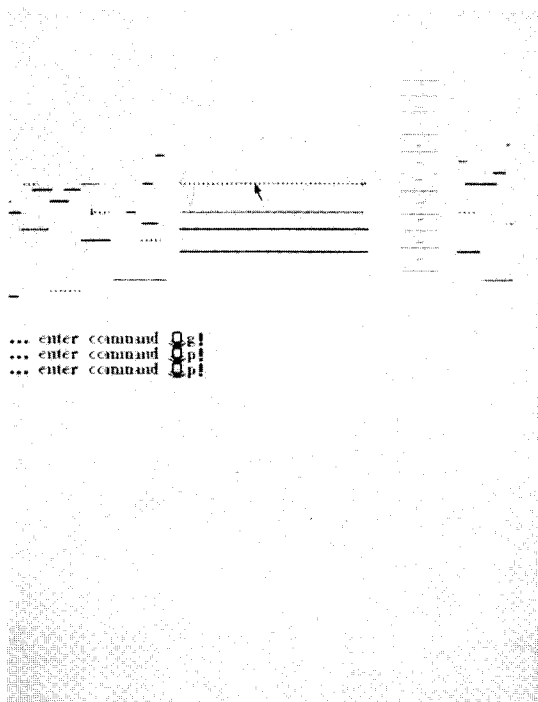
We play it back until a place we want to change occurs (and appears on the screen). We point at the note we wish to change.



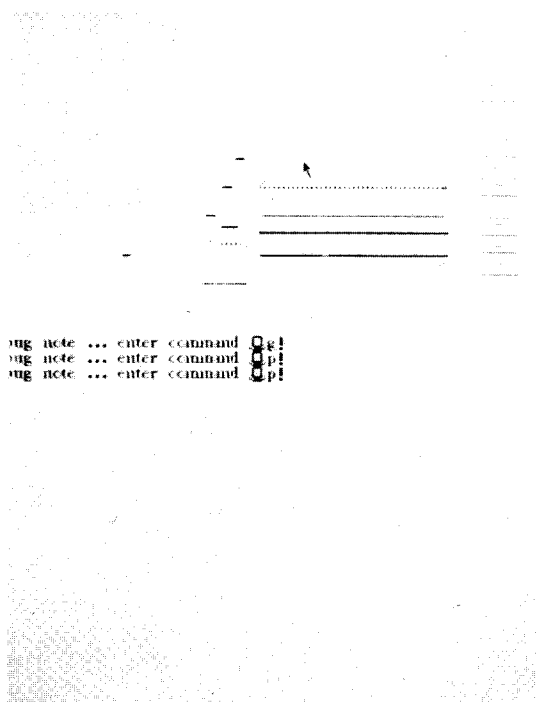
The command "p" allows various pitches to be tested (by vertical placement of the pointing device).



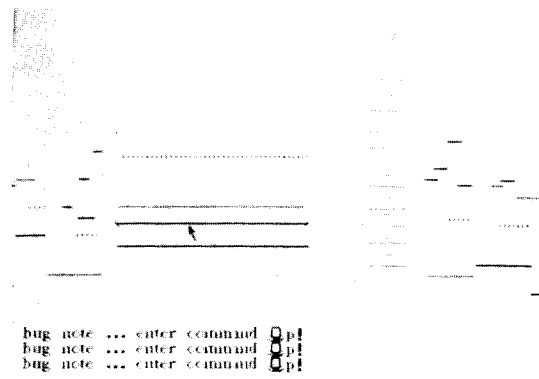
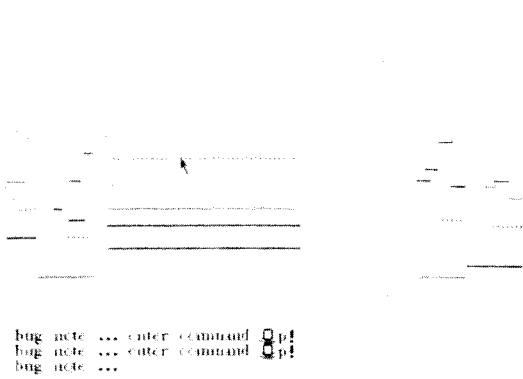
We find the note we want and the old note is changed.



The voicing of the final chord could be vastly improved so we point at the top voice (which is the "tonic" note)...

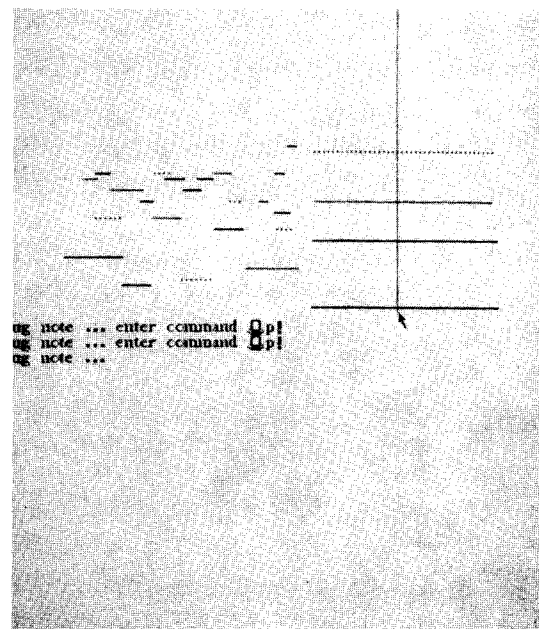
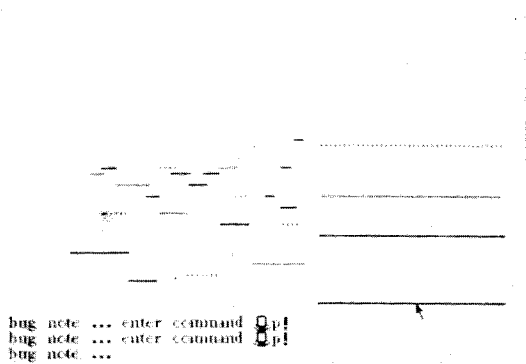


...look for a pitch which goes well with the immediately preceding note...



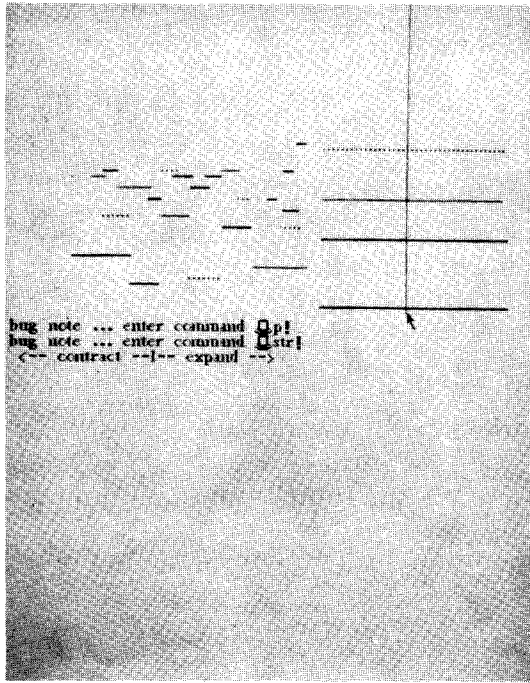
...and change it. The new note is the "third" of the chord.

We now have a doubled "third" (a no-no), so we grab it...

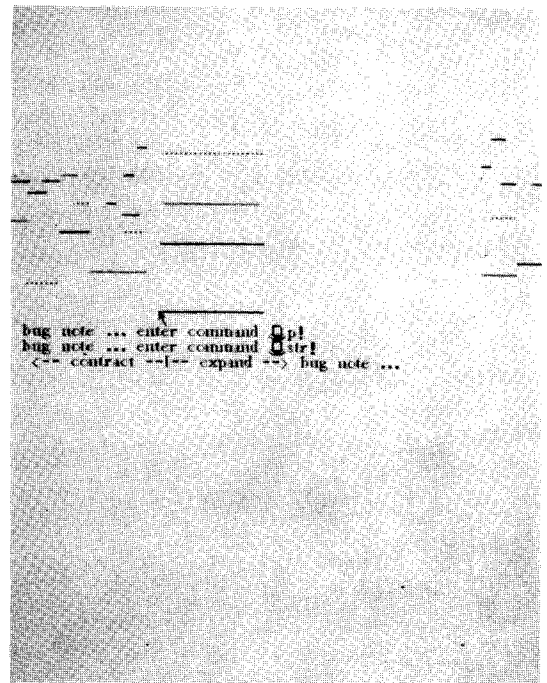


and move it to double the "tonic" instead (a yes-yes). The spelling of the chord from bottom to top is now: tonic, tonic, fifth, third; a nice voice for an ending which is nicely led by the first note we changed.

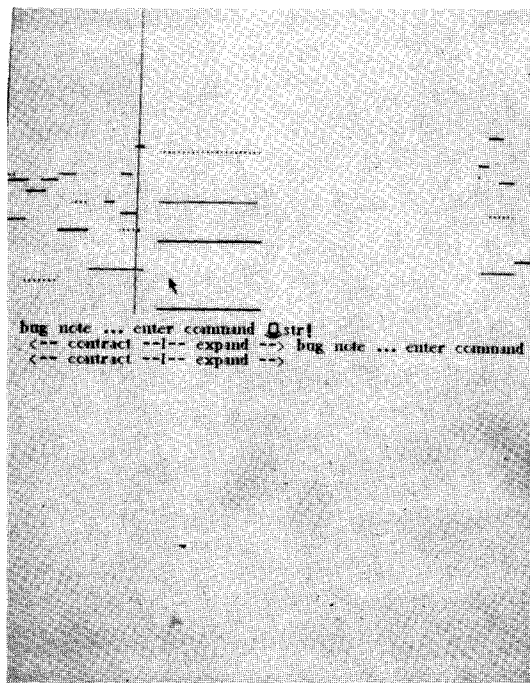
The final chord is too long we slice it, and...



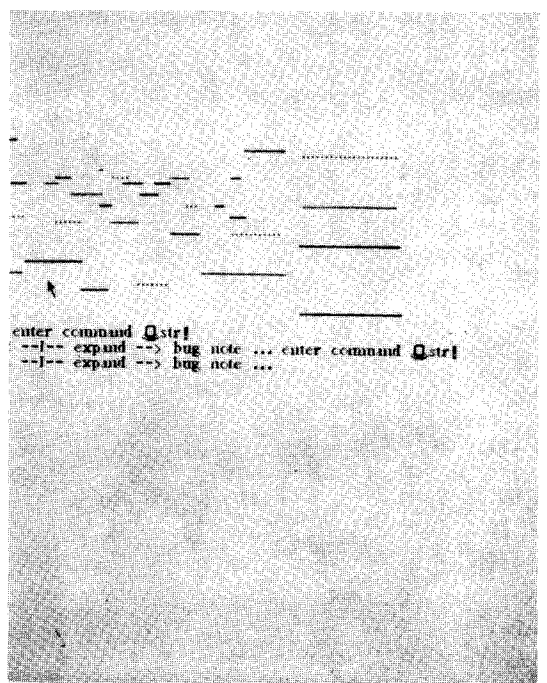
...delete the front part...



...to shorten it.



We would like to retard the lead into the ending, so we slice it and...



stretch it.

For children, this facility has a number of benefits:

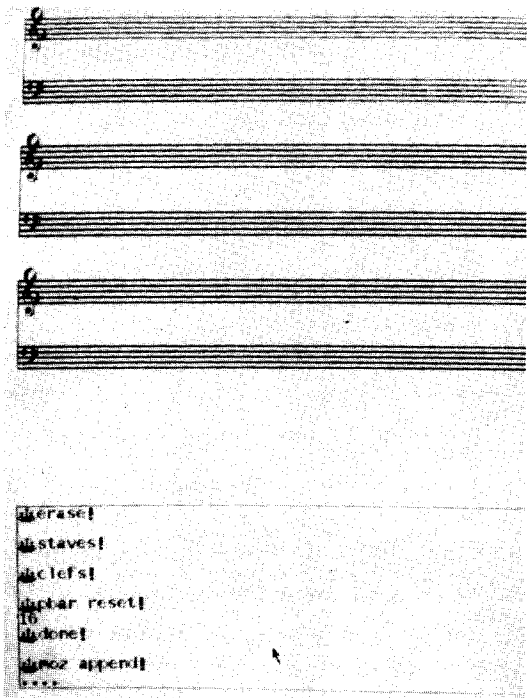
a. The semantics of the system are easy to understand since they intentionally are anthropomorphisms from the real world.

b. The strong similarities between the audio and visual worlds, and between the arts and the sciences, are emphasized because a single vernacular *which actually works* in both worlds is used for description.

