TEACHING SMALLTALK
(2 papers)

SSL 77-2 June 1977

Methods For Teaching The Programming Language Smalltalk

by Adele Goldberg and Alan Kay

Smalltalk In The Classroom

by Adele Goldberg

Key Words and Phrases:

Smalltalk, computer uses in education, teaching children programming, computer-based curriculum, message-oriented programming language, programming projects.

CR Categories

1.5, 1.50, 4.22

XEROX

PALO ALTO RESEARCH CENTER
3333 Coyote Hill Road / Palo Alto / California 94304

Abstracts

Methods For Teaching The Programming Language Smalltalk

by Adele Goldberg and Alan Kay

A description of the Smalltalk programming language is presented, based on how it is taught to children. Curriculum materials are composed of model Smalltalk class definitions that the student uses, modifies and extends to new models. A problem formulated in Smalltalk involves, first, description of all the objects that might be involved in the solution and their relations; second, grouping of these objects in classes according to the similarity of actions each can take; third, design of the message system that the objects will use to communicate with one another; and fourth, creating members of each class with the desired characteristics. Each member of the class remembers its individual properties, but refers to the class definition in order to know what messages it can receive.

After presenting Smalltalk as a message-oriented system, a number of projects carried out by junior high school students over a two-year period are exhibited. Student-made videotapes are used to monitor tutorial sessions and to evaluate the final results.

Smalltalk In The Classroom

by Adele Goldberg

We have been teaching the programming language Smalltalk since the Spring of 1974. In 1976 we placed several Smalltalk systems in the independent study center of a Palo Alto Middle School. Three new courses were taught, in computer simulation methods, graphic techniques, and geometry. Each course is described, illustrating a number of applications of the Smalltalk system. Evaluative comments on the use of the school resource center are provided.

XEROX

PALO ALTO RESEARCH CENTER
3333 Coyote Hill Road / Palo Alto / California 94304

METHODS FOR TEACHING THE PROGRAMMING LANGUAGE SMALLTALK

by

Adele Goldberg and Alan Kay

Xerox Palo Alto Research Center

Learning Research Group

INTRODUCTION

Computer programming is a popular subject area for school-age children. The basic goals of courses available in many elementary, junior, and senior high schools are to give students an introduction to computer technology and its potential impact on society, as well as to teach programming as a problem-solving tool. Computer programming provides a unique environment in which to teach problem solving concepts: it is a context in which the concepts are immediately useful and create immediate pleasure. For each programming project, there are a variety of possible solutions and a variety of methods with which the solution can be presented. For these reasons, and because a student can generate his own project ideas, programming is a creative activity. As we will point out again, a number of properties of good problem-solving can be emphasized through programming: planning and organizing, describing, implementing, carrying out a plan to completion, and evaluating. Through the immediate feedback provided by the computer (from error diagnostics or the results of executing a program), a child becomes his own evaluator and, potentially, gains confidence in his learning abilities.

We are devising curriculum materials which center around a programming language called Smalltalk [1]. Smalltalk is implemented on a small, stand-alone computer system. This computer system includes a high-resolution, black and white display screen for graphic input and output, devices for pointing at objects on the screen, a typewriter keyboard, and a digital-to-analog converter for sound output. Although we attempt to teach basic programming concepts and analytic skills in the Smalltalk curriculum materials, this system differs from other computer environments significantly enough to result in a very new learning experience. The combination of a computer medium in which it is possible to access and use a variety of kinds of information (sound and pictures, as well as text and numbers), and a programming language that makes it not only easy to do these activities, but also obvious that they can be done, is a powerful new medium for educational applications.

COMPUTER PROGRAMMING

Historically, computer programming projects have concentrated on numerically-oriented problems, with a great deal of emphasis given to playing and writing interactive games. Since the input/output devices of the past have typically been teletypewriters, the games concentrated on typewritten output and, if possible, short, one letter responses by the game user. Availability of plotters tended to increase the emphasis on picture making (computer art).

Other programming curricula included picture construction with lines, following the introductory work in Logo "Turtle Geometry" [2]. In some systems, the completed line-drawn pictures could be animated. Non-graphic projects examined the development of number systems by reproducing algebraic algorithms and arithmetic-teaching programs [3]. Some projects examined the structure of simple sentences in order to generate English sentences or foreign language translations. Also, a number of innovative studies in physics, topology, and economics have been carried out using computer models [4, 5].

Typically, a student's resultant computer program is in the form of a linear sequence of operations that tends to be a set of simple transformations on numbers, strings, or matrices; the program causes some expected result. The student programmer is taught to view the problem in terms of subproblems to be solved, and to represent the solution of the subproblems in terms of computer programs that can be combined to provide a solution to the problem as a whole. Instructional emphasis is often placed on finding a representation for the problem data (for example, the grid for a board game or the grammar of a language) that allows the data to be easily accessed and manipulated. Most programming courses for students in junior or senior high school also try to give the students experience using basic computer science techniques, such as search and sort algorithms, various levels of numerical methods, or special "tricks" for representing data for fast retrieval.

These projects are useful in teaching basic programming concepts, in particular,

sequencing
conditional action
evaluation of stored sequences
literals
name/value pairs
procedures and procedural parameters

as well as higher-level skills for dealing with computational context, goal/subgoal planning (problem formulation), representation of information, and self-evaluation of results ("debugging").

THE PROGRAMMING LANGUAGE SMALLTALK

Smalltalk differs from most other programming languages in that the act of programming in Smalltalk is one of description of a model. The programmer describes how a single object works, and then abstracts this to describe the actions and properties of a general class of objects. In the process, it is necessary to clarify the notion of what properties are characteristic of all, or only of some, class members. These methods are closest to that of Smalltalk's grandparent, Simula [6].

A problem formulated in Smalltalk involves,

first, description of all the objects that might be involved in the solution and their relations;

second, grouping of these objects in classes according to the similarity of actions each can take;

third, design of the message system that the objects will use to communicate with one another; and

fourth, creating members of each class with the desired characteristics. Each member of the class remembers its individual properties, but refers to the class definition in order to know what messages it can receive.

Ultimately, it is necessary to embed the members of the classes in a controlling method for scheduling their activities. Given an easy way of expressing class properties and the set of ways for classes to interact, specifying algorithms is as simple in Smalltalk as in other languages.

Hence, a great deal of emphasis in teaching Smalltalk is placed on skills connected with the organization and communication of information. At some stage, the students' abilities to specify an algorithm is assumed; instruction turns to the problem of how to model a complex situation of interacting, active components. Because Smalltalk is specially designed as a simulation language, the act of building the model results in a description that runs on a computer. The model can then be observed, modified, and run again.

Smalltalk Project-Proposing Curriculum

We introduce Smalltalk by presenting the student with a model Smalltalk class definition. The key idea is to furnish a definition of a class whose members can cause effects that interest the student. Programming concepts of sequencing, naming, and conditional action are taught by having the student write simple sequences that make use of one or more instances of the class. In this manner, the student learns two basic Smalltalk ideas: instantiation--creating members of a group or class of objects; and communication--sending a message to an object in order to have the object carry out some sequence of actions. He also learns that, while the class definition contains the description of messages and actions, it is the instances of the class that actually do the work, i.e., draw lines, paint gray tones, print text, and so on. This factoring of descriptions and actions is different from many programming languages in which the programmer must deal with the active sequence of events at the same time he or she tries to specify the abstract objects.

The student can then modify the class definition in order to add new capabilities or change the methods used to realize current ones. Each new class member, as well as those already created before the class was modified, responds immediately to any changes in the class definition. The student can thereby explore the concept of sharing capabilities and knowledge among class members. When an object receives a message that it understands, the object responds by carrying out some sequence of instructions. By substituting different sequences of instructions in the original model, the student can produce new results. For example, the student could create a new

graphic image as the response to the message *display*. This modified model can be further extended, or it can be used in new contexts.

This basic idea of presenting a model Smalltalk class definition that the student uses, modifies, and expands, is called a "project-proposing curriculum." The model is proposed as an example of a class with capabilities extending over a whole family of objects. For example, a class of *squares* might have capabilities similar to any regular polygon, such as triangle, hexagon, or circle. The model is involved in a set of initial projects that are chosen to describe the language syntax and basic communication concepts. Each model has several possible extensions from which the student can choose in order to learn new programming techniques, for example, graphic communication or storage methods. The explanations of model projects, with suggested extensions, are presented to the students in the form of illustrated booklets.

The curriculum framework we use, then, involves

use of an already existing model,

reproduction of the model with some addition,

substitution into the model to produce a new result (new class definition), and then

recycling with further extensions to the class definitions, and introduction of the model in new contexts.

In their initial experiences, the students explore models that get them into programming with quick and fun results. At the same time, they get sufficient knowledge about the basic concepts so that they can carry out project ideas of their own. In this manner, programming techniques are taught and practiced within a structured framework that encourages innovation on the part of the students.

An Example Project: Box

One introductory activity we have employed with elementary and junior high school students is a series of projects that use, modify, and extend the definition of a *box* class. A box is an object that looks like a square: it can be drawn on the display screen or erased; it can grow bigger or smaller, and turn right or left; it is possible to make a box move to different screen locations. The illustrated booklet, *The Box Book*, helps a student learn Smalltalk by exploring square boxes, having them "play leap frogs", "dance" together, and make designs. The following explanation is adapted from the booklet.

Smalltalk consists of objects that send and receive messages. Each object must know how to receive and respond to any messages it is sent. Smalltalk is like a post office. The objects are like people who mail letters containing "messages". The object, or person, who receives the letter must read each message to find out what things to do. The name *box* is the name of a class of

Smalltalk objects that we have provided. If you type

```
box new name "joe"!
```

you are telling Smalltalk to create joe, a new member of the box class. (Note that the symbol we call "do-it", !, indicates the end of the message.) Because joe is now a member of the box class, anything a box can do, joe can do.

For example, a box can *grow*. You can send joe a message to grow by typing

```
joe grow 50.!
```

which says to a box, increase the length of your sides by 50 units. The original image of the box as a square on the screen disappears and a larger square appears.

You can type different numbers after the message grow in order to have joe grow by different amounts. You can make joe smaller by typing a minus sign before the number.

```
joe grow -20.!
```

After creating more Smalltalk objects that act like boxes, you can tell them to grow different sizes. For example, try

```
box new name "ann".!
ann grow 100.!
box new name "jan".!
jan grow 60.!
```

The result is shown in Figure 1.

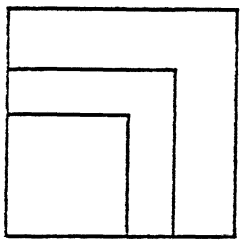


Figure 1.

If you type the message

```
3+4.!
```

you are sending a *number*, 3, the message to add itself to another number, 4. The number 3 responds with the sum, 7. When a box receives the message grow, it expects to then receive a

numerical value telling it how much to grow. That value can be the result of a message to a member of the number class. So, for example, you can type

```
joe grow 10 * 3.!
```

or, if i is the name of an number,

```
joe grow 10 * i.!
```

A box can *turn*. Send **joe** the message to turn by typing

```
joe turn 30.!
```

joe turns right because the number is greater than 0. To make **joe** turn left, you type a minus sign before the number.

```
joe turn -45.!
```

ann can turn around and around when you repeatedly send her a message to turn. Tell **ann** to do the turning lots of times

```
do lots (ann turn 10).!
```

Or **jan** can rock back and forth.

```
do lots (jan turn 20. jan turn -20).!
```

By working with boxes in this manner, the students learn about the display of graphic information on a video screen. By examining the changes in the boxes, they learn about the relative sizes and orientations of graphic objects.

In *The Box Book*, we adopt the metaphor that programs are "movies" in which there are "roles" that actors play. A role may always be filled by the same actor, that is, the role is a constant. For example, in the movie in which the box **jan** rocks back and forth, the role of the rocking box is always filled by **jan**. But usually the actors vary, that is, the role is a variable.

The Box Book continues by introducing the *movie* class, a method for creating a script for one or more characters. One movie might have an object spin around and around. Let's write a movie in which any member of the box class can be the actor. We will give the movie the name *spin*.

```
movie new name "spin".!
```

This movie will have one character, **wheel**.

```
spin characters wheel.!
```

In the script, the character **wheel** is told, 36 times, to turn 10 units. The counter **'repeat'** can start at 1 and count to 36.

```
spin script (do 36 (wheel turn 10)).!
```

Who will be the star in the movie? Each time we want the movie to play, we must send it a message that tells which box will be the star actor. For example, we might type

```
spin with jan.!
```

The object **wheel** in the definition of **spin** is the name of a character in the movie. It is just like Mr. Spock in the movie Star Trek. "Mr. Spock" is the name of the character, and Leonard Nimoy is the actor. When we type

```
spin with jan.!
```

the movie **spin** is given a chance to do something. The first thing it does is receive the message **with**. It then responds to this message by finding out who will play the role, **wheel**. In this case, the message was the box **jan**. Now, whenever the script for **spin** tells the character **wheel** to do something, it is actually the actor **jan** that does it. That is, **wheel** is an alias for whichever square box we name in the message.

The Box Class Definition

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 *pen*. The pen remembers the proper orientation and location on the display screen.

The pen class is provided in the basic Smalltalk system. Members of the class respond to messages to draw a line on the display screen. The line can be black or white, thick or thin. The pen itself can be positioned at different points on the display screen, facing in any direction. We might use a pen to draw a square:

```
pen new name "pal".!
do 4 with (pal draw 100. pal turn 90).!
```

Here we created a pen whose name is **pal**. We then told the new pen to draw a line 100 units long and then to change its orientation by 90 degrees (turn a right angle). Doing this four times means that four lines, of equal length and perpendicular to one another, are drawn on the display screen.

In *The Box Book*, we show the students a "planning table" for the box class. Table I is an example of such a table. In it we include English descriptions of the intended response to each

planned message, followed by the Smalltalk descriptions needed to carry out the response. In the example planning table, we have used a special Smalltalk symbol **SELF**. It is a reference to the currently active class instance, and is used for having an object send itself a message. For example, to create a new instance, the class responds to the message **create**. The response is to create the pen, **pal**, and the number, **size**. Then the new instance sends itself the message **draw**. Each time you see a colon (:) in the Smalltalk description, it means that the object expects to receive a value, an instance of a class such as number, from the message. The name **SELF** is actually not required; interpreting a message such as **undraw** within the context of the class description refers first to the class' dictionary of messages and then to the supraclass', and so on.

The response to the message **undraw** is to erase the box from the screen. The pen can draw with white or black ink. We assume the background of the display is white. Drawing with white ink is a way of erasing black marks. So we can erase the box by changing the pen's ink to white and having the box object send itself a message to draw. This effectively has the box trace over its square image with the white ink.

Modifying the Box Class

The students can then modify the box class definition in order to make it possible for members of the class to move around the display screen. Suppose we would like to type

joe moveto point 200 100.!

We want this message to mean that the box **jan** will reposition its graphic image such that the lower lefthand corner of the square is at screen coordinate 200, 100. Hence, to get boxes to move around, the students must learn about the coordinate system of the display screen. The plan for the new message is shown in Table II. It shows that, to move the box, it is first necessary to erase the current image, get the pen **pal** to change locations, and then draw the box again in **pal**'s new location.

A box can also move around the screen by following a pointing device. Typically, the pointing device is a "mouse", a rectangular object with three buttons on it. It inputs its x and y positions as it is moved about on a table. A cursor on the screen tracks the mouse position. The mouse is a Smalltalk object that can send three different messages about (1) the vertical position of the cursor (**mouse x**), (2) the horizontal position of the cursor (**mouse y**), and (3) the combination of buttons currently pressed (**mouse button <button identifier>**). The message **mouse point** returns the point at **mouse x, mouse y**.

To get the boxes to follow the mouse, the students must learn about the correspondence between the movement of the mouse and the movement of the cursor on the screen. We devised several hand/eye coordination exercises to give the students practice with this new mode of

communication. Simple programs such as "chase" and leapfrog games, and "sketching" with different-sized boxes, resulted from the use of the new move message, and provided further practice with the pointing device.

The students can write movies that make a box grow, turn, and follow the cursor only when a mouse button is pressed. To write a conditional statement in Smalltalk, one uses a special symbol, ⇒. This symbol is always preceded by a question and followed by one or more messages delimited by parentheses. The message(s) will be sent only if the answer to the question is not false.

```
question ⇒ (action if question is not false)
```

Suppose the mouse buttons are ordered left to right on the rectangular surface. Then a possible box control movie is

```
movie new name "boxcontrol".!
boxcontrol characters star.!
boxcontrol script
 (do lots
    (mouse button left        ⇒ (star grow 5)
    mouse button middle       ⇒ (star turn 10)
    mouse button right        ⇒ (star moveto mouse point))).!
```

If the left button is pressed, the character whose name is star will grow 5 units; if the middle button is pressed, star will turn 10 units; if the right button is pressed, star will change its location to follow the cursor; else, we repeat the process of checking for mouse buttons.

Who can be the actor in this movie? Clearly, any box. But actually any Smalltalk object that can "read the script", i.e., respond to the messages in the sequence (in this case, **grow**, **turn**, **moveto**). We might type

```
boxcontrol with joe.!
```

or

```
boxcontrol with ann.!
```

If objects respond to the same messages, then, and only then, they can fill the same roles. So, for example, the students can modify the box class definition, substituting other shapes for the drawing of a square box: a circle, a rectangle, a spaceship, a flower, and so on. These different shapes still respond to the same messages as box; therefore, a member of one of these new classes can fill the same roles as a box in any of the students' already-defined movies.

There are many possible extensions of the box project that the students are encouraged to try. For example,

increase the numbers of characters in a movie;
give boxes the ability to have different border widths;
keep track of the box instances you create in order to guarantee that one box doesn't erase another box;
create a member of a class as a copy of an already existing member;
define the class polygon;
send messages by pressing mouse buttons or pointing to words or pictures (menus) on the display screen;
use the class *rectangle* to have boxes that are "painted" with different gray tones;
use the class *paragraph* to have boxes with text words in them; or
create the class *picturebox*, a box in which you can sketch a picture and move it around the display screen.

More general types of extensions to projects are discussed in [7]. In a subsequent section, we will describe several extensions that Smalltalk students have completed.

In summary, Smalltalk is based on a few simple anthropomorphic metaphors having to do with communication, state, and classification. The basic parts of Smalltalk to be learned consist of:

(1) The notion of classification: in particular, the syntax for defining a class of objects and the methods for creating members of a defined class.

(2) The methods of communication: in particular, mechanisms for sending and receiving messages. In order to trace the flow of events, the programmer asks: *who* sends the message, in *what* context; *who* receives the message, *when*; and *how* does the receiver get the message. Sending yourself a message (recursion) is treated as a natural phenomenon.

(3) The special symbols of the language:

for receiving messages

: receive the value of the next thing in the message
ß receive literally the next thing in the message (next word or words grouped by parentheses)
ß peek to see the literal next thing in the message, and see if it matches an expected word;
◁ peek to see the next word in the message, and, if it matches an expected word, then fetch it.

for returning values

↑ followed by an expression

for specifying a conditional clause

⇒ as in

question ⇒ (action if question not false)

(4) The methods for iteration, basically for repeatedly carrying out some sequence of instructions

do lots (<instructions>**)**

or doing something a fixed number of times

do <number of times> **(**<instructions>**)**

There are other, of course, more complex scheduling methods not typically taught at this level of instruction.

(5) Knowledge of the basic classes already defined and available to every Smalltalk user because these classes have proven to be generally useful; e.g., *number* and *float* (for arithmetic); *pen*, *rectangle*, and *paragraph* (for presenting graphic information); and *point*, *text*, *list*, and *file* (as storage methods).

STUDENT'S EXTENSIONS OF BOX

During the past three years, we have used *The Box Book* with a number of children ranging in age from 9 to 15. They have followed the suggested extensions in order to create a number of interesting projects. In this section, we describe a few examples.

Painting Systems

The clown construction example in Figure 2 was done by nine-year old Kathy who wrote class definitions for several geometric shapes, each modelled after the box definition. The shapes are controlled by pushing buttons on the pointing device. Her clown, like many of the "box movies" she wrote, is part of her attempt to learn how to use geometric abstraction to represent movement and real forms.

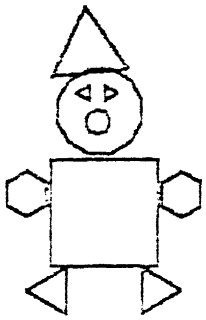


Figure 2. Kathy's Clown constructed from geometric shapes.

Another student, Marian, placed images of each kind of geometric shape in a box at the top of the display screen. She could then point with the cursor to one of the boxes in order to copy the shape onto a new location on the display screen, thus using the shapes as though they were *ink stamps* (Figure 3).