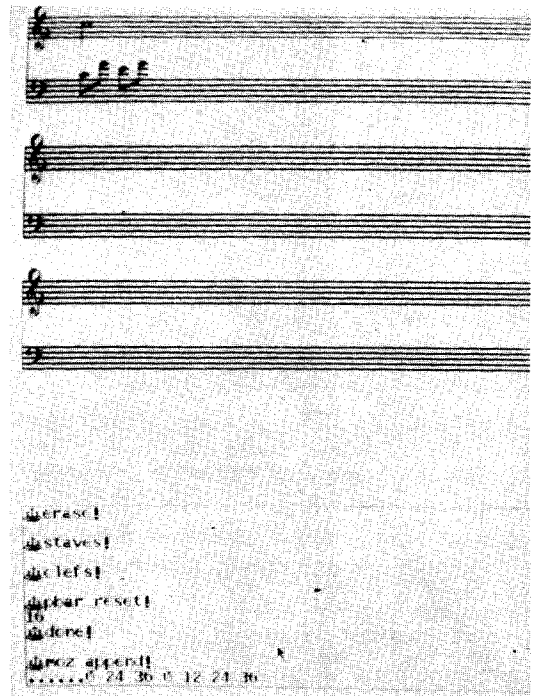
c. Children can gain skill and coordination by learning how to play. The system will show and play for them what they just tried, then allow them to compare their efforts to a more expert model, much in the manner of skiing instruction.

d. The arts and skills of composing can be learned at the same time since tunes may be drawn in by hand and played by the system. A line of music may be copied, stretched, and shifted in time and pitch; individual notes may be edited. Imitative counterpoint (probably the best single basic for thinking about music) is thus easily created by the fledgling composer.

A Musical Score Capture System Programmed by a Musician. OPUS is a musical score capture system written by a novice programmer who is an experienced musician. The system produces a display of a conventional musical score from data obtained by playing a musical keyboard. OPUS is designed to allow incremental input of an arbitrarily complicated score (full orchestra with chorus, for example), editing pages of the score, and hard copy of the final result with separate parts for individual instruments. The sequence below shows a score being captured with the OPUS system.



This sequence shows a score being captured. The screen shows a blank score ready for some notes.



Part of a Mozart minuet is played on the musical keyboard. The notes appear on the screen.

erase!
staves!
clefs!
paper reset!
16
done!
append!

erase!
staves!
clefs!
paper reset!
16
done!
append!
..... 24 36 0 12 24 36 0 16 24 0 12 24 36

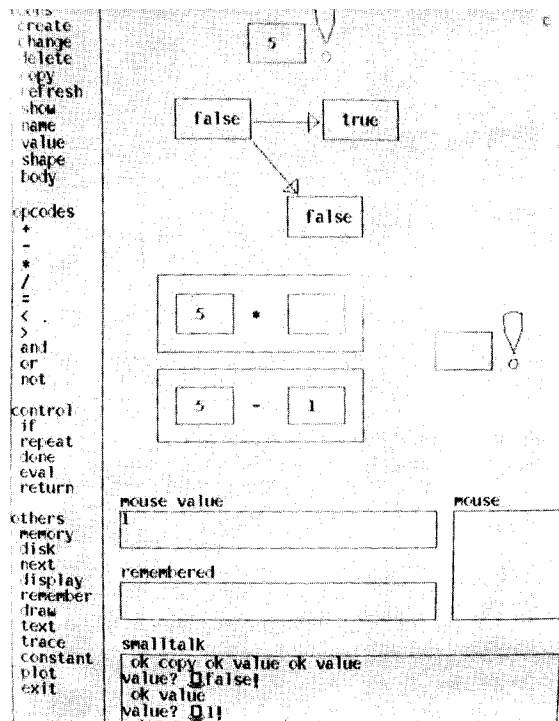
The current measure shows a triplet of 16 notes successfully recognized. The differences in duration of a 16th note triplet and a 16th note duplet is less than the normal temp error of an average instrumentalist.

The rest of the piece rolls in...

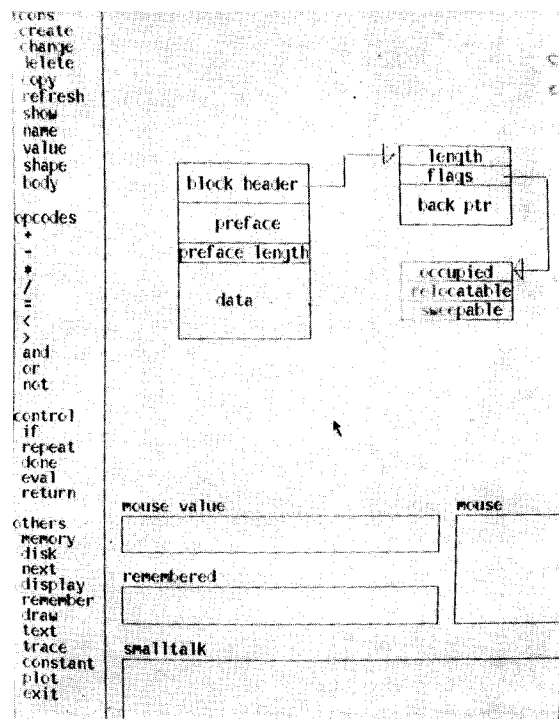
Programming Using Pictures and Examples as Programmed by a Computer Scientist. Even a casual study of human communication and creativity reveals some interesting guidelines for future interactive computer systems. Humans have a strong tendency to form extreme generalizations from isolated facts; they then wait until bugs appear before modifying the model. These generalizations are not easy to explain in the abstract; thus the communication of an idea from one human to another is usually done with the most concrete of examples, frequently aided by pictures. The automatic generalizing of the receiver, however, will also create an abstraction from the examples; this is usually checked by the participants exchanging further concrete instances of the new generalization.

Contemporary programming consists largely of constructing generalizations of ideas and actions which have concrete instances when the programs are executed. This communication is highly abstract, consisting of symbolic-parametric structures which frequently confuse people as to what they will do when they are actually executed. Almost the only programming activities which do not have this air of mystery to them are those which have a very direct "hands on" contact with the medium--such as CRT-text-editing, drawing, and playing music. In these cases, changes of state caused by the interaction with the medium are immediately perceived (seen, heard). The programs can be corrected if the state changes are different from expectation.

These ideas have led us to experiment with combining the semantic power for simulation of Smalltalk with an interface which allows the human user to communicate in terms of concrete examples (using pictures), where important state changes are shown during the act of programming. This experimental system, called PYGMALION [28], tries to generalize the examples in order to allow the human to remain in the concrete world where he feels most comfortable.



a stage in the definition of factorial showing a conditional, a recursive call, and arithmetic operations



an iconic data structure

IV. Smalltalk: A Communications Medium for Children of All Ages

Even more important than the hardware considerations are questions dealing with how an owner (child or adult) can communicate with his medium without constantly seeking the services of an expert. We wanted our system to be immediately and usefully controlled by its owner--simple things should be *very* simple (while not constraining later expert use) and complex things should be *very* possible.

Everything in Smalltalk is based on a few simple anthropomorphic metaphors having to do with communication, state, and classification. There are no "nouns" or "verbs", but rather *objects* in *process*. Every transaction, description, and control is thought of as sending messages to and receiving messages from objects in the system. Every object belongs to a class; every object has *memory*; objects communicate with each other by sending *messages*. A class contains the ability to recognize and reply to messages. Each class has certain capabilities such as drawing pictures, making musical (or other) noises, or adding numbers.

Many of the ideas in Smalltalk are consolidations and simplifications of ideas from the past, in particular: the Burroughs B5000 [5], SKETCHPAD [29], SIMULA[7,8], the FLEX machine [16,17,18], CDL [11], and (to a lesser extent) LISP [22] and JOSS [27].

Children are introduced to Smalltalk by first getting them to send messages to members of already existing classes such as the class *turtle*. In order to "make" a turtle (☺), a child types:

```
☺ ← turtle.!
```

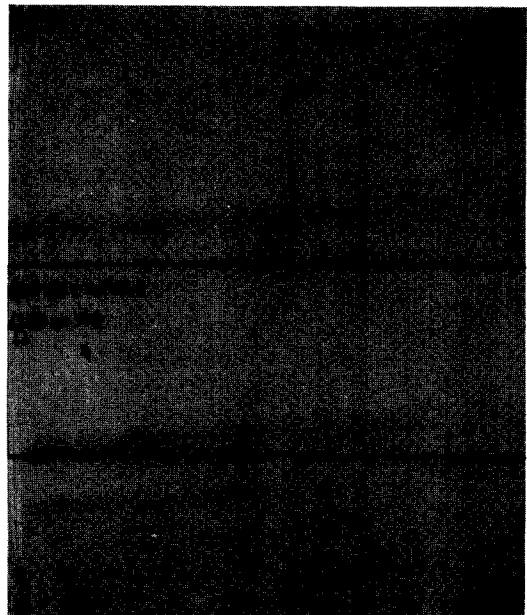
Here we are asking Smalltalk to create a new object, whose name is the single character "☺", as an example of a "turtle".

The child now types:

```
☺ go 50!
```

which says to the turtle (☺), forward (go) 50 units, do it! (!)

The child adds two numbers:



2 + 2!

and draws a square:

do 4(☉ go 100 turn 90.)!

☉ turn 90 says to the turtle ☉, turn right 90 units.

After each response, Smalltalk displays a representation of the Dynabook (☉) to let the user know it is listening. Thus, the above dialogue actually looks like:

☉ ☉ go 50!

☉ 2+2!

☉ do 4(☉ go 100 turn 90.)!

☉

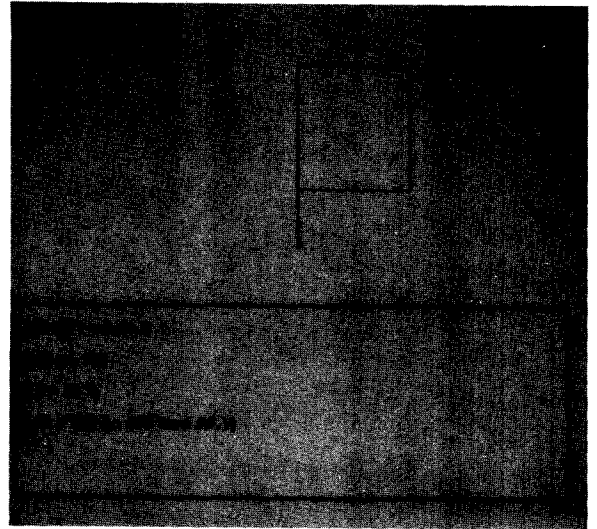
(The line and the square appear elsewhere on the display screen.)



The recipient of a message is always the first symbol in the message. Looking at it this way, ☉☉ go 50! is a message to Smalltalk consisting of ☉ go 50!. Smalltalk's sole job, however, is to act as postman for messages. It finds the turtle ☉ and informs ☉ that a message is waiting. ☉ looks at the message (go 50), understands it, and so draws the line. ☉ has nothing more to do so it tells Smalltalk, which, having nothing more to do, displays itself and waits for more messages.

The power of expression in Smalltalk comes from the ways that any user can extend the number of objects (including new *class* descriptions) which can interpret messages and cause new effects. For example:

☉ ☉x+50!


Here we are asking Smalltalk to associate two objects, the symbol *x* (quoted as a literal word because we mean *it*, not *what it may stand for*), and the number 50.





The children chose the pointing hand () to symbolize the idea of a literal word. We had been using double quote (") but they felt that something which seemed to point directly at the token made the idea more clear. They were able to replace (") with () using the font editor.

 x!




50 If we mention the name part of something we told Smalltalk to learn, it will give us the other; so 50 is given back.

 x + 45!
95


This also works in context. 50 was the "meaning" of x. With all the other possible numbers it belongs to a *class* which knows (among other things) how to add. So we are really sending 50 (a *number*) the message +45 which it knows how to do.

  y ← x + 45!
95

y is now associated with the number 95.

  square ← class (do 4 ( go 100 turn 90))!

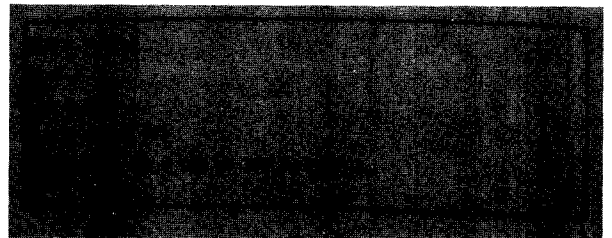
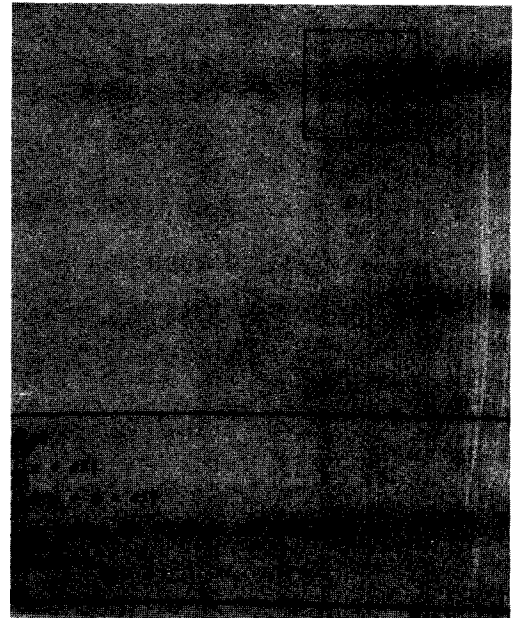
Here we are associating the symbol square with some actions to take. The symbol class is used to indicate that an *abstraction* is being associated with the name square. In an earlier version of Smalltalk, we adopted the Logo convention in which to indicates that the symbol following should be associated with some action:

to square (do 4 ( go 100 turn 90))!

 square!

As before, when the name of something we told Smalltalk to *learn* is mentioned, the other part is retrieved -- in this case some turtle messages to be *done* 4 times cause a square to be drawn.

When children are asked to look at the above definition to see what there is about it



that has to do with 'squareness' in general, they point to the 'turn right 90' (\odot turn 90), the 'do 4 times' (do 4(...)), and 'moving forward some distance' (\odot go...) not necessarily 100. A definition such as

```

👤  $\odot$  square+class size
    ( $\odot$  size+:.
    do 4
    ( $\odot$  go size turn 90))!

```

more fully captures the idea of 'square'. The notation means that 'square' will learn 'size' by receiving a message (\odot size+:.). The number associated with 'size' will then be used by the turtle (\odot go size) to determine just how far an edge should be drawn.

```
👤 square 50. square 100. square 150.!
```

```
👤 for n+1 to 200 do (square n.  $\odot$  turn 3.)!
```

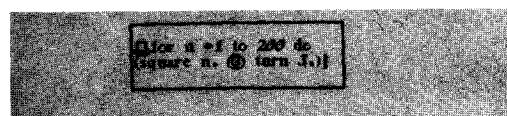
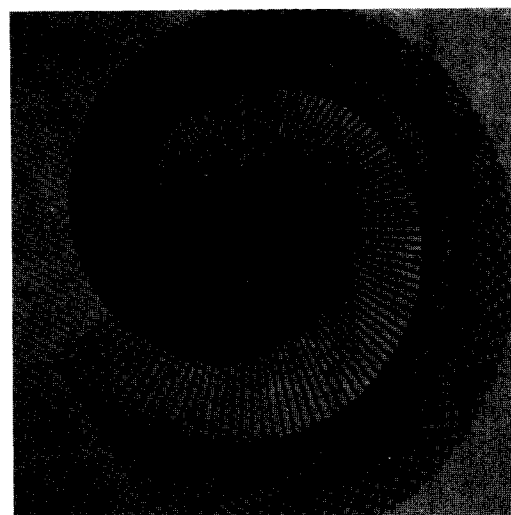
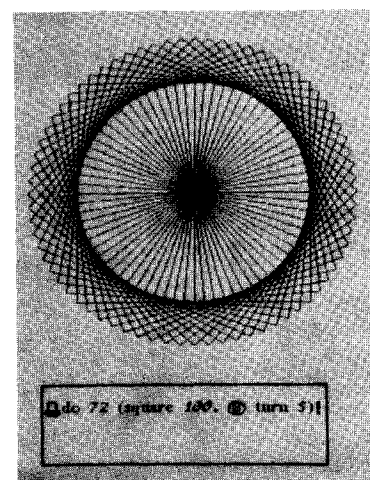
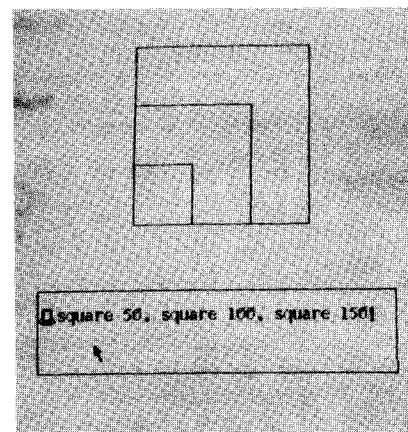
n is the number of repetitions done so far. The graphic result of the above messages is depicted to the right.

We have now created an object just like those which are already in Smalltalk. It has a name, square; it can be sent messages; it can receive messages and act according to their contents; and it can send messages of its own to aid production of desired effects.

This basic ability, to dynamically add new "recipes" to an already existing repertoire, is found in all so-called stored program machines. However, the way it is done and the intrinsic power of the particular act highly constrain the kinds of things that users actually attempt. Most programming languages can only talk about doing one thing at a time (such as the previous examples) and, unlike Smalltalk, find it very difficult to discuss and represent even such simple situations as patients flowing about in a hospital, kids in a school, trains on a track, spaceships in the sky, or bouncing balls in free space.

All of these examples illustrate an important epistemological idea -- grouping objects in common when they possess similar qualitative properties which differ only in quantity.

Classification into kinds or classes of objects that are generalizations of their properties is a popular idea in our history. In



these terms, humans are a *class* because they have common properties like language, tool using, physical appearance, and so on. Each individual person is an *instance* of the *class* human and has his own meanings for the shared properties, e.g., all humans have the property *eye color* but Sam's eyes are *blue*, Bertha's are *green*.

Smalltalk itself is built from *classes*. "number" is a *class*. Each individual number such as 2 or 17 is an *instance* of the *class* number. They differ only in their numerical value (which is their sole property). They all share a common definition of the different messages they can receive and send. "turtle" is a *class*; each instance shares the ability to draw lines, but has its *own* knowledge of its orientation and where it is located in the drawing area.

To teach kids to program in Smalltalk, we selected a series of projects that use, modify, and extend the definition of a class we named *box*. The kids are given a partial definition in which members of the class share the ability to draw and undraw, to turn, and to grow. The kids write programs to make boxes *dance* on the screen, spin around to draw designs, and play leapfrogs. They then modify the class definition in order to *teach* the members of the class *box* how to move around the display screen and to follow the stylus. Using *box* as a model, they invent their own class definitions, substituting the drawing of a square box to be any other shape: a circle, a rectangle, a spaceship, a box with a lid that opens up, a geometric design, and so on.

There are several ways to define the class *box*. One version has each member of the class retain knowledge of its size, its position on the screen, and the orientation (the tilt) of its drawing. An alternative definition, which we describe below, provides each member of the class with knowledge of an instance of the class *turtle*. The *turtle* instance remembers the proper orientation and location on the display screen. This definition might look like:


```

box ← class : size
(isnew ⇒ (turtle + turtle.
           size ← 50.
           SELF draw.)
draw ⇒ (do 4
        (go size turn 90))
undraw ⇒ (white.
          SELF draw. black)
grow ⇒ (SELF undraw.
        size ← size + :.
        SELF draw.)
turn ⇒ (SELF undraw.
        turn :.
        SELF draw.)
move ⇒ (SELF undraw.
        penup goto (:)(:).
        penon.
        SELF draw.)

```

The syntax for a conditional statement is:

```

question ⇒ (actions to take if question
            answered positively)
else do this action

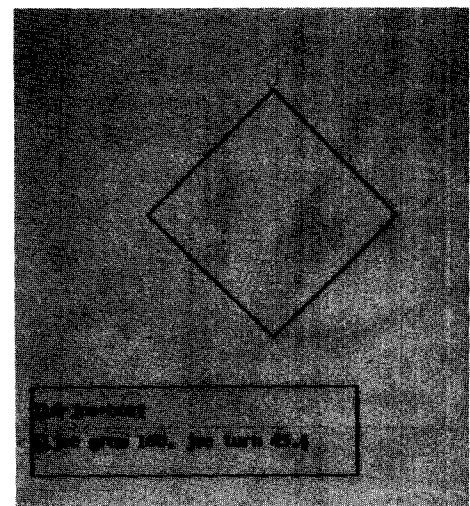
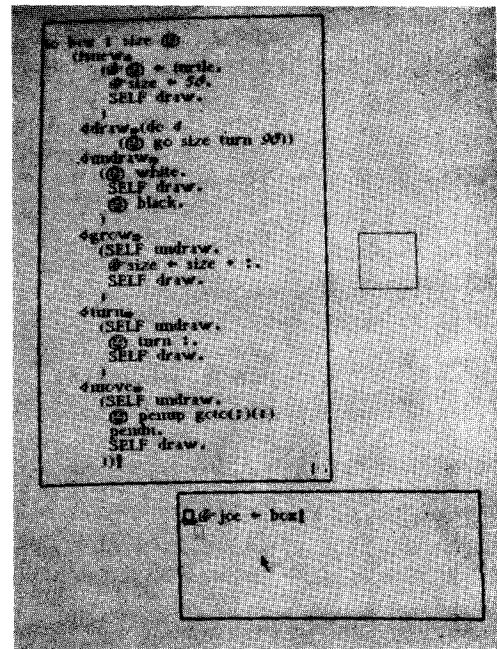
```

In the above definition, we are associating the symbol box with a set of actions. Each time a new member of the class is created, the question `isnew` is answered positively. Three actions are then taken: create a new turtle to draw this box instance, associate the symbol size with 50, and then send a message to draw the square on the screen.

With its eyes (◀), a box instance looks at any message it receives.

If it sees the message: then it carries out the actions listed below:

draw	The box has its turtle draw a square on the screen.
undraw	The turtle can draw with white or black ink. If we assume the background is white, then drawing with white ink is a way of erasing black marks.
grow	After erasing itself, the box instance retrieves a message which is interpreted as an increment to its size. It then redraws itself as a bigger or smaller square.
turn	To change the orientation of the box instance on the screen, the



turtle turns. The turtle retrieves a message which it interprets as the amount it should turn to the left or right.

move To change the position of the box instance, the turtle must goto new x and y coordinates. To avoid leaving a trace on the screen when it changes position, the turtle picks up its pen (penup) and then puts it down again (pendn) after moving (goto).

Hence, we could type:

☞ joe ← box !

Associate the symbol joe with a new instance of the class box.

joe grow 100. joe turn 45. !

Send a message to the box joe to grow 100 units and then to turn right 45 units.

We can then teach Smalltalk how to turn a box 10 units to the right and do it 72 times. We add a new message to the class definition:

☞ rotate ⇒ (do 72 (SELF turn 10))

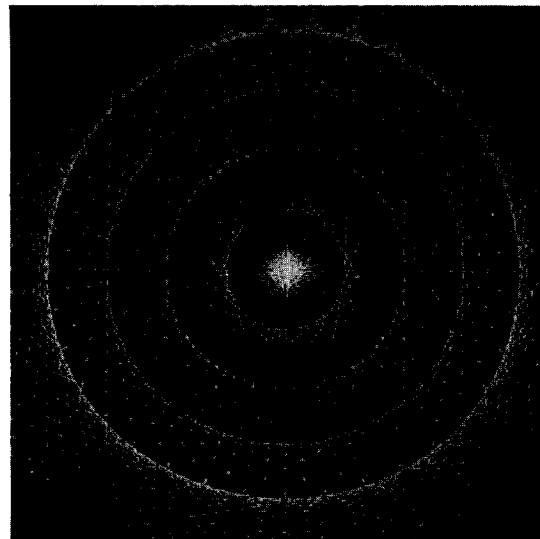
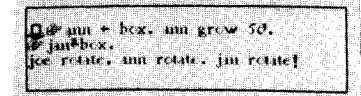
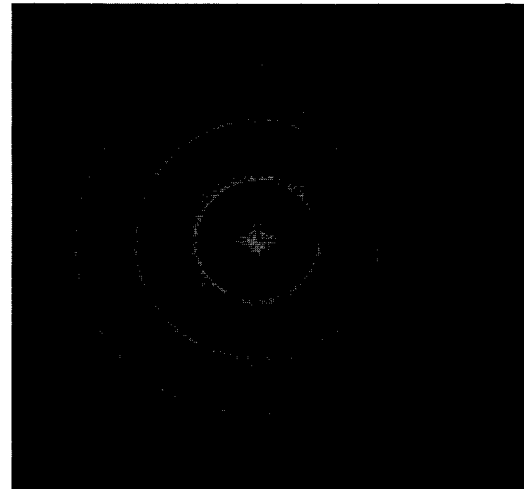
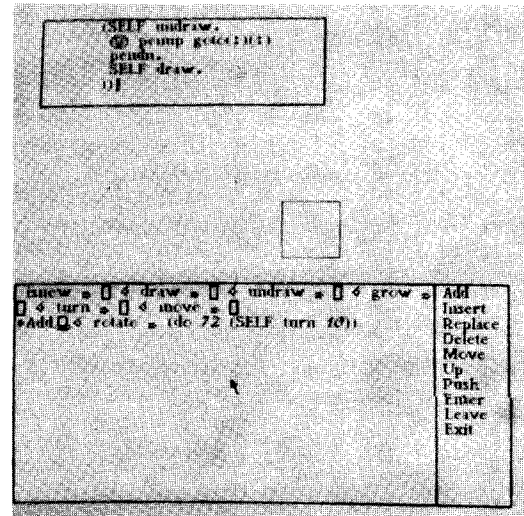
To draw the pictures shown to the right, we change the background of the display screen to black and tell Smalltalk:

☞ ann ← box. ann grow 50.