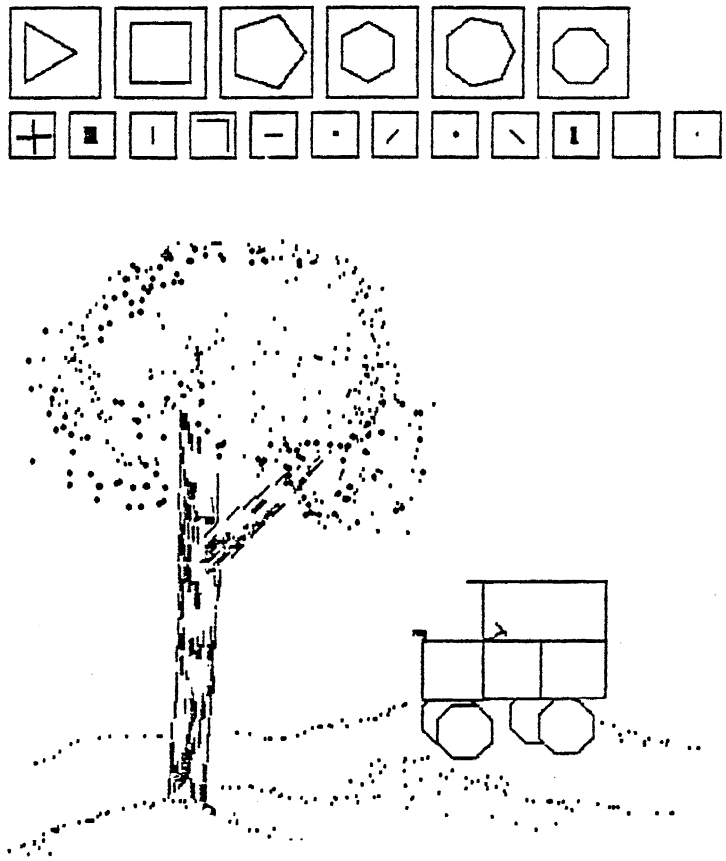


Figure 3. Marian's Ink Stamp painting system.

Twelve-year old Susan generalized these geometric classes into a class of polygons she named *shapes*: each instance remembers its position on the screen, its orientation, its size, and the number of its sides. She changed the meaning of the message **grow** in order to increase the number, as well as the length, of the sides. After learning how to create Smalltalk text windows (rectangles with text in them), and how to determine which word the cursor is pointing at, she defined a class *menu* (shown at the bottom right corner in Figure 4). Susan is able to point to a shape instance on the screen, and then send the shape messages to grow, turn, move, change its border width, delete, or copy itself by pointing in a menu of message words.

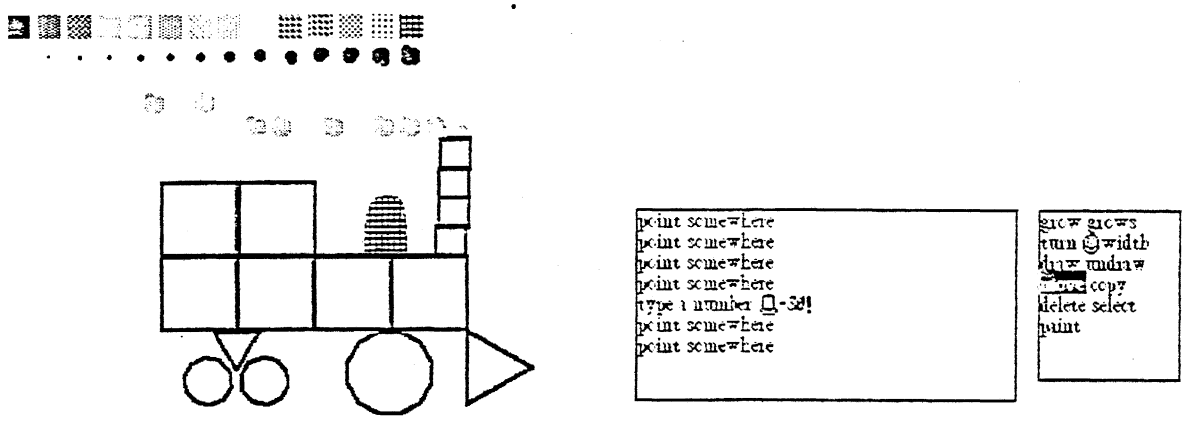


Figure 4. Susan's Painting tool includes menus for creating geometric shapes and painting with gray tones.

These kinds of painting tools are generally useful. With them, we can devise exercises that help a student form abstractions, for example, to see a triangle as a hat, a tree, an eye, or a nose, depending on the context in which the pieces are arranged.

Spacewar

Several kids were interested in designing rocketships for the game of spacewar. Dennis wanted his ship to shoot torpedoes, while Kathy was interested in simulating rocket takeoff, ignition fire, travel, and landing. Kathy's rocketships are simple extensions of the box class in which the response to the message **draw** is to combine a rectangle and two triangles to form a ship with fire coming out one side. Dennis invented two kinds of ships: instances of the class *trek* were peaceful ships that moved in formation through space, serving as "sitting ducks" under attack by the *war* ships. Both ships move forward and backward, turn left and right, move slow or fast, all under keyboard or mouse control. The ships shoot rockets which cause other ships to explode on impact.

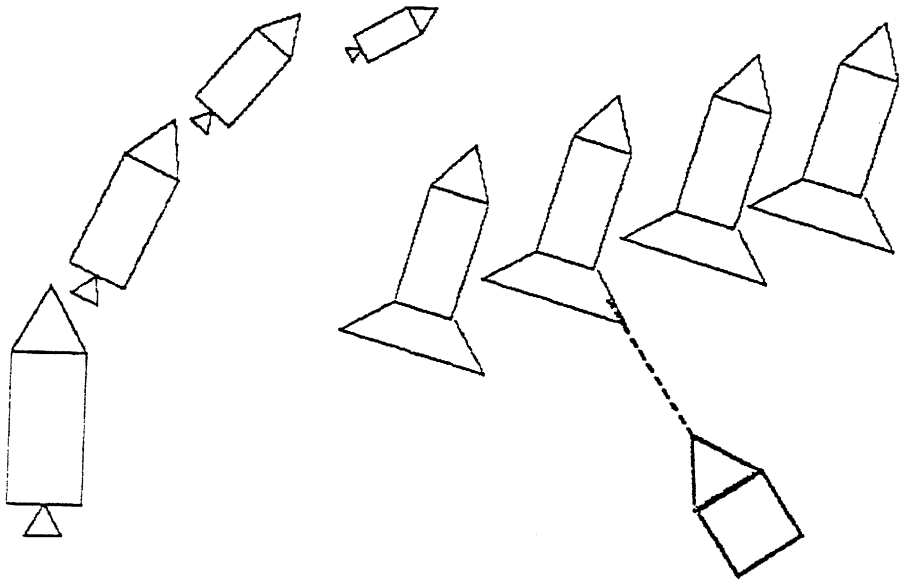


Figure 5. Various games of Spacewar.

We note that, in order to do their projects, these programmers had to understand division by negative numbers, testing inequalities, counting by 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. In each example, the students had to be able to schedule the activities of the several kinds of objects in the environment simultaneously. These concepts were made clearer to the students because they had a need to know the concepts in order to apply them to their project work.

Games

Lisa extended the box 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.

Dennis wrote a guessing game as a means for learning how to read characters from the typewriter keyboard. The object of this game was to choose a secret code number corresponding to a character on the keyboard; the player tries to guess the appropriate character by striking the keys.

Lisa used this game, 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 in her extended 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 appears (see Figure 6).

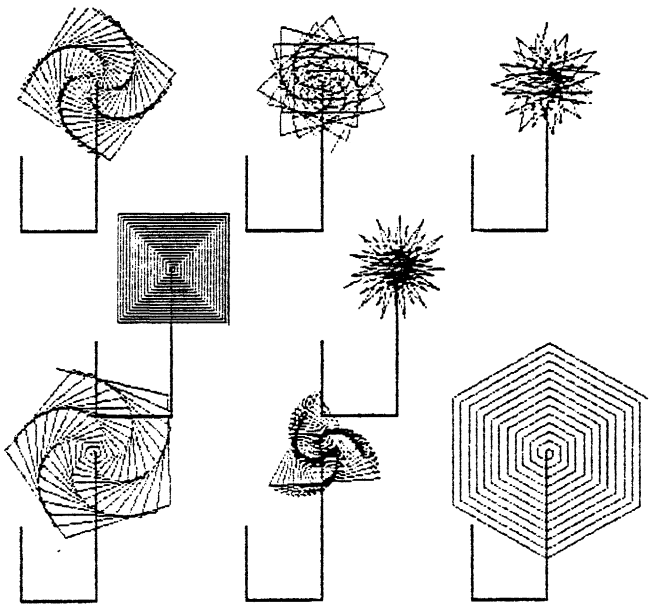
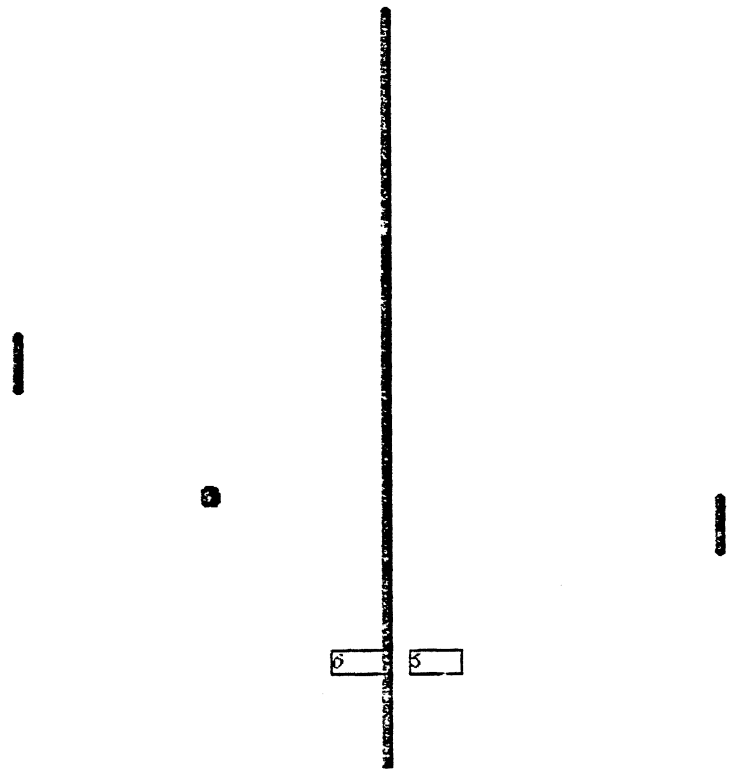


Figure 6. Lisa's box guessing game. When all the secret codes have been guessed, each box displays a different spiral design.

She then extended the definition once more by adding the ability to point with the mouse cursor to a box on the screen. Now the player presses a button on the mouse to indicate that the game should start; the box that finds the cursor inside its square area plays the guessing game.