☞ jan ← box.

joe rotate. ann rotate. jan rotate. !

The last picture was made by sending messages to three different size boxes to rotate 15 units.



V. Smalltalk, Dynabooks, and Kids

We are in the business of model building, and one of the models we are trying to build is a way of giving people access to useful means and media for thinking about things. The model has to fit unanticipated notions. Does it? Can people comfortably use Smalltalk to model their real world and their imaginary ones?

We have been introducing children to the magical world of controllable media. The environment we set up in PARC contains a number of Dynabooks, a music keyboard, a turtle, and most importantly, the children who are learning about "ideas and reality" by designing and constructing tools of their own.

Example Smalltalk Programs

In this section, we present several examples of the kinds of programs kids have already designed. Each example is a modification and extension of the *box* class that was defined in the previous section; each was developed over a period of several weeks. These students met twice a week, with sessions lasting from 1-1/2 to 2 hours.

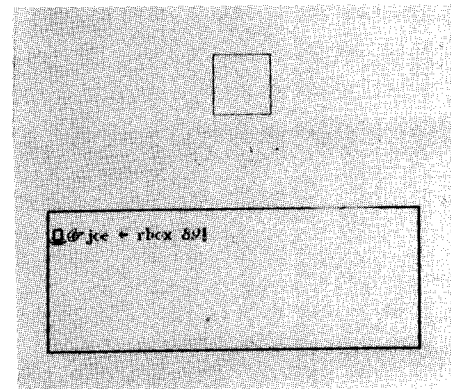
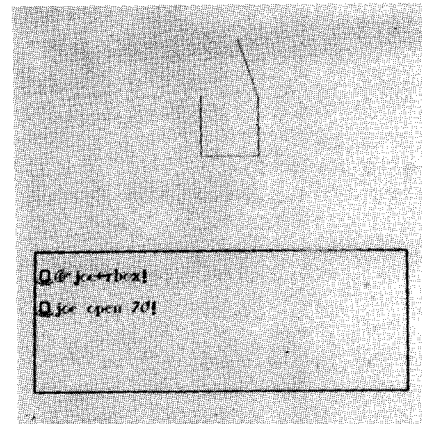
Lisa extended the *box* class definition, adding the ability to recognize the message *open*. The response to this message was to have the instance of the box open its lid (one side of the square) a specified amount.

```
☞ joe ← rbox. !
```

```
joe open 70. !
```

Dennis wrote a guessing game as a means for learning how to read characters from the typewriter keyboard. The object of his game is to choose a secret code number corresponding to a character on the keyboard (ASCII code); the player tries to guess the appropriate character by striking the keys. Lisa used this program, adding hints and the restriction that the player has only ten guesses, after which the player receives the correct answer. She then incorporated the game into her *box* class: each instance remembers a code number and responds to the message *guess* by starting the game. If the player guesses correctly, the box lid opens and a design is drawn.

```
☞ joe ← rbox 89. !
```



```
joe guess. !
too low
too high
you win!
```

The design is drawn by the turtle program design:

```
design ← class a b
  (a ← 1. b ← 0.
   do 100 (go b ← b+1. turn a))
```

The value that the turtle turns each time, a, is identical to the code number.

Lisa extended her class definition once more by adding the ability to point to a box on the screen. Now the player presses a button on the pointing device to indicate that the game should start; the box that finds the cursor inside its square area plays the guessing game.

Instances of the class box are square shaped. Susan defined several new classes, each a different shape, but each capable of drawing, undrawing, turning, growing, and moving. Her shapes included a class for triangles, rectangles, and circles. She then generalized these classes into a class for polygons: each instance remembers its position on the screen, its orientation, its size, and the number of sides it has. The message grow took on a new meaning:

```
ann ← shape 5 !
```

Ann is an instance of the class shape; it is a polygon with 5 sides.

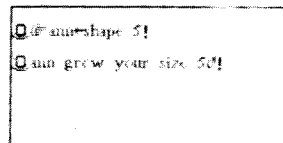
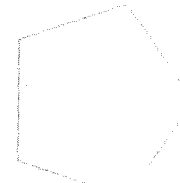
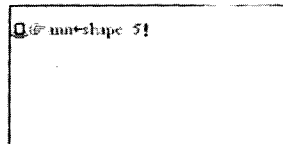
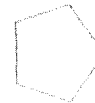
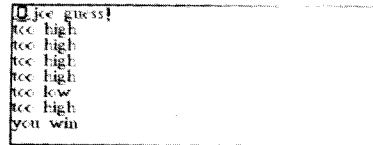
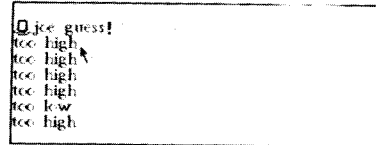
```
ann grow your size !
```

Ann increases the length of each of her five sides.



```
ann grow your sides 3 !
```

Ann is now an octagon. Susan's messages look more and more like English sentences.

Susan's then extend her class definitions to include a class menu. She has learned how to create Smalltalk windows (display frames), write words in the windows, and find out at which word the cursor is pointing. She uses the words to determine the messages to be sent to



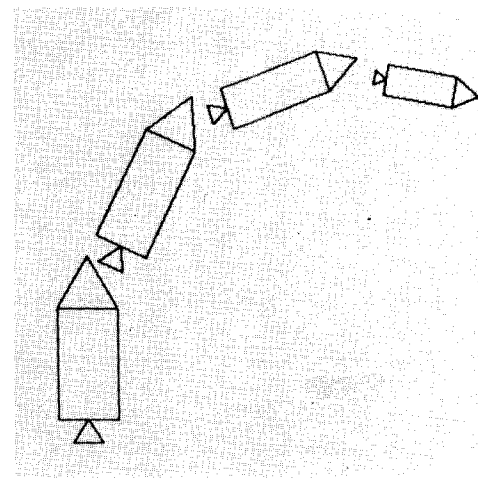
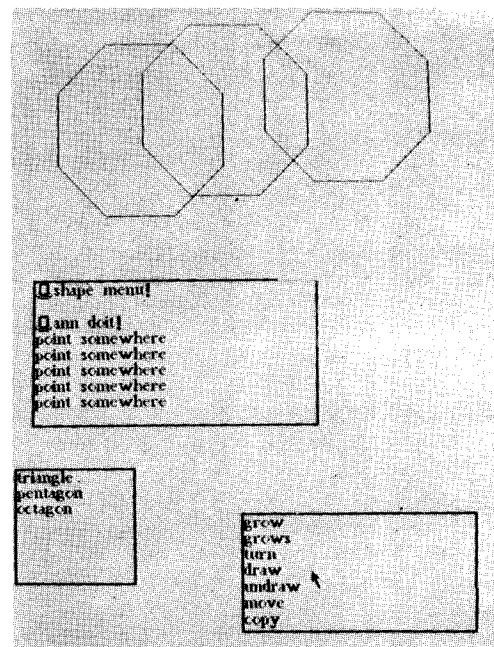
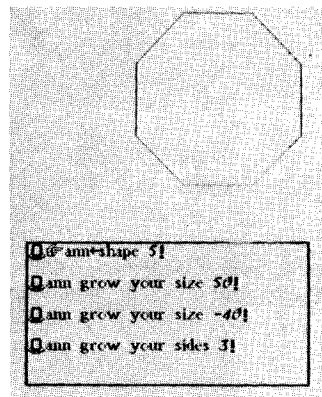
an instance of *shape*. The windows are instances of the class *menu*.

She is able to point to an instance that has already been created and then send it messages to grow (size or sides), turn, movie, delete, or copy itself (thus creating a new instance). Since each instance of *shape* owns its own turtle, , she can also send messages to change  *width* or *color* !!. A new instance of *shape* receives control from the *menu* by pointing to the word *select* and then pointing to the desired instance.

Kathy and Dennis were both interested in designing rocketships. Dennis wanted his ship to shoot torpedoes, while Kathy was interested in simulating rocket takeoff, ignition fire, travel, and landing. Kathy's rocketships were simple extensions of the *box* class in which the response to the message *draw* was to combine a rectangle and two triangles to form a ship with fire coming out one side. Each rocketship owns an instance of the class *turtle*; the turtle remembers the ship's position and orientation. To move the ship forwards or backwards, the turtle receives a message to go (+ or -) a specified distance. Hence, Kathy did not have to consider the complex problem of computing trigonometric functions, a level of mathematics with which she is not yet familiar.

Dennis has been inventing different versions of spacewar games. His first attempt involved two ships, one designated as "left", the other as "right". The left ship was controlled by striking keyboard keys a, s, w, z, and d; the right ship was controlled by keys ;, ', [, /, and l. The keys respectively mean: ship turn left, ship turn right, ship go forwards, ship go backwards, and ship shoot a torpedo. After learning how to test for inclusion in a square area, Dennis was able to "blow up" a ship if it was hit by a torpedo.

Initially, Dennis' war game required the two players to take turns striking keys; the player lost his turn if he hit a meaningless key. Winning depended on optimizing the ship's movement. Dennis also tried a version in which the players could hit the keys as fast as they pleased, and the control program, *war*, would try to respond to every key without alternating players. He then designed a new rocketship, instances of the class *treck*. *Treck* ships are peaceful ships that move in formation through space. Dennis' present goal is to use



kathy's rocket ships

the treck ships as "sitting ducks", under attack by the war ships.

In doing their projects, we note that these programmers had to understand division by negative numbers, testing inequalities, counting with increments, graphing, and testing for inclusion within an area of a polygon, as well as notions of classification and instantiation. They studied the differences between integer and decimal arithmetic, the application of conditional logic and sequencing operations, and coped with problems of computational context.

These four programmers are 12 years old; their teacher is thirteen.

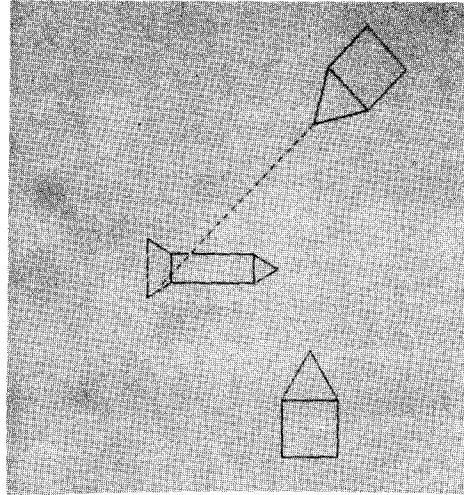
Outlined of Planned Projects

In the near future, we hope to expand and move our learning environment to a location in Palo Alto near schools and playgrounds. The major part of the day will be spent exploring with children, and pursuing structured experiments in visualization, abstraction, content, and curricula--much of it in conjunction with interested teachers from neighborhood schools, many of whom have already expressed great willingness to help. The center will be a community resource the rest of the day, which will allow us to gather information about casual use of the medium.

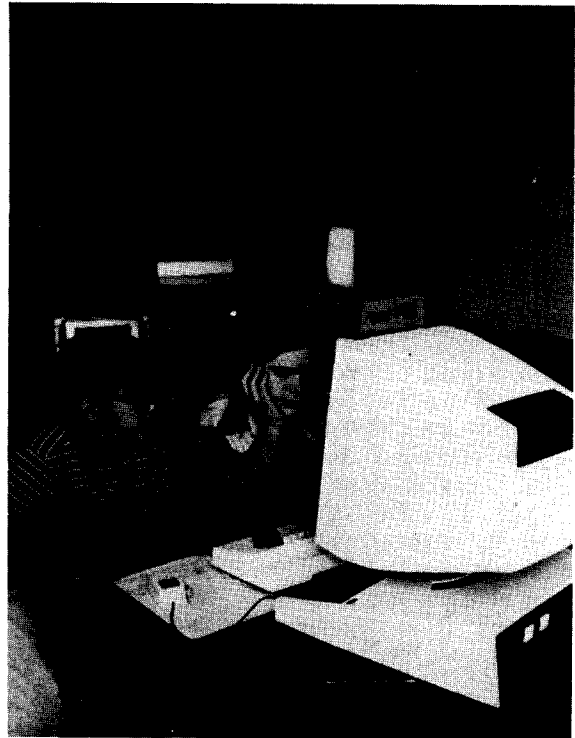
The initial design of our resource center is based on 12 Dynabooks. In addition to keyboard, display screen, and point device, peripheral equipment includes music keyboards, hi-fi amplifiers and speakers, touch-sensitive screen, a graphics printer, and an alphanumeric printer. We have also sought games, books, art materials, and mechanical devices that the children can use to enhance their explorations.