Pong, shown in Figure 7, was implemented by two students, Elliot and Sandy. They scheduled paddle turns according to input from the keyboard, and kept a running score in a scorebox. Comparing their original game to those in the stores, the boys realized that the store version provided different ball returns depending on where the ball hit the paddle and whether or not the paddle was in motion; they revised their own game accordingly. Because the boys designed the class definition for *paddle*, they were able to extend the game so that each player had multiple paddles.



By Elliot, age 11, and Sandy, age 12. The pong game.
The following keyboard characters control the paddles.

	left paddle	right paddle
move up	a	k
move down	s	l

Figure 7. The game of Pong.

Scott's blackjack game shown next makes good use of the menu idea for selecting messages, and extends the notion of a box as an area for holding information, in this case, about a playing card. The game depends on two class definitions for *player* and *card*; it runs by shuffling and dealing cards to the different players, one of whom is always the dealer, and keeps cumulative track of the winnings and losses of the players.

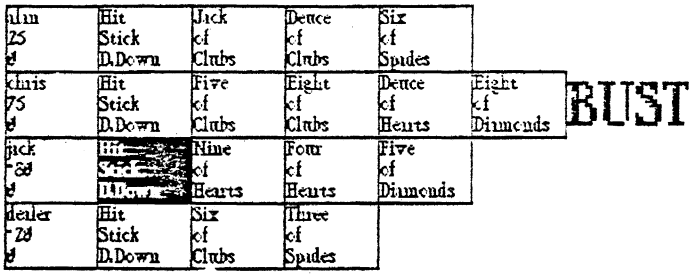


Figure 8. The game of Blackjack. In the middle of playing, it is Jack's turn to select from the menu of "hit, stick, or double down".

Simulations

After several attempts to group the parts of the body, twelve-year old Marian provided definitions for *arm, leg,* and *head.* Her class *human,* then, owns instances of arms, legs, and a head, and was capable of moving at all possible body joints. She then placed the members of the class human in dance routines, baseball games, and a badminton game.

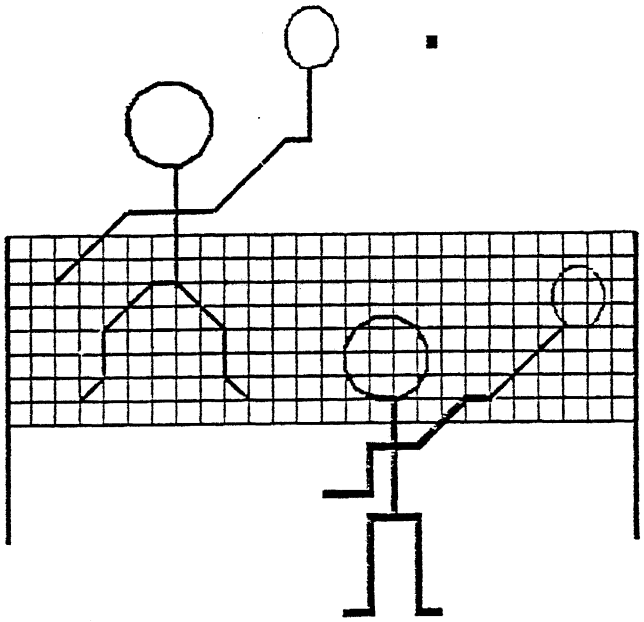


Figure 9 Marian's simulated game of badminton.

Many other examples of simulations were done in the context of a course taught at Jordan Middle School, Palo Alto, California, and described in detail in a companion paper [8].

OTHER PROJECTS IN THE CURRICULUM

As we have stated, our focus in teaching programming is actually on communication and classification. We attempt to give the students models in which they can easily communicate with some entity, such as the graphic entity of a box, in order to explore its original capabilities

and to provide new ones. There are a number of such entities that we have identified as appropriate for initial programming experiences. In the area of graphics, we are preparing *The Box Book*, as already described; *The Picture Book*, in which the students program using icons that are provided for them, or that they design themselves; *The Movie Book*, for creating animated picture sequences that can be associated with text in order to tell a story; and *The Palette Book*, emphasizing the design and implementation of painting and animation tools.

The Word Book and *The Poetry Book* will help the students work with verbal communication. With access to an on-line dictionary of words, their meanings, and, possibly, visual images, the students can play with and invent word games. A computer graphics systems is an exciting place to explore the impact of the form of a word--the size of its characters, whether they are bold or italics, whether they are static or dynamic--these all enter into the message the word actually carries. *The Poetry Book* explores a new use of this medium, something we call "dynamic concrete poetry". There is planned, of course, a *Number Book* and a *Book Book*, for writing down stories, fiction or non-fiction, using both pictures and words. And a *Music Book* for composing musical pieces and designing instruments to play them. *My Book* will be a place for creating calendars, diaries, budget forms, greeting cards, and so on.

One thing that makes a computer special for educational purposes is the ability of its user to store and retrieve a variety of kinds of information--sound and pictures, as well as text and numbers. This multi-modal approach provides several ways in which students can retrieve and view information. With a storage/retrieval system that allows the student to add his own data and his own methods for relating the data, students can gather information which they can share with others. The information can then become part of the total bank of information which an entire class can use as a basis for inferences and generalizations. We will explore this idea in *The Find-It Book.*

EVALUATION METHODS

We are interested in evaluating our work on two levels: individual student accomplishments and the teaching methods of the learning environment.

Individual Student Accomplishments

Some individual student accomplishments have been outlined in the previous sections. It is our empirical observation that children in the age group we have worked with most (12 years and older) can learn to write Smalltalk class definitions and to embed instances of the classes in complex environments. They can build tools that demonstrate an integration of problem-solving skills, producing unique extensions of the tutor-provided projects. In doing so, the children have shown a willingness to share their newly acquired knowledge, and to cooperate with their peers and adult tutors to generate new project ideas.

The question remains, however, when using anecdotal evidence such as we have presented, "Who really got the ideas and who really wrote the definitions?" Initially, we attempted to learn this information by saving the student's on-line protocol (each keystroke input). Although this provided us with a sense of which exercises the children repeated often, and which syntactic parts of Smalltalk entered into recurring errors, the information lacked the verbal exchanges and the planning or descriptive sketches provided by the tutor.

Videotaping all class or tutorial sessions appeared to solve this problem. At least one other research group (the Logo group at the University of Edinburgh) has attempted to use videotape. They agree with our observation that the tutor's interference, generally with motivational intentions, is more extensive than imagined. Our videotapes are particularly useful for training new tutors through self-observation, or by criticizing other tutorial interactions. The approximately fifty hours of videotaping we have done will be applied to this purpose.

It is not possible, however, when taping a class of five or more students, to monitor each student all of the time. Moreover, it is not sufficient to follow the tutor (if there is only one) to capture dialogues, because it is also important to obtain the student's responses to the tutorial--making use of plans, reviewing any sketches made, describing any new information to another student, and so on. Often we observed that the students will turn to the tutor for information they could have determined themselves (from written materials or some on-line experimentation). Tutors, wanting to participate actively, rather than waiting to be asked for help, tended to offer unsolicited advice. At the public school in which we placed several Smalltalk systems, however, a tutor was often not available. The students had more time alone, so more of the ideas and solution methods could be attributed to them.

If we assume it is not generally possible to determine the planning and implementing history, we can still determine the extent to which the student comprehends a completed project. Comprehension can be measured by having the student use the project, in expanded form or in a different context. Each student is asked to describe the projects he or she carried out. The verbal description and an on-line demonstration is videotaped. In order to "improve the videotape", the student is asked by the "cameraman" to make changes in individual definitions, or, in the case of a class-planned tape, to integrate (coordinate) two or more projects. Typical changes involve the

- regrouping of instances of two classes into one class
- control methods (e.g., use of the keyboard rather than mouse buttons, or use of a textual menu rather than typing)
- numbers of objects (e.g., a race for five rather than ten cars)
- kinds of objects (e.g., substitute a swimmer for a car in a race simulation)

- mixture of kinds of objects (e.g., a waiting line with trucks as well as cars)

- placement of graphic information on the display screen (perhaps a suggested improvement in layout, or thicker lines for better visibility)

- (re)scheduling of existing objects with different actions to take during a simulation, or with different decisions models

Success in this task is purely a behavioral test of whether or not the changes were carried out. Each of the above kinds of changes requires the student to know who (which object) has what information, or who has control of which parts of the active events. As we said earlier, these are kernel ideas in writing Smalltalk programs. This test is a learning experience that is fun; it provides a videotape for the class of their work. When carried out as a class-planned tape, changes for improvement are imposed by the students themselves, providing additional review. Use of this non-competitive evaluation method is described in [8].

Teaching Methods

The two levels of evaluation are closely tied, since the students' accomplishments result from the programming projects proposed in the curriculum. The teaching methods used are highly structured in terms of particular introductory materials to be followed by each student; they are very flexible with regard to the kinds and numbers of modifications the students might make to the models. Giving the students the freedom to select introductions from a variety of well-organized materials, and to define the use of the materials according to individual interests, gives the students some control of their learning environment. We have implemented our curriculum in a framework of self-assessment and self-imposed review. Programming is an interesting course of study for this purpose. Projects last long enough so the students can get involved with their work; a number of different visual or verbal ideas can be tried in the search for a solution. Since several solutions are possible, the students can begin to make judgments about the relevant worth of each approach, comparing and sharing new techniques with one another.

Each time a new programming project is attempted, the student is able to practice skills already acquired. Review comes in the form of redoing work, perhaps with some variation in the resulting definitions, or in the form of using previous work (for example, using previously defined data storage methods or methods for user interactions). Algorithms that are embedded in a class definition for one project reappear in new definitions. The students can see a transfer of their (developing) skills into new project areas.

Will the curriculum project books teach the basic programming concepts as intended? This evaluation is carried out in two parts. First, do the books contain material related to teaching these concepts? Second, do the students learn the concepts?

The Box Book is the only one with which we have extensive experience. A researcher, familiar with teaching programming, but independent of the development of *The Box Book*, composed Table III, a categorization of concepts she found in the book. These clearly include the basic concepts listed earlier.

Evaluation of whether or not the students learned from *The Box Book* is confounded by the mixture of students we have had--differences in age and differences in previously acquired programming skills. *The Box Book* seems somewhat oversimplified for students with strong backgrounds in programming in other languages. However, the book does give these students a useful introduction to the new programming notation; it also offers a metaphor that helps the students understand the notions of classes and instances.

The Box Book begins with a simple sequence on using the members of the box class in order to make pictures on the display screen. This material is understandable to children under 11 years of age. Because the concept of classification is not well-known to the younger students, the Smalltalk class concept appears better suited to students above 11 years of age. Hence our interest in providing iconic languages for young children as bridges to the Smalltalk programming concepts. And, of course, non-keyboard input devices, such as a button box or touch-sensitive screen, make the bridges easier to build.