There are three major aspects of our resource center program.

1. We intend to provide an environment conducive to using and building *personalizable* tools for developing ideas in art and music, as well as in science and mathematics. The word "personalizable" is key to our conception of the Dynabook. Some of our efforts have concentrated on using the kinds of activities found in other computers-in-education projects, but in a truly personal learning environment--one in which the student is



space war by dennis



encouraged to extend, modify, generate, and test varied ideas about the content of newly acquired information.

2. We are interested in assessing the utility of such a resource within a residential community. Hence, we hope to do more than provide our own examples of the use of the Dynabook in and across subject areas such as English, art, drama, music, and mathematics. To this end, we will seek examples of the use of the Dynabook by visitors from the immediate community. Through this variety of users, we seek to broaden our experiences in the resource center.

3. Specific programs for testing the use of the Dynabook idea are planned. The first, that of teaching programming concepts for the purposes of providing a (computer) framework for problem-solving and for graphic design, has already been initiated [13,14]. Working with over 100 students, we have already devised, modified, and evaluated methods for teaching Smalltalk programming. Much of the information already gathered has stirred new ideas for input/output techniques, encouraged expanded applications of a personal information-retrieval system, and given us a measure of certainty about ways in which kids will be able to use our interim Dynabook.

We are currently creating two types of structured curriculums. The purpose of the first is to teach the user how to use to advantage the numerous building blocks we have developed. This curriculum is intended for the user who already has an objective in mind and is investigating the value of the Dynabook in pursuing it. The second curriculum comprises a program for developing thinking skills needed in learning to define activities ("wants or needs"). We broadly identify these skills as:

an ability to generate ideas that are appropriate to a given task constraint, an ability to vary one's ideas widely;

an ability to perceive old tools in new ways in order to make use of them for a new purpose;

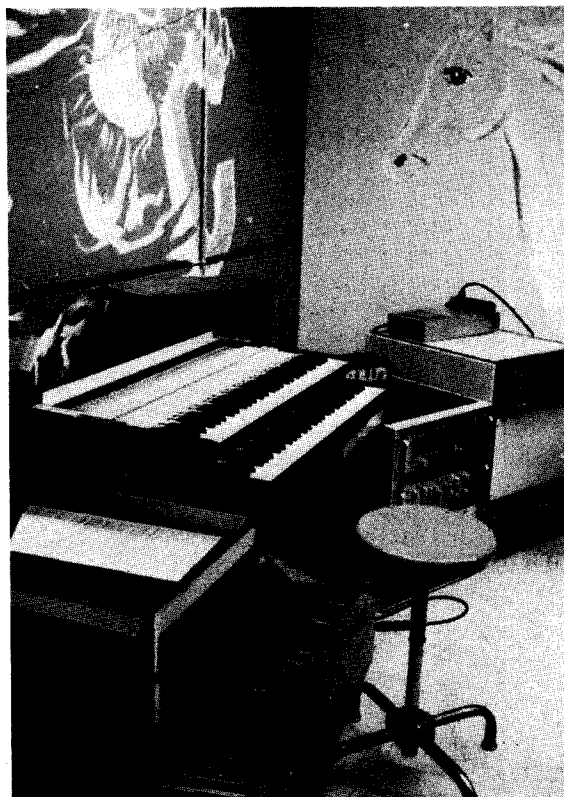
an ability to view ideas (or information) in different domains; and

an ability to generate activities that are personally interesting in the sense that they make daily tasks and leisure time more enjoyable.

Questions of transference are particularly



typewriter keyboard



organ keyboard

relevant in the context of this second curriculum. Many concepts, such as computational context and questions of control, are encountered in both programming and everyday activities. A trivial result of our research will almost certainly be that the children with whom we share our Dynabooks will know many, many things that their counterparts don't know. We are much more interested in whether different metaphorical structures are formed, whether the kids think more powerfully, more flexibly. This is the area where the ability to transfer deep ideas and techniques to new areas of interest should be qualitatively evident if there is anything to our suppositions at all.

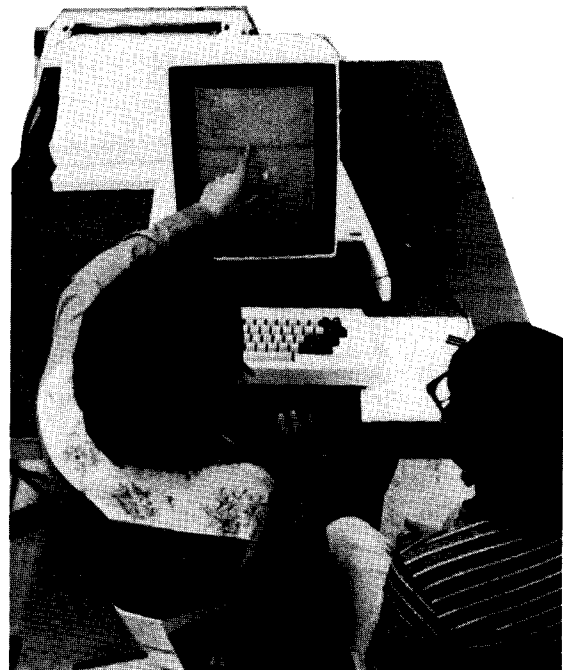
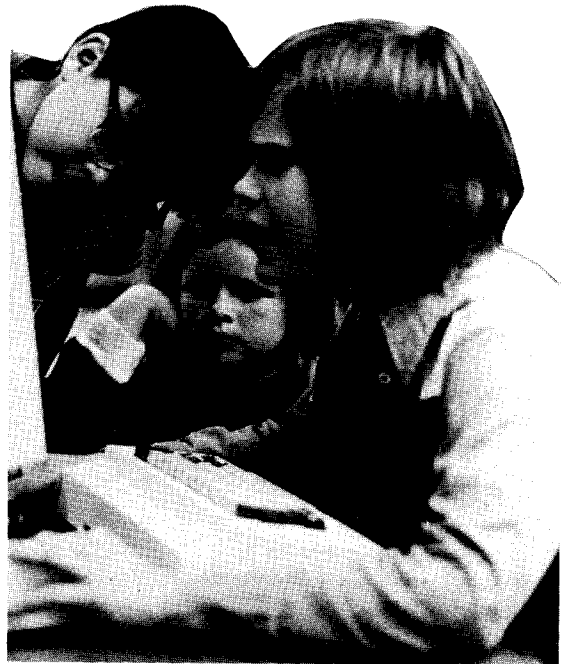
We use our computers in much the same way Covington and Crutchfield [6] use detective stories for creativity training, for getting a child to generate reasonable but novel ideas that can lead to a solution to a stated problem. Moreover, we can attempt to encourage a child to generate the very problems he or she tries to solve, thus fostering divergent thinking skills. A list of the kinds of *question asking* and *question generating* skills that we are nurturing appear in the following section entitled "Teaching Smalltalk". Also, some music and drawing skills, as well as adeptness at organizing material and specifying patterns of behavior, should be apparent in students who have spent time in our program.

We will be exploring the difficult task of helping students search for problems as well as solutions, using text and font editing, storage and retrieval of factual material, music synthesis, and drawing and painting, as well as Smalltalk programming. We will use these explorations to formulate the two curriculums mentioned earlier. There are three areas in which we anticipate exciting results and in which we will therefore concentrate our initial efforts at evaluation. They are detailed in sections:

1. Teaching Smalltalk
2. Plans for a Readiness Program
3. Experiments with Schools

1. Teaching Smalltalk

We are interested in teaching children of all age groups how to control sound and pictures through Smalltalk, and plan to learn a



great deal about their inherent abilities to abstract and generalize. How can these skills be aided through providing a suitable environment?

Why teach programming? Programming is an "activity-centered" learning experience. We view the ability to program as a new dimension for exercising good thinking habits; and for *exploring* tasks, in contrast to merely performing them. Smalltalk models provide a (manipulable) perspective on a programmer's view of his world; Smalltalk objects provide a consistent and useful means for investigating logical implications of classification schemes. When a student writes a program, he is free to try things his own way, as an expression of his creativity and individuality. The kids write programs to

- ...decode and encode messages,
- ...make up new language constructs,
- ...create and control graphic objects, and
- ...define their own (game) rules and strategies.

They compose music, play it back, relate it to programming and animation, and improvise on the keyboard to their own accompaniments. The kids are learning:

- real symbolic/mathematical thinking and concepts;
- that the physical world can be "captured" and understood by doing it themselves;
- that creativity and skill in balance are far more powerful than either apart;
- that style can be learned and improved through debugging and restructuring; and
- that *they can do things*--that they are not objects to be manipulated by the rest of the world but are *people* who can act on the world themselves.

What are we learning? In teaching a programming language, we are learning

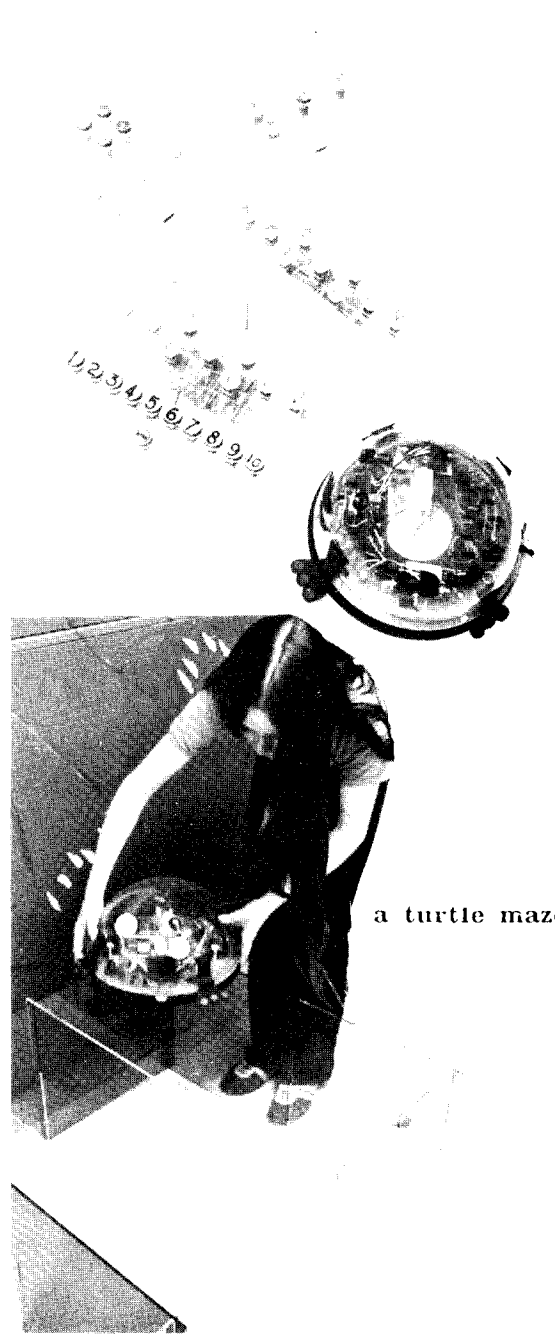
- ...how to present new concepts and ideas,
- ...how to observe what the kids are doing, and
- ...how to motivate kids to try new projects.

We are exploring ways to find out

- ...what a child knows,



button boxes and turtles



a turtle maze

...what kinds of structures are used to relate knowledge, and
...how the child learns to manipulate these structures.

In the long term, we will evaluate our teaching by examining:

- the design structures of the programs;
- the levels of abstraction which are used;
- the extent to which the kids use existing models and permute or extend them;
- the concepts learned as viewed from use of these concepts in programs;
- the manner in which the kids seek information for debugging or extending their programs;
- the extent of tool-building versus the combination of existing tools; and
- how well the effects of the causes are visualized, and how debugging techniques are used.

We capture the protocol (the sequence of keystrokes) of a user's session on the machine and examine it to determine what kinds of information are needed to find bugs and explore a program's structure. We ask the kids about the analogies they use and what their current picture is of what they are doing. This knowledge is helping us develop:

- a Smalltalk monitoring system (visualization package), placing major concern upon the visual presentation of the state of execution;
- a means for letting Smalltalk be self-explanatory, to increase the reality of viewing our students as autonomous learners; and
- some automatic "filter" programs which will allow us to focus on important changes in the student's style and approach.