Our teaching method, to propose projects that the students try out and modify, is based on the premise that there is no one best way to teach programming, but that we should provide a variety of projects and a variety of teacher/student arrangements from which the kids can choose. The projects, to be successful, have to lead the students on enough to get them to generate unique extensions. If the students only try out the examples as given, then the curriculum is, in a sense, a failure. For the students we have worked with, the material has encouraged many different kinds of projects that have sustained their interest for long periods of time.

REFERENCES

1. A. Goldberg and A. Kay (Eds), (1976), *Smalltalk-72 Instruction Manual*, Xerox Palo Alto Research Center Technical Report SSL-76-6.

2. Seymour Papert, Teaching Children Thinking, (1970), *IFIP Conference on Computer Education*, Amsterdam: North-Holland.

3. Wallace Feurzeig, et al. *Programming-Languages as a Conceptual Framework for Teaching Mathematics*, (1977), Final Report on BBN Logo Project.

4. Hal Abelson and Andy deSessa, (1977), Student Science Training Program in mathematics, physics, and computer science, MIT Logo Memo 29.

5. Marian Visich and Ludwig Braun, (1974), *The Use of Computer Simulation in High School Curricula*, Huntington Computer Project, State University of New York at Stonybrook.

6. Ole-Johan Dahl, and Kristen Nygaard, (1966), SIMULA--an ALGOL-Based Simulation Language, *CACM, IX*, 9, pp 671-678.

7. A. Goldberg and B. Tenenbaum, (1975), Classroom Communication Media, *ACM SIGCUE TOPICS in Instructional Computing*, Vol 1, Teacher Education.

8. A. Goldberg, *Smalltalk in the Classroom*, (1977), Technical Report SSL-77-2 (this report).

Table I. A Planning Table for the Box Class

Message the box can receive	English description of the action the box will carry out	Smalltalk description
new	It creates a new box that needs its own *pen* to draw the new box on the display, and that must remember its **size** whose first value is 50. Then it draws itself on the display screen	**pen new name "pal".** **number new name "size".** **size value 50.** **SELF draw.**
draw	The box has its pen draw a square on the screen at the pen's current location and orientation. The length of its four sides is **size.**	**do 4** **(pal draw size.** **pal turn 90).**
undraw	Erase the box.	**pal white.** **SELF draw.** **pal black.**
grow	After erasing itself, the box instance retrieves a message which is interpreted as an increment of its size. It then redraws itself as a bigger or smaller square.	**SELF undraw.** **size increase by :.** **SELF draw.**
turn	After erasing itself, the box instance retrieves a message which is interpreted as an increment of its orientation. Note, since the pen, rather than the box, remembers the orientation, the box has to tell the pen to turn.	**SELF undraw.** **pal turn :.** **SELF draw.**

Table II. Plan for Adding a Message to the Box Class

Message the box can receive	English description of the action the box will carry out	Smalltalk description
moveto	After erasing itself, the box instance retrieves two messages which are interpreted as the new coordinates of the box. Note, since the pen, rather than the box, remembers the location, the box has to tell the pen to place itself at the new location.	**SELF undraw.** **pal place at :.** **SELF draw.**

Table III. Concepts Found in The Smalltalk Project Book: The Box Book

Smalltalk

objects
receive and send messages
name/value
class (has name)
instantiation
completeness of description
syntax
SELF--name for object that is looking at the message
messages consist of many submessages

sequence of instructions

grouping
can have name (procedure)
sequence generalized as procedure
local variables
contrast between procedure definition and calling on a procedure
procedure may receive more than one message
redefinition

flow of control

order of evaluation
hierarchies
conditional clauses
iteration
infinite loop
repetition
increment
termination conditions
interrupt

context

symbol may have different meanings in different contexts
substitutability (e.g., instances of same class)
type constraints (e.g., instances of different classes that can play same role in a procedure)

problem solving

many possible solutions to a given problem
planning, templates
identification of components
determination of differences
expansion of model
demonstration of active process
analysis (is it working as you thought it would?)
coordination, integration (e.g., shooting one box out of another)
extension (e.g., make polygons from box class)
interrogation -- ask objects about their properties
experimentation--what will happen if? try and see...
reaching a final answer through successive partial results

general

increasing dimensions
invisible existence (objects exist but may not be shown on the display)
negative numbers and binary arithmetic
orientation, rotation
denotation (e.g., multiple names of same object)
Cartesian coordinates--lines named by numbers
directionality
complementation, inversion (e.g., of display screen--black to white)
defaults
printing

SMALLTALK IN THE CLASSROOM

by
Adele Goldberg
Xerox Palo Alto Research Center
Learning Research Group

INTRODUCTION

In an earlier paper [1], we described the methods we have developed for teaching children how to write computer programs in a language called *Smalltalk*. We are interested in this endeavor for two main reasons. First, programming can be rewarding both as a creative activity and as a model of analytic problem-solving. A child gains confidence in his learning by becoming his own evaluator. This is accomplished through the immediate feedback provided by the computer, or by the child's own comparison of his results with his intentions. In his ability to generate programming project ideas, a child also gains some control over his learning environment and, potentially, is motivated to explore more, or deeper, problem solving tasks. A number of properties of good problem-solving can be emphasized through programming: planning and organizing, describing, implementing, carrying out a plan to completion, and evaluating.

An even more significant reason stems from our interest in developing methods for manipulating information in ways that are interesting to users who are not computer professonals. These methods include ways to store and retrieve information, edit it, organize it, provide structure for it, speculate about it, and generalize about it. A difficult and important problem in designing such an information system is that of providing easy means to:

give the owner of the computer immediate control over the available information, and

allow the owner to augment the system with new methods that fit individual requirements.

Children (and their teachers) represent a significant portion of the potential users of such a system. A child's learning experiences typically involve searches for new information; new facts and ideas are related to those previously acquired. Reorganizing information often helps highlight non-obvious connections. For example, building special classifications for the facts, such as a timeline or a graphic map of trade relationships or of territorial control, often makes it possible to discover dependency relationships.

Hence we are interested in providing a system in which children can easily store, retrieve and organize information, and can test out, through simulations that they themselves build, their ideas of how that information can be used. One of our initial concerns, then, was how to teach children to program in a language, Smalltalk, that is especially designed for describing classifications and simulations.

PROGRAMMING CLASSES

We have been teaching Smalltalk to children since the Spring of 1974; for an outline of classes see Table I. The basic purpose of the classes was to develop and use a variety of methods for teaching Smalltalk programming, and to prepare, with the help of our students, illustrated project booklets that the students could use. Our methods, and the format of the booklets, are described in detail in [1].

Until the Spring of 1976, these classes were held at the Xerox Palo Alto Research Center (PARC). In order to participate in our program, a student had to be excused from two consecutive Jordan classes or arranged to attend after school hours. Parents or teachers had to provide transportation, or the kids bicycled a considerable distance. In the Spring of 1976, we placed two Smalltalk systems in the independent study center at Jordan Middle School, Palo Alto, California. This center is funded through the state program for "mentally gifted minors" (MGM). Prior to this, most of our programming classes consisted of seventh and eighth graders (12- and 13-year olds) from Jordan.

Locating the computer systems at their school made it easier for the kids to schedule class and work sessions, especially for shorter time periods. With the elimination of the transportation problem, the students were able to come before school opened, during their "brunch" and lunch breaks or study periods, and after school, as well as during scheduled class periods. Because the computers were located in their school, many students, not singled out for attention in the MGM program, but still curious about the new technology, could be made aware of our classes. We tried to provide tutorial help, before and after school hours, to students who showed such an interest.

The Jordan experience, then, represents an attempt to place the Smalltalk system in an open educational environment or resource center. This five-month experience served as a pilot study for transferring our learning center model into a typical school environment. The model consisted of several Smalltalk systems linked by a communications line (the original two systems were subsequently increased to three), one Diablo printer, and one Data General Nova system and button box device used to control a robot "turtle" [2]. Paper copies of graphic information displayed on the screen were available from a Xerox Graphics Printer located at PARC. In addition to these computer devices, we used standard audio-visual equipment, and a variety of toys and games from conventional manufacturers for the purpose of stimulating ideas and acting out programmable sequences. Materials for building cardboard mocks-ups of businesses and public facilities that (might) use computers was part of a study of the application and impact of the technology on society.

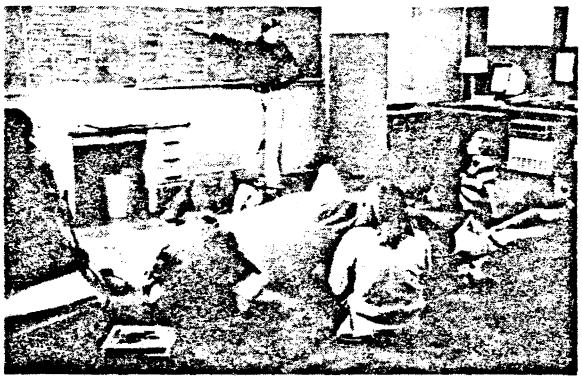


Figure 1. Jordan Resource Center

The casual access to the Smalltalk system at Jordan was highly successful in terms of the usage of the room and in clarifying our own curriculum ideas. Almost 100 students utilized the center's resources, 53 of whom were in formally organized courses. Table II shows a breakdown of the average number of hours spent in the room by students in the various classes. This data was obtained from a log book kept by the students who were responsible for signing in and out of the room. Since students often forgot to sign the log, we must assume that the data is not exact, but is, rather, a lower bound. Also, the time indicated does not include time spent in large group presentations.

Table II includes an entry for "hangers-on". This was a label we used for students who did not actually work on the computers, but just spent time in the room. These students' roles as critics appeared to provide an on-going discussion of the computer's impact on society and its potential for the future; hence their inclusion in the usage table.

JORDAN CURRICULUM

Four courses were taught during the Spring 1976 semester. One, an animation class, was conducted informally by a former Jordan student, now in high school, who returned to the middle school twice a week. Courses in computer simulation methods, graphic techniques, and some geometry were prepared.

Simulation

The initial Smalltalk simulation course focuses on the development of mechanisms for specifying abstractions of physical activities that fit into a counter-service paradigm (counters with clerks, and lines filled with objects such as cars or people waiting for service). The course emphasizes skills in observing dynamic events, collecting data about these events, and using the graphic capabilities of the computer to represent essential characteristics of the events.