Tutorial Dialogues with Smalltalk

We are experimenting with a variety of methods for capturing student-tutor interactions. Using only the stored protocols, we are unable to record the verbal assistance and sketches a human tutor makes on paper. We are presently making conventional audio and videotape recordings of selected student-tutor dialogues. More than fifty hours of videotaping, coupled with keystroke-by-keystroke records of student sessions on the Dynabooks, have provided us

invaluable information on (a) how the kids understand and use the concepts they are taught (in many cases, we capture one student's explanations to another, and learn, through their analogies, how the students view their work); (b) the kinds of questions and analogies tutors use that do or do not succeed in helping students (edited clips of this material will form part of our preservice tutor-training program); and (c) techniques of videotaping kids and computers. These recordings have aided us immeasurably in the evolution of our tutorial design.

Specifically we are asking the following questions:

What kinds of information do people seek in order to determine characteristics, features or bugs, of a descriptive definition? How does one learn to see aspects of a problem?

What is the nature of the representation of information that satisfies individual learning needs?

What kinds of questions can a person ask himself in order to direct his search for information?

How, through his own question asking, does a helper demonstrate productive problem solving behavior?

Can we provide a useful model of good question asking? Polya's [26] initial endeavors in this area represent an important starting point in developing such a model. We have to seek variations that rely on modes of exploration as well as individual background. What do we have to know, for example, about a person learning Smalltalk before we can provide him with a model that directs his viewing of his own work?

Finally, how does the helper determine that the learner has understood a particular concept well enough to terminate the current helper-learner interaction?

We are not primarily concerned with the implementation of a computer-based helper; rather, we want to determine the nature of the media *any* helper might use in communicating methods and aids. This information will naturally direct the design of an on-line browsing system, but might also provide some insights into the very arts of teaching and learning. We are building an on-line library of

our project ideas in which browsers can see what our Smalltalk objects do by viewing animated representations of the objects.

The kids themselves can think about their Smalltalk definitions by creating pictures and sounds depicting the messages each object understands and sends.

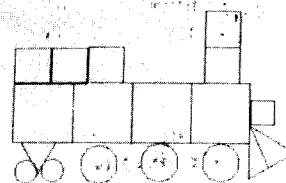
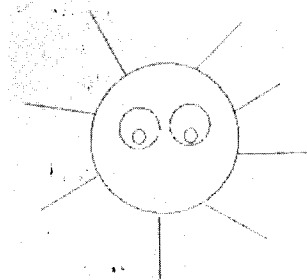
The Smalltalk *document* system will be a useful mechanism through which students can keep notes about what they are learning, cross-indexing on concepts they feel they have used, storing example programs they have completed, and commenting on any difficulties they have encountered. The students can then use the retrieval mechanisms to ask themselves questions about their own work, explore relationships between different programs, and investigate the usefulness of models and building blocks. Their own choices for key concepts will help them organize their work and focus on similarities and differences, while helping us learn how they understand what they have been doing.

Composing Pictures from Geometric Shapes

In observing kids constructing pictures with turtle geometry commands (Smalltalk simulation of the Logo line-drawing language), we noticed two main problem areas: (1) a difficulty in determining the subparts that should be written in order to construct the whole picture; and (2) a difficulty, once the parts are shown to them, to reconstruct the whole picture. Although these are symptomatic of conceptual programming problems, we observed that they may, in part, be visual problems. That is, the child views the picture as a whole and is unable to break it apart, or the child can not synthesize well enough to put the pieces back together. (A bit like humpty dumpty's soldiers.)

For example, a first problem in drawing a train is seeing the relationship between the rectangles, the circles and the total object; it is not necessarily which state transformations to use to piece the shapes together.

As Papert comments, one of the synthesis problems is a sequencing problem: all the parts are there but the order in which to glue them, as well as the glue itself, is not. In some cases, the child is able to put together very simple pictures but finds it too difficult to



pictures made from geometric shapes

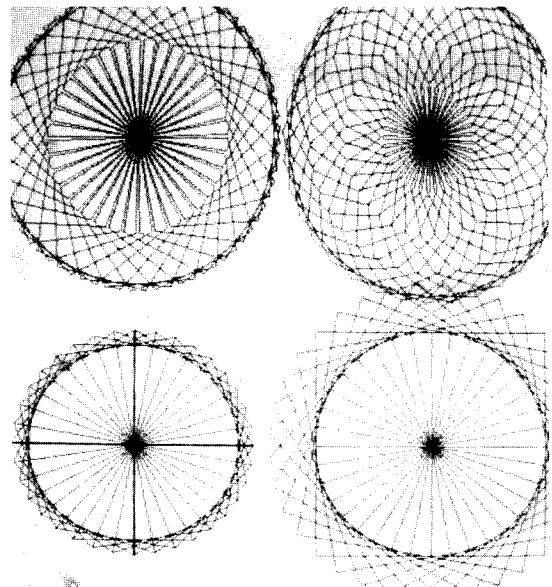
make the same picture on a smaller scale, to repeat pictures on the screen, or to change the picture slightly by simple changes (such as replacing an equilateral triangle by an isosceles triangle, or a square by a rectangle).

The problems the children demonstrated are not necessarily solved by more programming examples. The problems likely show up in non-programming work the student does: copying from a blackboard to paper at his desk, moving his eyes across a page, making maps or constructions in art class. Walk through a day with a ten year-old child and you will see a lot of cutting, pasting, drawing and moving of geometric shapes. Some require perceptual skills the child has not yet learned and is busily figuring out; most require some imagination on the part of the child; all require transferring information from one space to another, from one medium to another.

A set of Smalltalk programs were designed to help a child use a display and pointing device to manipulate simple geometric objects: to make pictures, to invent doodles; and to both compose and decompose these combinations. A child can construct complex pictures by scaling, rotating and copying simple geometric objects. Simple Smalltalk programs generate designs from basic geometric shapes by variations on repeating, rotating, scaling, superimposing, placing, and combining drawings of these shapes.

This work with picture construction is, admittedly, a predecessor to learning iconic programming. By "iconic programming" we mean the specification of a sequence of instructions by pictorial, non-alphanumeric symbols. Simple examples were implemented on the PLATO system in such lessons as Moveman and in a language under development called PAL [2]. The Moveman idea is borrowed from the movement of Papert's turtle; it is limited in terms of the kinds of things the single icon can do and in terms of the user's ability to modify it. Although not described as such, we view PAL as a potentially useful system for specifying animation sequences, mainly limited by the user's inability to draw his or her own icons and to share these icons with friends.

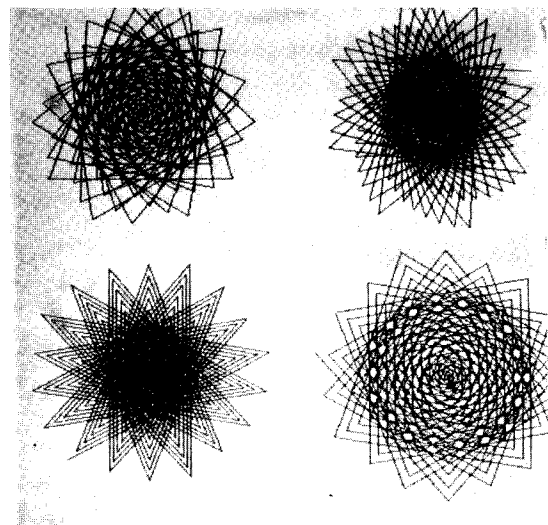
These observations of limitations of other programs reflect the very high premium we place on flexibility and modifiability. Providing a variety of carefully selected



```

point 381 381.
polys 4 100.
point 128 381.
polys 3 100.
point 128 128.
polys 5 75.
point 381 128.
polys 10 40.
    
```

by Tom, age 10



```

point mx my!
see 0 110!
point mx my. see 0 134!
point mx my. see 0 140!
point mx my. see 0 100!
    
```

by Scott, age 9

examples, in which rules are used and strategies applied, undoubtedly reinforces the use of the rules and strategies. We expect that making it possible to formulate and test hypotheses and counterexamples will extend the user's understanding of their applicability and limits. Brown's work on SOPHIE [3], a system for learning about electronic trouble shooting, is a unique example of the application of these ideas. Our instantiations on this idea include helping the user to restructure the control mechanisms (of, for example, a program for constructing pictures from shapes), to redesign graphic layouts, to replace and append to objects, to delete, modify, or add to (game and simulation) rules and strategies, and to reverse playing roles.

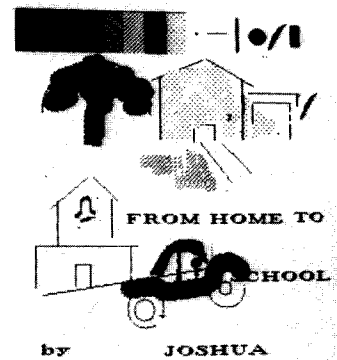
2. Plans for a Readiness Program

We believe that every young child is an *exceptional* child. And we are convinced that children will not fail in school if they are properly prepared for academic study *before* they are expected to perform.

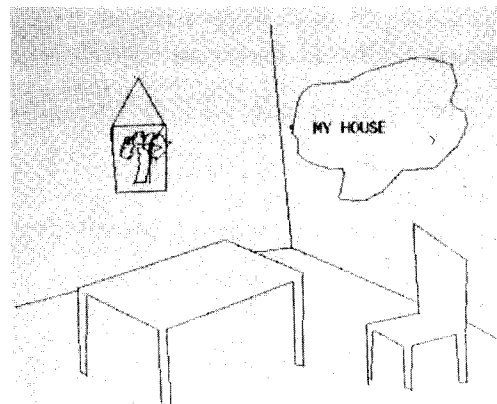
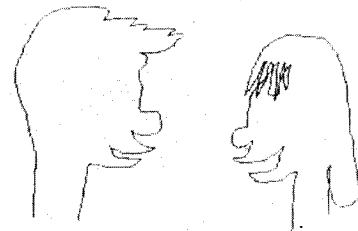
Our approach is basically to motivate a child to learn to read and to write by concentrating on his or her learning to communicate, to ask questions, tell stories, and acquire information. A child learns arithmetic because it can help him do things that bring him enjoyment. For instance, children we have worked with can plot on graphs and use the binary number system, and create classes and instances of those classes. We suspect this is at least partly because these abilities make possible graphic and other effects that are attractive to these children. They are learning to explore uses and implications of their own designs.

Exploring, probing for detail, and generating ideas characterize a thoughtful person; they are characteristics that we think can and should be acquired early.

We are interested in developing thinking games (written in Smalltalk) for primary school-aged children who will be able to play and modify them, as well as write their own. Coupled with materials we mention below and those developed by other researchers, these games will make up a special program for a group of five year-old children. We plan to supplement computing activities with a variety



making story books



and coloring books

of kinesthetic materials. During this age, the basis for many concepts and abilities are initially learned through manipulation of physical objects. The reader will note that our choice of activities reflects a Piagetian epistemology (frame of reference). We have delineated activities for different stages; the activities outlined in this part concentrate on sensory-motor development for primary ages.

Thinking Games for Primary-aged Children

Among the published collections of activities for children of various ages, we have found several that are extendable into Dynabook-based activities. They include *Thinking Goes to School* by Hans Furth and Harry Wachs [12], which provides a program for children in kindergarten through third grade. The activities described in this book are designed to "develop the child's thinking ability [through] experience with the acts of thinking".

At the fourth-fifth grade level, *Workjobs*, by Mary Lorton [21], is a collection of manipulative activities, each built around a single concept of perception, matching, classification, sounds and letters, sets, number sequences, combining and separating groups, or relationships. The activities use homemade materials that fit any school's budget: cardboard carpentry, dime-store paraphernalia, and household memorabilia. Another source is the *Productive Thinking Program* of Covington and Crutchfield [6], the result of research into training children to yield enhanced performance on Torrance tests of creative thinking. We borrow freely from these books in presenting the Dynabook as a medium for early experiences that develop a child's thinking habits.

General Movement Thinking, conscious control of one's body, is basic to a child's ability to move his eyes across a page or from blackboard to desk, and to hold a pencil or brush without strain. Conventional media for stimulating general movement thinking include a gym mattress for transporting and positioning the body into swimming, crawling, wheelbarrowing, and bicycling motions; a balance board and a walking rail for coordinating the head, limbs, and torso; and a few balls for juggling and playing catch.

We add a Dynabook for animating drawings of human body movement, of juggling, hopping,

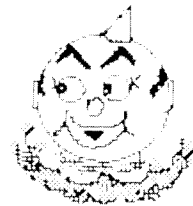
skipping, bouncing, balancing, throwing, and catching. Animation is a fun way to study sequencing movements, classifying movements, and judging spatial relationships between body parts. Taking advantage of the ability to superimpose movies, we can attribute these movements to endless varieties of objects. (One example is the tossed-ball animation shown on the next three pages).

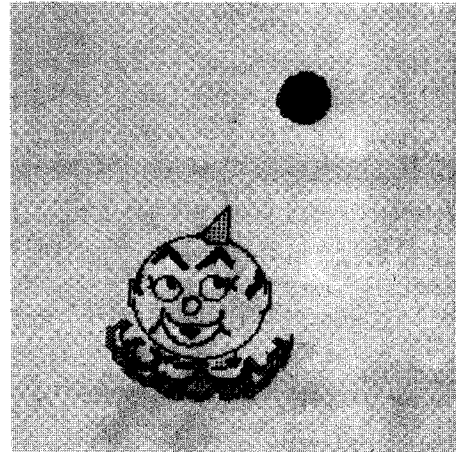
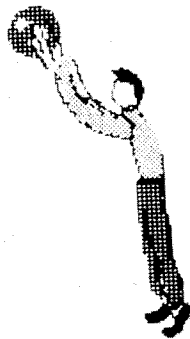
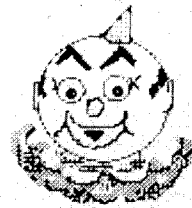
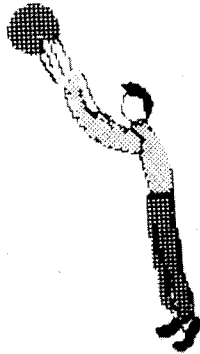
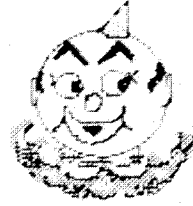
Discriminative Movement Thinking is typically practiced with eye tracking games devised from such devices as: a turntable, marbles, and string, finger paint sets and clay. These concentrate on learning to control one's finger movements. The Dynabook can help a child practice hand-eye control by combining a stylus for pointing with graphics for games of tracing through a maze or tracking animated objects, and finding hidden shapes in geometric designs. We also add the Dynabook with a musical keyboard. It is a fun way to learn to coordinate movement with sound.

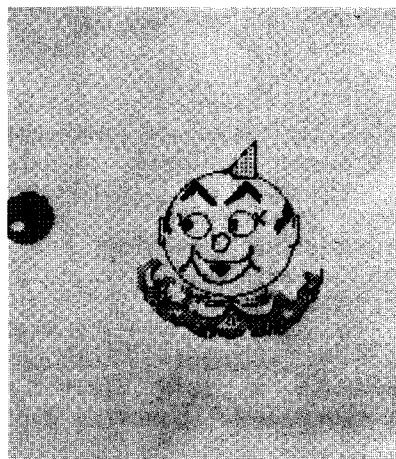
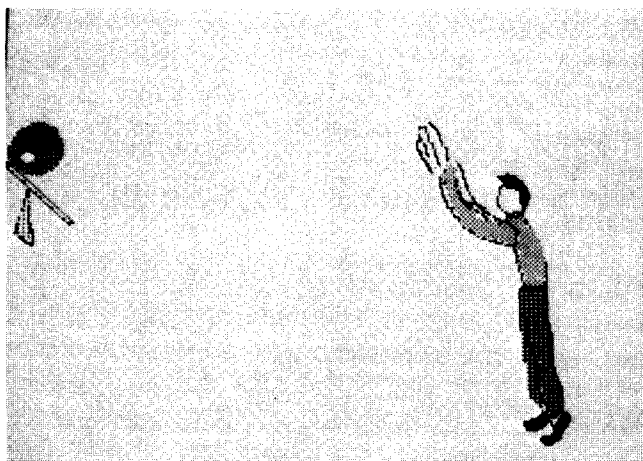
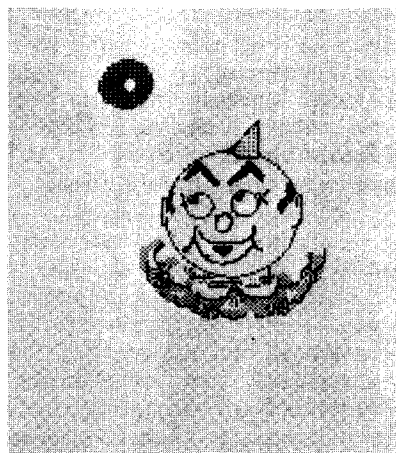
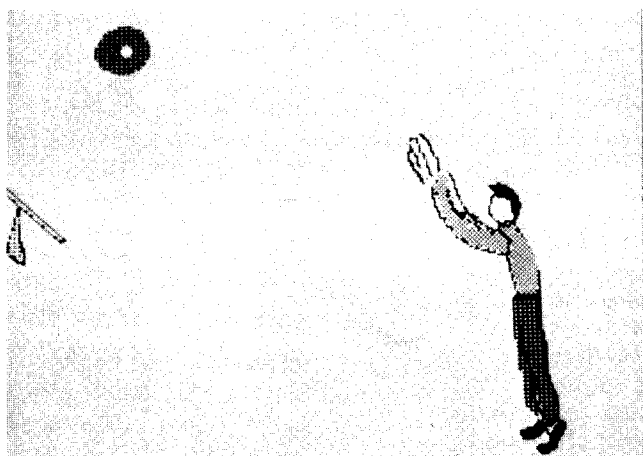
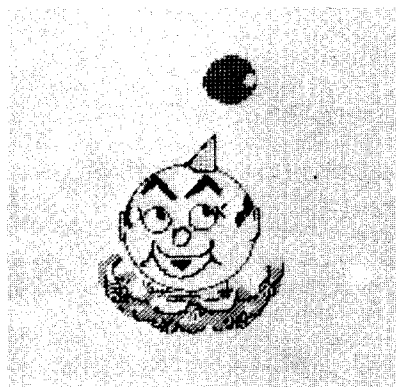
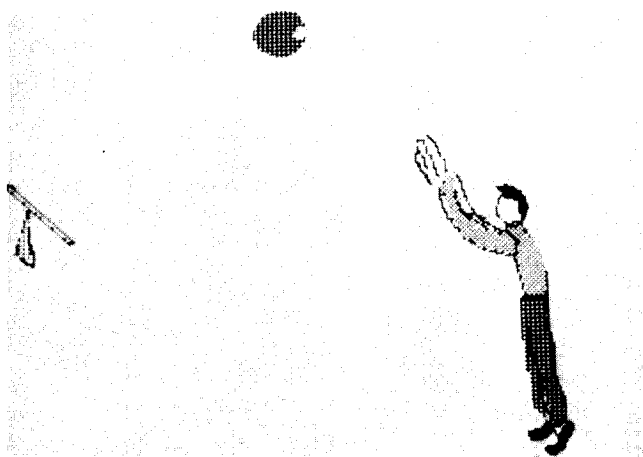
Superimposing Movies

The sequence on the left consists of three movies: a tossed ball, a backboard, and a basketball player.

The sequence on the right consists of two movies: the same tossed ball movie, and a clown with rolling eyes.







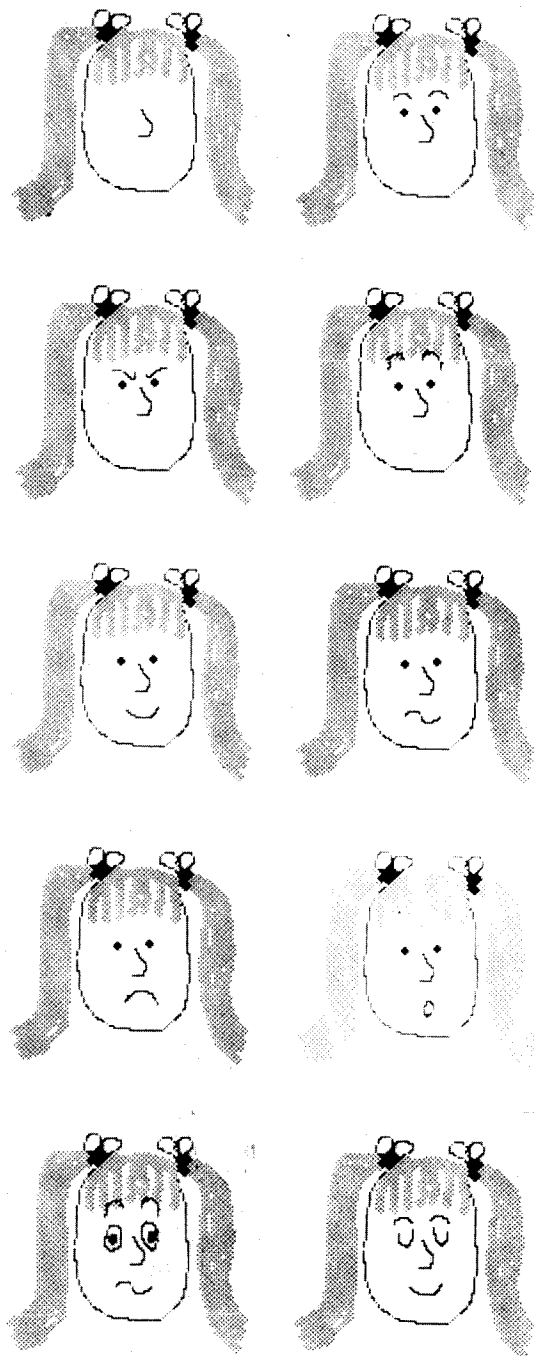
Perceptual Thinking--visual, graphic, and auditory--involves decoding, encoding and interpreting sensory data. Conventional educational games use: parquetry blocks for determining similarities and differences between shapes; a pegboard and pegs, tackboard and tacks, for copying shapes and connecting dots and for displaying designs and shapes to copy; and jigsaw puzzles for filling in outlines, with and without directing lines.

Such activities serve as a bridge to the more abstract world of the Dynabook, a world for creating numerous variations on these games. With the Dynabook, children can construct designs from basic geometric shapes, arrange, permute, overlap and match shapes and pictures. They draw pictures, imagining how a scene would look from various spatial or social perspectives; and compose pictorial stories for friends. These static activities can come alive by integrating the pictures with movement and sound. Children can also investigate the absurd by removing parts of pictures or completing partial scenes, making greeting cards, cartoons, and story books (The series entitled *The Metaphorical Way* [15] offers extensive applications of this idea.)

For example, our version of the PLATO game *faces*, shown to the right, can be done by a child. The child paints his own choice of eyes, eyebrows, nose, and mouth, or selects from drawings stored by his friends. Moreover, the quiet face can now move, changing the game from one of forming facial shapes, to one of investigating facial expressions. The example superimposes a movie of eyebrows with that of a mouth, over a face movie. The 3-frame eyebrow movie interacts with the 4-frame mouth movie to give us effectively 12 different facial expressions. The game takes on a livelier nature as the child adds yet another movie--two frames depicting blinking eyes.

The display screen is a new medium for studying laws of conservation and invariance through symbolic models. Visual thinking games emphasize concepts of parts-whole and time perception. Exercises in copying, completing, removing, and transforming pictures and music, develop a child's skills in nonverbal communication.

In *Logical Thinking*, knowing how to execute a rule correctly is one level of performance. Knowing how to explore a system



of rules, and even to specify these rules, requires, perhaps, a higher level. Perhaps here the Dynabook is uniquely suited. Children can construct or modify games that require invention and testing of classification schemes, exploration of the relationship between a thing and its class, between a thing and its name and a class and its name; they can find new ways of ordering, permuting, and cross-classifying.