The material is considerably more difficult than any we had previously attempted; it was especially directed to students who already had programming experience (each student in the simulation class had previous experience programming in Basic). Two classes were held, each consisting of nine students who attended formal presentations and discussions once a week for a regular forty-minute school period. They were expected to work on the computer at least one additional hour each week. The course continued throughout the five-month semester. Written summaries of the class presentation were always provided.

The course begins with a model of a simple list of objects, in this case, circular bubbles floating randomly around the display screen. Everything in Smalltalk is based on a few simple anthropomorphic metaphors having to do with communication, state, and classification. Every object in the system belongs to a class; objects communicate with each other by sending messages. A class definition contains a description of the properties of each class member and methods the objects use to recognize and reply to messages. Each class has certain capabilities such as drawing pictures, making musical (or other noises), or adding numbers.

The *bubble* class is a model class definition that is easily extended to represent other objects floating about: letters, numbers, snowflakes, designs, polygons. Exercises to modify the graphic representation of a *bubble* introduce methods of visual communication: line drawing, text, and gray-scale sketching. These exercises are similar to those used in *The Box Book* discussed in a separate paper [1].

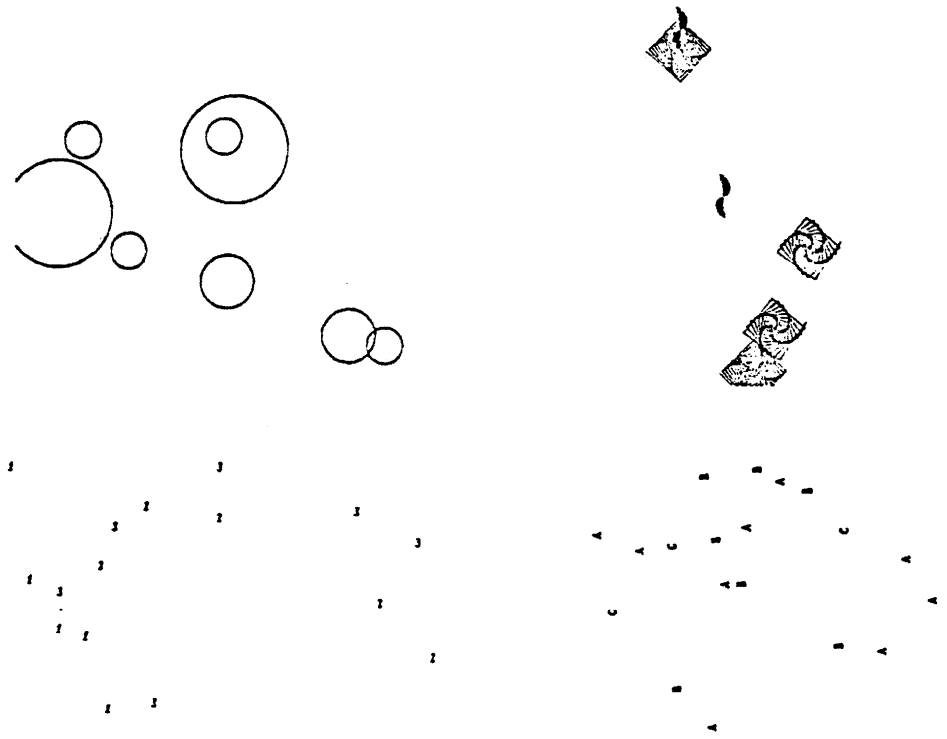


Figure 2. Bubbles and Bubble Extensions

Smalltalk in the Classroom

In this course, we discussed the basic parts of a Smalltalk class definition and the concept of communication among members of classes. We did not teach the syntax of the Smalltalk programming language. Rather, we concentrated on schemes for classifying objects, such as the bubbles, according to descriptive properties and actions. The schemes were written in a special format we call design templates. They are forms in which a student can specify (a) a set of variable names, whose values describe properties of a class (such as the size of a bubble); and (b) a set of messages a class member might be sent. The messages can be annotated with comments stating the intended response to, and expected side effects of, each message.

Students were expected to learn the precise programming syntax from the project booklets and the instruction manual [3], sharing examples and helping one another during small group sessions. They could complete the design templates by appending an appropriate sequence of Smalltalk instructions to the comments. If a student needed help, a tutor could check the correspondence between the comments and the actual instructions, as well as discuss the communications design. This format succeeded (in the sense that each student wrote several class definitions) probably because the students already knew the basic programming concepts of sequencing, iteration, and conditional action. Significantly different introduction methods are necessary with less experienced students.

The simulation curriculum continues by introducing methods for modifying lists. The students built a model of bubbles bursting when they get too near one another; the list of active bubbles changes when a bubble bursts and goes away, or bursts into two or more new bubbles. Here, we focus on the implications of decisions, such as the definition of near and the model of how bubbles will actually burst on contact (one or the other or both). We also examine different scheduling methods for simulating sequential or parallel processes, and study the possibly different results by having the students themselves act out the roles of the bubbles.

We then use lists of objects to create races of cars, horses, and swimmers, beginning an exploration of the idea of collecting statistics on the activities of the objects in the simulated environment. The pictures shown in Figure 3 are taken from some of the students' races. Although each student was presented with an identical model, each designed a unique variation, both in terms of the graphic presentation and in terms of the underlying method for determining the objects' movements. A number of the students incorporated a *clock* to indicate "simulated time".

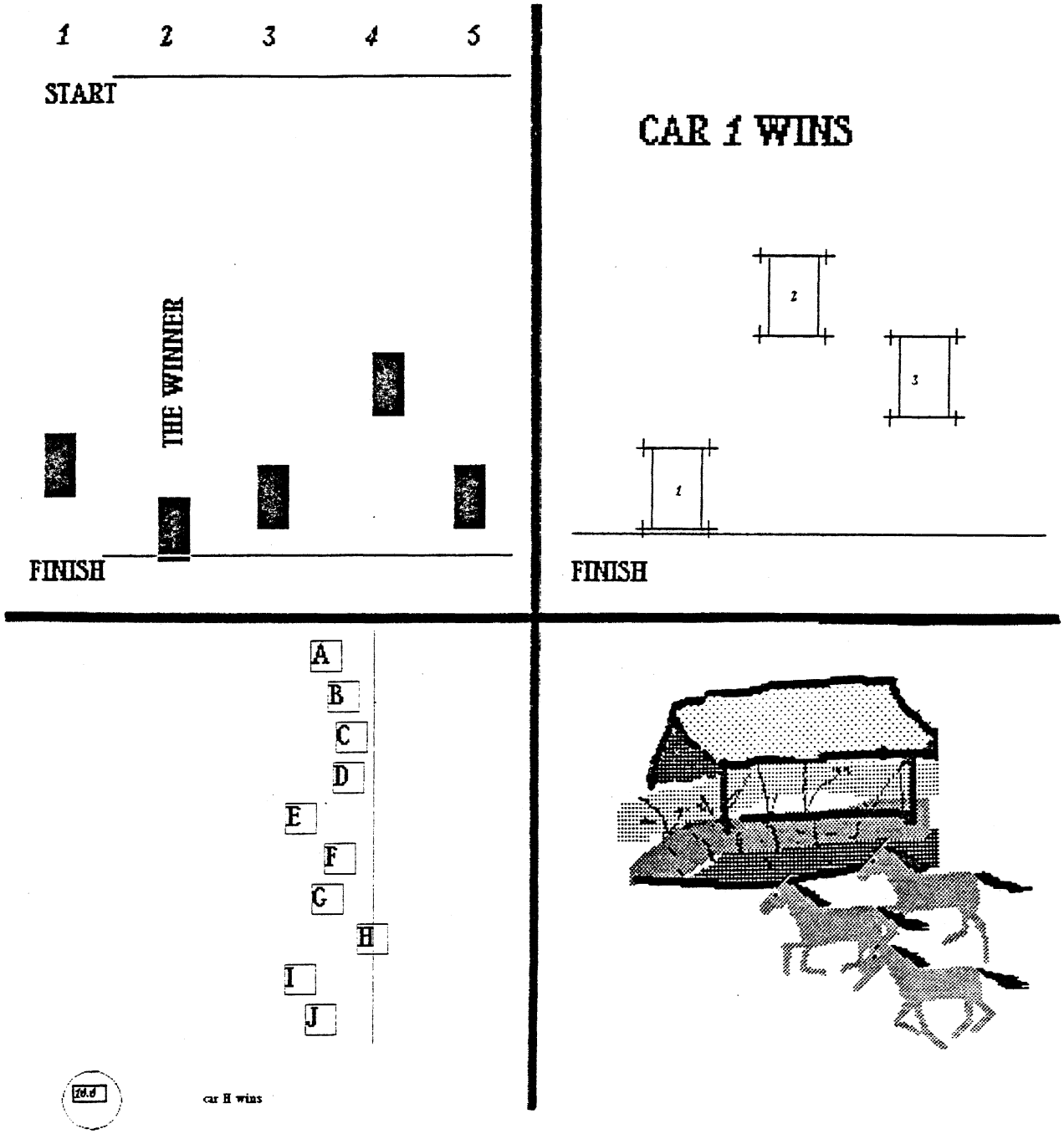


Figure 3. Races

We believe that in this simulation class, junior high school students designed a data storage mechanism for the first time. The available Smalltalk storage mechanisms proved not to be comprehensive enough for use as waiting-lines, so the kids designed their own. Their method (with several variations actually implemented) was a *waiting line* class: members of the class are lines that objects can join, leave, and get service from on a first in-first out basis; an object may also cut ahead of objects already in the line. The storage method is not concerned with details of the actual objects in the line, passing on responsibility by sending messages to the objects, and leaving it up to their class definitions to handle the actual drawing, undrawing, and moving. Hence a *line* can contain members of several different classes.

The students proceeded to study *Simpula*, a Smalltalk simulation of scheduling mechanisms in the Simula programming language [4]. *Simpula* operates primarily through scheduling pseudoparallel processes by means of a sequencing set. The set holds the quiescent processes sorted by desired time of activation. Associated with each process are the object itself, the time the object is scheduled to wake up and do something, and a message to tell the object what to do. This message is either constructed by the object when it runs, or is a default message. There is a system time which indicates the simulation's current progress. One object can be scheduled in connection with more than one event, each event invoking a different activity by the object. In order to understand the structure of *Simpula*, the students designed a class definition for *chained lists*; it was possible to add and delete items from such a list, where items were ordered with respect to a numerically-valued property (typically, this property represented *time*).

Up to this point, the students had been considering the problems of taking actual dynamic processes, such as their school cafeteria, a car wash, or parking lots at a shopping center, and building models in Smalltalk. The reverse process was illustrated in order to demonstrate to the students the kinds of information that can be obtained from an abstract model. A hospital simulation, built with *Simpula*, was shown to the students as an example of an already running machine model. The objects in the simulated environment are hospitals, departments, staff, patients who walk through corridors to get to department desks, waiting rooms, and staff attention. Each of these objects is able to describe, on demand, itself, its history and/or its immediate future.