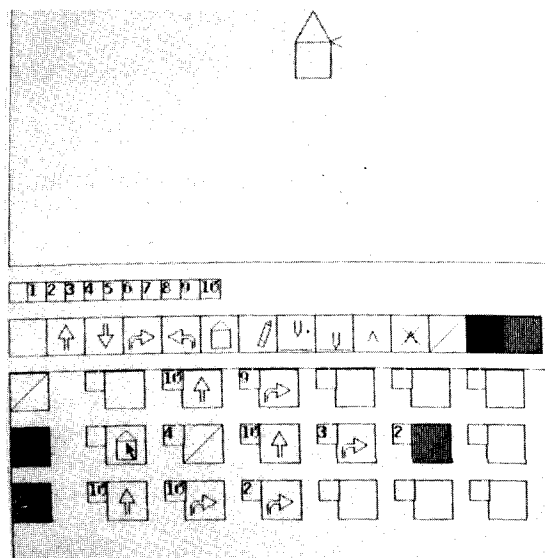
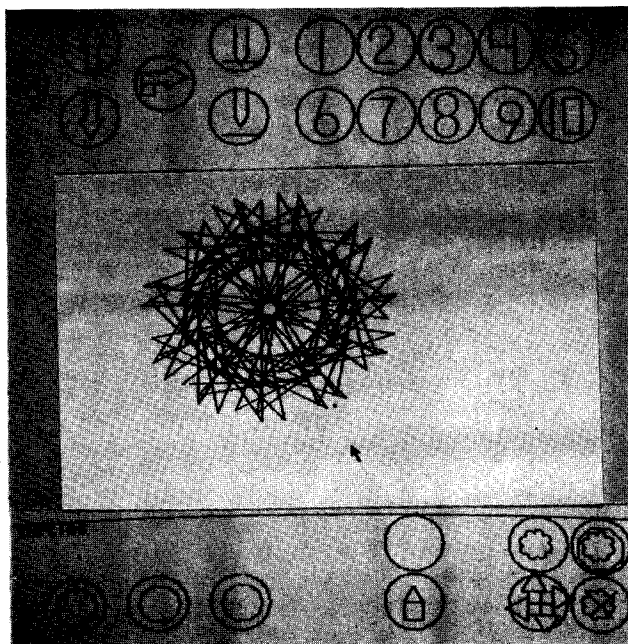
Button Boxes and Turtles

The MIT Logo project recently extended the range of usefulness of their mechanical turtle [1] with the invention of a button box control device. Young children push iconically-labelled buttons to move the turtle around the floor, making the turtle draw simple pictures with its built-in pen, toot its horn and blink its light during the trip.

Using the button box in this form, our goal is to provide an attention-grabbing system in which the child can explore the notion of a *sequence of actions* causing a pictorial effect. The goal in using this device is to help very young children (a) visualize the concept of *sequencing*, and (b) stretch their capacity for attending to a single task. We have observed that the medium is a new context for thinking about direction relative to an individual object.

We have written Smalltalk simulations of the button box and other iconic languages that allow easy editing of command sequences and permit manipulation of a class of objects (for example, a class of trains). We think of these Smalltalk simulations as "weaning" activities to introduce the notion of a sequence of tasks producing abstract results.

We have been using these non-alphanumeric systems with children between the ages of 4 and 7. They use, first, the actual button box and then the simulations. Our goal is simple: to determine which of the above concepts we can teach using a combination of mechanical "toys" and Smalltalk simulations.



button box simulations

3. Experiments with Schools

We have already begun projects in cooperation with several local schools. These projects will continue with efforts concentrated on tying our computer work together with the kids' classroom activities. The following two experiments make use of our text editing, font editing, and animation capabilities; while the third investigates the impact graphic feedback has on a child's decision-making processes.

English Classes for Junior High and High Schools

We already know, informally, that 12- to 13-year-old kids enjoy using a computer text editor for preparing book reports and their own stories and compositions. Some teachers place sufficient importance on the form of these reports that the students receive two grades, one for form and one for content. We have no doubt that grades on form would increase if reports could be prepared with a good text editor. But why not the content too? Why not provide a way for students to create form and let form *help* them express their ideas?

Using a text and font editor, it is easy to

- ...insert, delete, and modify characters, words and sentences, and
- ...move these text elements around the page.

We can also

- ...mix fonts,
- ...design fonts,
- ...use the multiple window capability to easily compare and contrast non-contiguous text selections, and
- ...mix pictures, sketched on the computer, with the text.

So the student can quickly and easily try different presentations of his ideas, adding and deleting information as these changes seem appropriate, even changing the form of individual characters, and then obtain a well-formatted hard copy for his teacher. When kids are graded on form, it is not surprising that they hesitate to change anything once their paper is typed. Yet it is from neatly typed text that we can best see what the completed paper says.

Of the many efforts devoted to improving writing skills, some have exploited the technique of dissemination of student work in the form of class journals and newspapers as a vehicle for practicing communication skills and for encouraging students to apply evaluative standards to their work. Ross Quillian at the University of California, Irvine, California has college students contribute to "journals" refereed by their classmates; the elementary-level students of Moore [23] publish a weekly newspaper. Preparation of such publications enhances the nature of the student's own evaluation of his writing because the material must be judged with respect to the context in which it will be published. We can support students in the production of a variety of documents, from private papers to undertakings with considerable distribution.

High Schools

We plan to explore the role of a powerful computing medium in high school art, music, creative writing, design, and drama groups. Using Smalltalk, the students can explore the concept of communication: how can objects communicate with one another, affect one another, interact or cooperate with one another? Constructing Smalltalk simulations develops an understanding of how to

- ...select pertinent information,
- ...assign ownership of that information,
- ...communicate among owners to exchange information, studying computational context, and
- ...investigate scheduling and control mechanisms.

Of the several efforts at introducing computers at the high school level, one of the more advanced is the Soloworks math program at the University of Pittsburgh [9]. It is an effort to develop a high school math program that combines access to a time-shared computer and terminals (the available language is Basic), a flight simulator, and a variety of audio/visual materials. In teaching programming, the Soloworks group has adopted the project approach we have emphasized: facilitating the development by the students of new and unique extensions of concepts and facts they have been taught. We expect our efforts at this level to both parallel and learn from Soloworks' results

in teacher training and teacher involvement, sample projects, and evaluation methods (which are not unlike those we previously described).

Spaceship Simulation Project

Another project we are exploring involves kids in a local elementary school's fifth-grade class. They are studying reading, writing, and arithmetic in the context of a simulation of a huge spaceship. The kids use a large mock-up of a ship, design the form of government that will control the lives of 3000 space citizens on a year-long journey, and make significant socio-economic decisions that affect life aboard a ship traveling in outer space. This year the kids will use Dynabooks in their decision-making activities. They will be able to design a model ship on a graphics display and will be able to simulate space travel and space adventures. We are interested in determining the effect graphic feedback from a simulation of their projected trip will have on their design and planning decisions.

VI. Summary

What would happen in a world in which everyone has a Dynabook? We have suggested that the quality of the relationship between learner and subject would be drastically changed, that the use of a portable library would make the world an interactive, active learning center. The Dynabook currently is too large to be portable, but it will allow us to begin investigating its role in improving the quality of human expression through its use in a resource center.

We have stated several specific goals. In summary, they are

1. to provide coherent, powerful examples of the use of the Dynabook in and across subject areas:
 - a. to develop readiness programs for young children;
 - b. to help high schoolers develop new metaphors for thinking about problems, and for communicating and expanding their ideas;
2. to study how the Dynabook can be used to help expand a person's visual and auditory skills;
3. to provide exceptional freedom of access so kids can spend a lot of time probing for details, searching for a personal key to understanding processes they use daily; and
4. to study the unanticipated use of the Dynabook and Smalltalk by children in all age groups.

VII. Acknowledgements

Many people (both from the *Learning Research Group* and from other groups at PARC) have worked hard to help make our dreams a reality. We have attempted to list below the names of these people.

From the Learning Research Group

Permanent Staff

Adele Goldberg
Dan Ingalls
Chris Jeffers
Alan Kay
Diana Merry
John Shoch
Dick Shoup

Visitors

Ron Baecker
Dennis Burke (age 12)
Barbara Deutsch
Marian Goldeen (age 13)
Susan Hammett (age 12)
Bruce Horn (age 15)
Tom Horsley
Lisa Jack (age 12)
Ted Kaehler
Kathy Mansfield (age 12)
Eric Martin
Steve Purcell
Dave Robson
Steve Saunders
Bob Shur
Dave C. Smith
Bonnie Tenenbaum
Steve Weyer

Radia Perlman
From Other Groups at PARC

Patrick Baudelaire
Dave Boggs
Bill Bowman
Jim Cucinitti
Peter Deutsch
Bill English
Bob Flegal
Ralph Kimball
Bob Metcalfe
Ed McCreight
Mike Overton
Alvy Ray Smith
Bob Sproull
Larry Tesler
Chuck Thacker
Truett Thach

VIII. References

- [1] from General Turtle Inc., Cambridge, Mass.
- [2] We reference several publications on PLATO:
- Alpert, D. and D. Bitzer, *Advances in Computer-based Education, Science*, 167, 1582, 1970.
- Bitzer, D. *Computer-assisted Education, McGraw-Hill Yearbook of Science and Technology*, February, 1973.
- Ghesquiere, J., C. Davis, C. Thompson, *Introduction to TUTOR, PLATO Service Organization, CERL University of Illinois*, 1974.
- Kammerahl, Hanna, *PAL: Picture Algorithm Language*, working draft of doctoral dissertation for Computer Science department, University of Illinois, Urbana, Illinois.
- [3] Brown, John S., Richard Burton, and Alan Bell, *SOPHIE: Sophisticated Instructional Environment for Teaching Electronic Troubleshooting*, BBN Report No. 2790, Final Report, March 1, 1974.
- [4] Bunderson, C. Victor, *TICCIT Project: Design Strategy for Educational Innovation*, Brigham Young University, Provo, Utah, ICUE Technical Report No. 4, September, 1973.
- [5] *Burroughs B5500 Information Processing System Reference Manual*, Detroit: Burroughs Corporation, 1964.
- [6] Covington, M. V., R. S. Crutchfield, L. Davies, and R. M. Olton, *The Productive Thinking Program*, Columbus: Merrill, 1972.
- [7] Dahl, Ole-Johan, and Kristen Nygaard, *SIMULA--an ALGOL-Based Simulation Language, CACM, IX, 9*, September, 1966, pp 671-678
- [8] Dahl, Ole-Johan, Bjorn Myhrhaug, and Kristen Nygaard, *SIMULA--Common Base Language*, Norwegian Computing Center, Oslo, Norway, 1970.
- [9] Dwyer, T.A., *Heuristic Strategies for using Computers to Enrich Education*, in the *International Journal of Man-Machine Studies*, Vol. 6, pp. 137-154, 1974.
- [10] Feurzeig, W., et al. *Programming-Languages as a Conceptual Framework for Teaching Mathematics*, Final Report on BBN Logo Project, June 30, 1971.
- [11] Fisher, D. A., *Control Structures for Programming Languages*, doctoral dissertation, Carnegie-Mellon University, May 1970.
- [12] Furth, Hans, and Harry Wachs, *Thinking Goes to School*, New York: Oxford University Press, 1974.
- [13] Goldberg, Adele, *Smalltalk and kids--commentaries*. PARC-LRG-3, June, 1974.
- [14] - , *Classroom Communication Media, ACM SIGCUE TOPICS in Instructional Computing*, Vol 1, Teacher Education, January, 1975, (with Bonnie Tenenbaum).

- [15] Gorden, W.J.J., *The Metaphorical Way of Learning and Knowing*, Cambridge: Porpoise Books, 1971.
- [16] Kay, Alan, *FLEX: an extensible simulation language which can be directly executed by computer*. University of Utah Note, September, 1967, University of Utah, ARPA Project, Salt Lake City.
- [17] - , *FLEX: a flexible extensible language*. M.S. Thesis. University of Utah, May, 1968 (University Microfilms).
- [18] - , *The reactive engine*. doctoral dissertation, University of Utah, September, 1969 (University Microfilms).
- [19] - , *A personal computer for children of all ages*. *Proceedings of the ACM National Conference*, August 1972, Boston.
- [20] - , *A dynamic medium for creative thought*. *Proceedings of National Council of Teachers of English Conference*, November, 1972, Minneapolis.
- [21] Lorton, Mary, *Workjobs*, Menlo Park: Addison-Wesley, 1972.
- [22] McCarthy, John, P. Abrahams, D.J. Edwards, T.P. Hart, and M.I. Levin, *LISP 1.5 Programmer's Manual*, Cambridge: MIT Press, 1962.
- [23] Moore, O.K., and A.R. Anderson, *Some Principles for the Design of Clarifying Educational Environments*, in Goslin, David (ed), *Handbook of Socialization Theory and Research*, N.Y.: Rand McNally, 1969.
- [24] Papert, S., *Teaching Children Thinking*, *IFIP Conference on Computer Education*, 1970, Amsterdam: North-Holland.
- [25] Papert, S., *Teaching Children to be Mathematicians Versus Teaching About Mathematics*, *Inter. Jour. Math. Educ. Sci. Tech.*, Vol 3, 249-262, 1972.
- [26] Polya, G., *How to Solve It*, Princeton University Press, 1957.
- [27] Shaw, C. *JOSS: A Designer's View of an Experimental On-Line Computing System*, *AFIPS Conference Proceedings*, XXVI, 1, Fall, 1964, pp 455-464.
- [28] Smith, David C., *PYGMALION: A Creative Programming Environment*, doctoral dissertation, Stanford University Computer Science department, to appear, 1975.
- [29] Sutherland, Ivan C., *SKETCHPAD: A Man-Machine Graphical Communication System*, MIT Lincoln Laboratory TR 296, May, 1965.