One of the simulation groups proceeded to design a simulated amusement park. Included in the design were lines of cars waiting to enter the park, lines of people waiting to get on rides or buy cotton candy or be seated in a restaurant, and, of course, ticket counters, cafeteria-style food services, and souvenir shops. The students took into consideration such variations as rides in which all seats were filled and emptied at once (e.g., merry-go-round), and rides in which seats were filled and emptied continuously (e.g., jungle boat ride). An amusement park consists of a variety of events happening simultaneously; each can be modelled separately and combined in a number of ways that illustrate different scheduling models. The students' design work on this simulation will be expanded into a new illustrated project booklet. One student built part of the amusement park simulation, adopting the graphic layout from the example hospital simulation. A view of his work is shown in Figure 4.

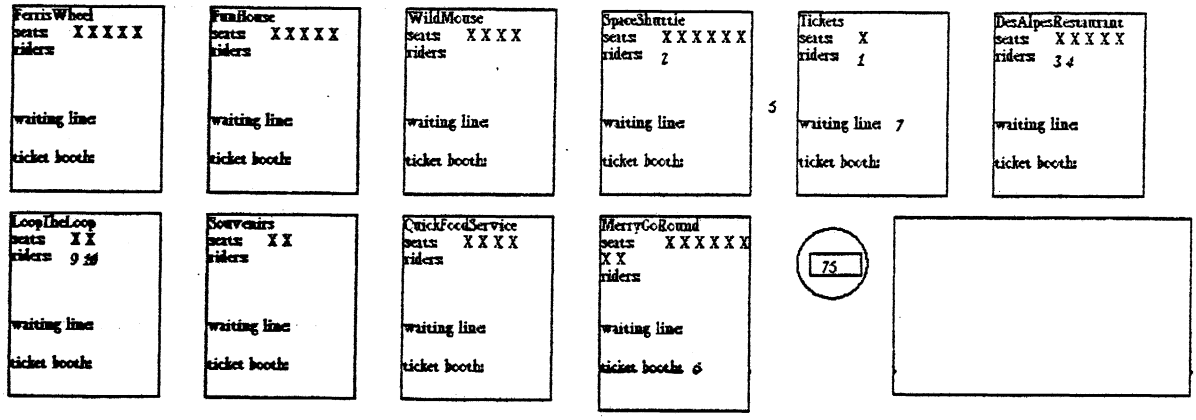


Figure 4. An Amusement Park Simulation. Each rectangular area represents a ride or service; x denotes the number of available seats. People are denoted by numbers, as riders, waiting or walking between rides.

Graphics

The number of students interested in using the Smalltalk systems was increased as a result of math class "field trips" to the Jordan resource center by over 400 students. During these trips, we gave the kids demonstrations of the use of the turtle robot, of a dial-up line to the school district time-shared computer, as well as of the Smalltalk graphic capabilities.

As a result of this increased demand, we presented two more courses: one on computer graphics and one on geometry. Each course attempted to provide some interesting pre-programmed tools that the students might use to create designs on a computer display screen.

The graphics curriculum is designed to give the students an awareness of the possibilities of high-resolution computer graphics. Basically, the students study models for line drawing, text presentation and editing, use of half-tones to give the impression of "color" or texture, and a variety of ways to make pictures move on the screen. We call the tools brush strokes (lines whose width's vary), polygons (string designs), and paint (shaded areas).

During the final class session, after an unfortunately short eight weeks, this class of fifteen students came up with a number of interesting ideas for a painting system: design and store new brush shapes, build new brushes by mixing already existing ones, use a scrolling menu of brushes in order to expand the virtual display space, (recursively) use the painting system to paint new brushes, and provide a variety of picture scaling and rotation operations. The ultimate goal in the course is to have the students design and build their own painting and animation tools.

Often a student from the graphics class teamed up as an "idea" person with another student, the "implementor", who could program, thus forming a very productive "research" team. Undoubtedly it was this sharing of ideas and knowledge that made the resource center the

stimulating place it proved to be. In examining the students' files after the study was completed, we found the same programs copied on several student disks, especially ones such as fantasy shown in Figure 5. In another case, a spacewar program written by one student showed up in another student's file, but modified to include a maze in which the ships travelled; and rockets that bounced off the maze walls.

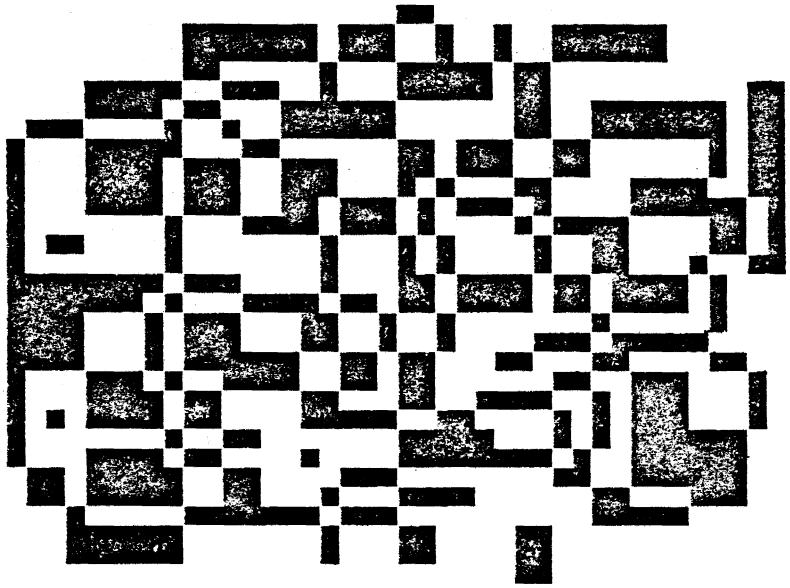


Figure 5. Fantasy

Geometry

In the geometry curriculum, the students are given class definitions for *point*, *line*, *triangle*, and *circle*. They are taught the basic Smalltalk notions of creating instances of the classes and of sending messages to these instances. For example, a student can create a *triangle* and then make designs by changing the positions of the vertices of the triangle. Several of the students' designs are shown in Figure 6.

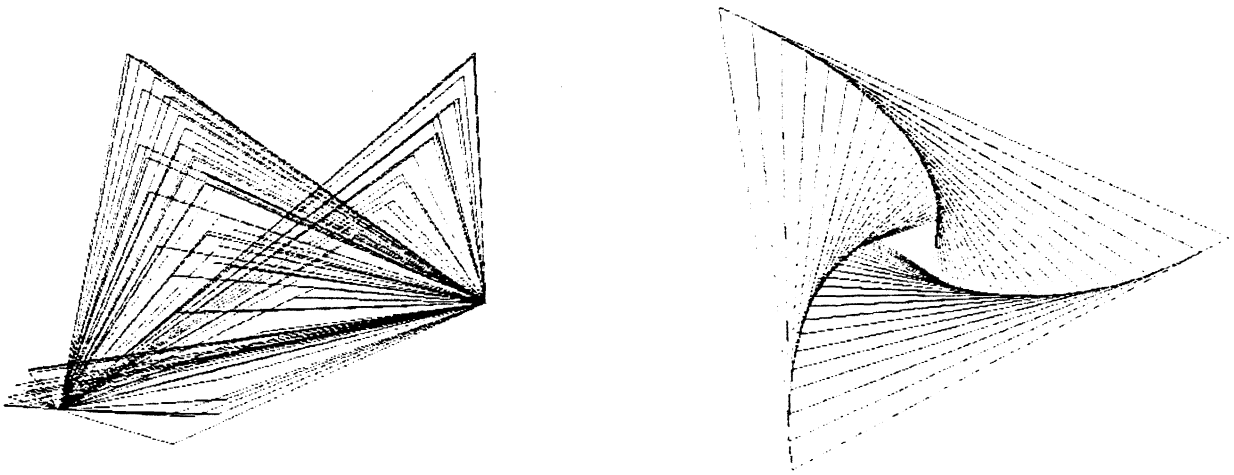


Figure 6. Triangle Designs

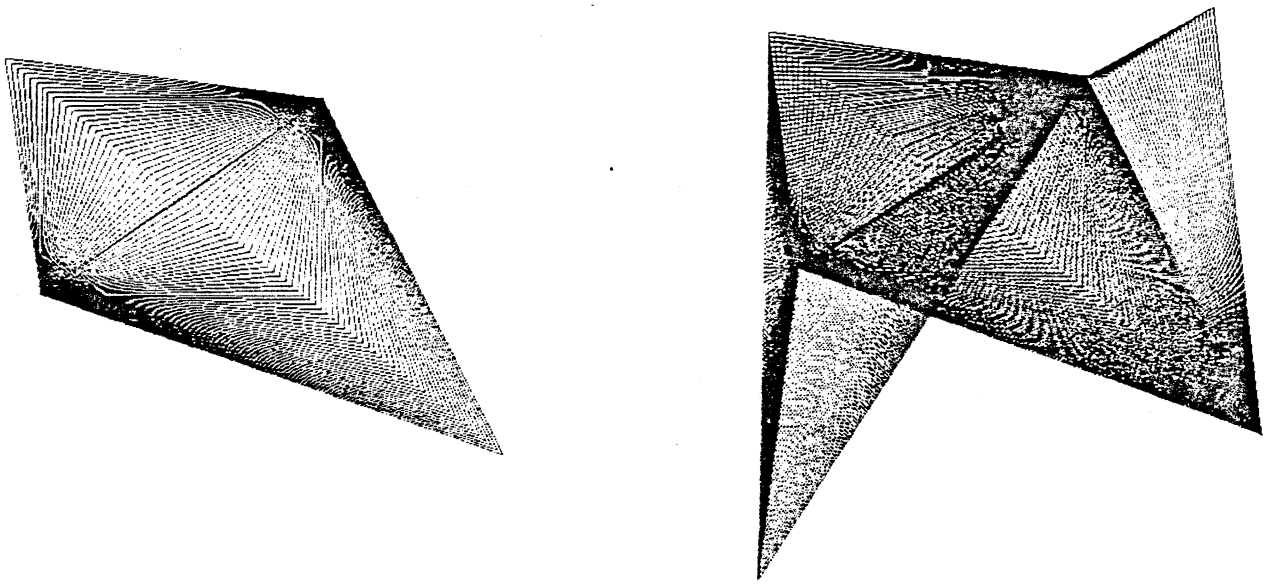


Figure 6. Triangle Designs (Continued)

The geometry curriculum pays particular attention to the problem of constructing a circle that circumscribes or inscribes a triangle. Members of the class *triangle* are created by pointing to three positions (vertices) on the display screen. Each *triangle* responds to a message of the form *bisect side <side identifier>* by creating a new *line*, i.e., the one that bisects the designated side. A *line* responds to messages of the form *intersect <line>* by returning the *point* of intersection. In order to circumscribe a triangle, we bisect two sides of the triangle and determine the intersection of the new lines. A *circle* is then formed with this *point* as center, and with radius equal to the distance from the *point* to a *triangle* vertex.

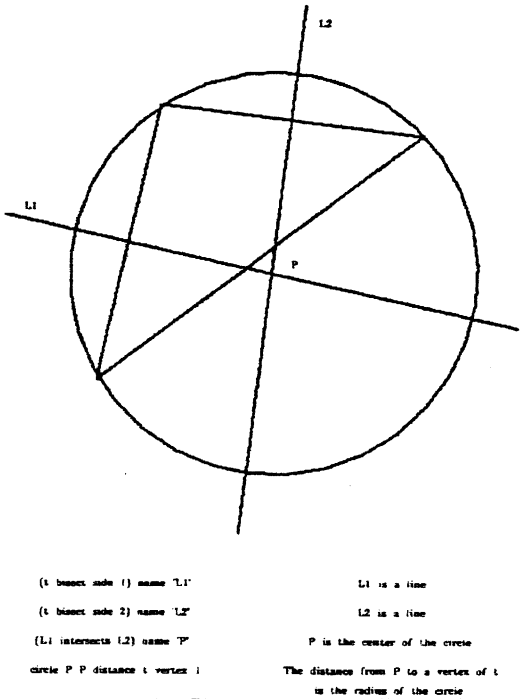


Figure 7. Circumscribe a Triangle

We inscribe a circle in a triangle by bisecting two angles of the triangle. The angle bisection is carried out by forming an isosceles triangle using the angle's adjacent sides; the bisector of the base of the new triangle also bisects the desired angle. The point of intersection of the lines that bisect the two angles is the center of the circle; the radius is the line from the point perpendicular to a side of the triangle.

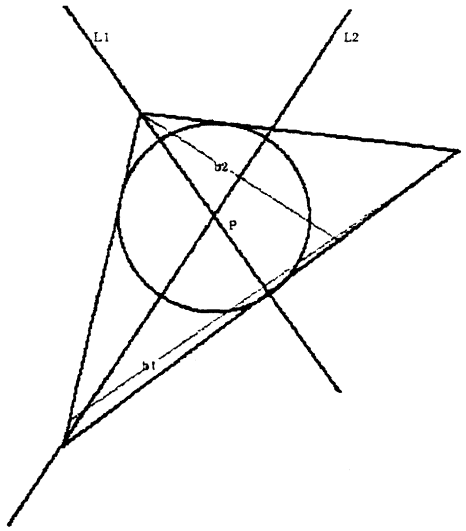


Figure 8. Inscribe a Circle in the Triangle

This geometry course is one example of how a trained teacher might make use of the Smalltalk environment to enhance classroom explanations. The particular sequence described here corresponds to a seventh-grade mathematics unit at Jordan.

EVALUATION

As reported on in [1], we have considered several methods for evaluating the students' individual accomplishments. We are interested, first, in whether or not the students learned how to use Smalltalk systems. We also wished to determine whether the casual access provided by the in-school resource center produces a qualitative difference in the use of the facilities.

Because the systems were located in their school, several students were able to involve their art and mathematics teachers in center activities, notably to include their Smalltalk work as a component of their regular class work. Many teachers considered the Smalltalk resource center as a kind of school library, and gave the students access to the center whenever regular class work was completed or whenever class problems could be done on the system. A number of the students received credit in their regular classes for work they accomplished on the Smalltalk system.

At Jordan, a tutor was often not available. The students had more time alone, so more ideas for extensions or methods for solving a problem could be attributed to them, rather than to the tutor. The lack of an adult tutor also seemed to encourage more cooperation and sharing of definitions among the students.

Videotapes of Smalltalk Projects

Once a student completes a project, we are interested in knowing the extent to which that student understands its components, that is, the class definitions and scheduling methods used. We attempt to measure this level of understanding by having each student both describe the project and use it, in expanded form or in a different context. Each student from the simulation class videotaped a verbal description and on-line demonstration. In order to "improve the videotape", the students were asked by the "cameraman" to make changes. Typical changes involved the

numbers of objects (e.g., a race for five rather than ten cars);

kinds of objects (e.g., substitute a swimmer for a car in a race simulation);

mixture of kinds of objects (e.g., a waiting line with trucks as well as cars);

placement of graphic information on the display screen (e.g., thicker lines for better visibility);

decision model for interaction among the objects (e.g., moving the race cars according to some fixed probability rather than random number selection).

Success in this task is purely a behavioral test of whether or not the changes were carried out. Each of the above kinds of changes requires the student to know who (which object) has what information, or who has control of which parts of the active events. These are kernel ideas in writing Smalltalk programs. This test is a learning experience that is fun; it provides a videotape for the class of their work. When carried out as a class-planned tape, changes for improvement are imposed by the students themselves, providing additional review.

The result of using this non-competitive evaluation method is now incorporated in a 30-minute videotape that we have put together to demonstrate a variety of Smalltalk projects. Each of the students was adept at making changes for graphic presentation and adding new capabilities to existing class definitions. They had good control over the class properties, knowing where changes had to be made.

The students were weakest on iterative methods used for scheduling objects, especially those in which the iteration counter is used as a variable in the computation. For example, one student was asked to change a ten-car race into a five-car one. Computation of each car position was a function of the initial display location, the position of the car at the starting line, and a spacing parameter that was used as the iterative increment. The student was not able to determine the increment. The simulation curriculum had not dealt in detail with iterative methods, mistakenly relying on the students' previous programming experiences.

The Jordan students were expected to learn the syntax of Smalltalk from *The Box Book*, a project book we described in [1], and from the longer instruction manual we prepared [3]. *The Box Book* seems somewhat oversimplified for students with strong backgrounds in Basic (that is, these Jordan students). However, the book does give these students a useful introduction to the new programming notation; it also offers a metaphor that helps the students understand the notion of classes and instances. These students showed some surprise at the ease with which they were able to implement programs they had previously thought impossible to do. For example, one student in the simulation class independently decided that the *bubbles* should burst. He also independently, and incorrectly, assumed they could not be made to burst in a computer simulation.

The Computer as Catalyst

An important, but difficult to measure aspect of our work at Jordan was the enthusiasm for the "research-oriented" approach to curriculum, in which students and teachers function as colleagues. With both students and teachers taking on roles as active learners, the students were free to fantasize, explore, and make mistakes. To quote the teacher assigned to the independent study center: "Since the curriculum was deliberately flexible, incorporating as it did the students' own ideas and projects, the students saw themselves as researchers and their work as valuable. The rigorousness of the curriculum arose then, as it should, out of their own self-imposed need for tools to carry out their ideas." [5]

ACKNOWLEDGEMENTS

Steve Weyer prepared and taught the geometry class; Steve, Dave Robson, and Al Borning assisted as tutors at Jordan Middle School.

REFERENCES

1. Adele Goldberg and Alan Kay, *Methods for Teaching the Programming Language Smalltalk*, (1977), Xerox Palo Alto Research Center, Technical Report SSL-77-2 (this report).

2. Radia Perlman, *Using Computer Technology to Provide a Creative Learning Environment for Preschool Children*, (1976), Logo Memo 360, MIT AI Lab.

3. Adele Goldberg and Alan Kay (Eds.), *Smalltalk-72 Instruction Manual*, (1976), Xerox Palo Alto Research Center, Technical Report SSL 76-6.

4. Dahl, Ole-John and Kirsten, Nygaard, "SIMULA--an ALGOL-Based Simulation Language", (1966), *CACM*, IX, pp. 671-678.

5. Joan Targ, Independent Study Center report to the principal of Jordan Middle School, (1976).

Table I. History of Smalltalk Classes for Children

Date	Number of Kids	Kids' Age	Class Schedule	Number of Systems	Where	Purpose
Spring 1974	4	12 yrs	twice a week, 1-1/2 to 2 hrs each session	2	Xerox PARC	Determine if Smalltalk can be taught to jr. high school students; develop initial teaching material.
Summer 1974	16 2 classes	9-10 yrs	5 times a week 1-1/2 hrs each session	4	PARC	Use of completed project book *(Box Book)*; trial with elementary school children.
	3	3-4 yrs	once a week 1 hour	1	PARC	Develop and try out exercises using the robot turtle running from a commercial minicomputer.
Fall 1974	5	12 yrs	twice a week 1-1/2 to 2 hrs each session	5	PARC	Peer teaching study: a 13-year old taught her own class using the *Box Book*. All sessions were videotaped.
Spring 1975	10	12-13 yrs	twice a week 1-1/2 to 2 hrs each session	8	PARC	Each student in the Fall class tutored a new classmate. Each group worked alone once a week.
Fall 1975	10	12-13 yrs	once a week 2 hrs each session	10	PARC	Animation Class--text editing to plan animated sequences that were then implemented in the Smalltalk/ Shazam system.
Spring 1976	19 in 2 classes	12-13 yrs	once a week for 40 minutes plus individual work sessions	3	Jordan Middle School	Simulation Class--build running simulations in Smalltalk and test the models with data collected from dynamic environments (e.g. the school cafeteria).
	15	12-13 yrs	once a week for 40 minutes plus individual work sessions	3	"	Graphics Class--study communication through high-resolution graphics; design painting or animation tool.
	9	12-13 yrs	once a week for 40 minutes	3	"	Geometry Class--study the relationships among points, lines, triangles, and circles.
	9	12-13 yrs	once a week for 40 minutes	3	"	Animation Class--implement simple animated sequences in the Smalltalk/ Shazam system.
Summer 1976	10	12-13 yrs	total of 80 hrs during the summer	10	PARC	Exploratory Experience-- students from Jordan continued their programming studies.
	3	13-15	10 hrs per week		PARC	Work Study program--high school students worked on applications programs.

Table II. Usage of Resource Center

simulation class	19 students, first part of course 10 students, second part of course average per student: 45.9 hours range: 14 to 150 hours
graphics class	15 students average per student: 18 hours range: 0 to 60 hours
geometry class	9 students average per student: 12.5 hours range: 4 to 22 hours
animation class	9 students who entered other classes 10 students not in other classes average per student: 8 hours range: 4 to 30 hours
"hangers-on"	12 students average per student: 11.5 hours range: 4 to 22 hours
visitors	27 junior high school students average per student: 1 hour 4 elementary school students teachers and school administrators