APPLYING THE LOGO ENVIRONMENT: LEARNING, DOING AND DISCOVERING THROUGH COMPUTERIZED LEARNING PROJECTS

M. A. Murray-Lasso

Computer-Aided Teaching Unit
Department of Systems Engineering
Graduate Division of the Faculty of Engineering
National Autonomous University of Mexico

Received: March 23th 2001and accepted the January 25th 2002

ABSTRACT

In this paper the Logo environment is applied to several original computerized learning projects, each of a very different nature: one for learning geometry generating numerous instances for verifying the results of a little known theorem of plane geometry, a second for learning the basics of the physical foundations of music through the construction, using the keyboard of the computer, of a little piano, and a third one to learn some aspects of Spanish grammar involving the conjugation of regular and irregular verbs. The projects were carefully selected to illustrate a wide range of possibilities and the flexibility of the Logo language and power of its instruction set and adaptive data structure. After introducing the main ideas of the Logo educational philosophy, a short introduction to the language is given followed by an introduction to the concept of a microworld. Next the idea of a computerized learning project is explained and the three projects mentioned above are presented. Complete listings of simple programs in LogoWriter are included together with some ideas for extending the projects in different directions. The projects presented in the paper are of a slightly more advanced nature than what usually appears in books on Logo. The paper is sprinkled with learning ideas that can be tried with groups of youngsters.

RESUMEN

En este artículo se aplica el ambiente Logo a varios proyectos computarizados de aprendizaje, cada uno de una naturaleza muy diferente: uno para aprender geometría generando casos particulares de los resultados de un teorema poco conocido de geometría plana, un segundo para aprender los fundamentos de la música a través de la construcción, utilizando el teclado de la computadora, de un pianito y un tercero para aprender algunos aspectos de la gramática española relativos a la conjugación de verbos regulares e irregulares. Se seleccionaron cuidadosamente los proyectos que se incluyen para ilustrar la gama amplia de posibilidades y la flexibilidad del lenguaje Logo, así como el poderío de su conjunto de instrucciones y de su estructura de datos adaptable. Después de introducir las principales ideas de la filosofía educativa Logo, se da una breve introducción al lenguaje seguida de una introducción al concepto de micromundo. A continuación se explica la idea de un proyecto computarizado de aprendizaje y se presentan los tres proyectos mencionados. Se incluyen listados completos de programas sencillos en LogoWriter junto con algunas ideas para extender los proyectos en diferentes direcciones. Los proyectos presentados en el artículo son de una naturaleza ligeramente más avanzada que la que normalmente aparece en los libros sobre Logo. El artículo está salpicado con ideas de aprendizaje que se pueden ensayar con grupos de jóvenes.

KEYWORDS: Learning environment, Logo, computerized learning project, Logo Writer.

1. INTRODUCTION

Logo is an educational philosophy with its own programming language whose psychological foundation is based on the theories of J. Piaget and whose principal promoter has been Seymour Papert, a former collaborator of Piaget and member of the Artificial Intelligence Laboratory, who is presently in the Media Lab at MIT. The best description of the Logo philosophy appears in [1]. The constructionist theory of learning (an extension of Piaget's constructivism, see [2] of Logo is roughly the opposite of behaviorismor programmed learning championed by Skinner. Exaggerating to provide contrast, in Skinner's approach the computer programs the student, while in Papert's the student programs the computer. Programming in the context of learning is to analyze, organize, and sequence knowledge with the purpose of teaching a process to a dumb machine that can only perform a group of certain elementary tasks. Which processes are considered elementary depends on the language in which the program is written. In the act of programming the student plays the roles of epistemologist and teacher in front of the machine. To get a program to function correctly it is necessary that the programmer organize and sequence properly all the steps in the process being programmed. This organizing forces the student to appropriate knowledge instead of simply memorizing what her teacher communicated. The formulation of hypotheses, exploration, experimentation, and verification, in other words, the practice of the scientific method are very important in the Logo philosophy. The goal is that much of the learning be acquired through discovery during the exploratory phase. The various programs written by the student are documents that reveal the functioning of her mind. They allow the teacher to see the paths that the student's mind takes during the search for the objective of the learning activity or the solution of a problem.

Logo's educational philosophy favors informal learning. The preferred road to learning mathematics is that used by a child to learn the mother tongue. He certainly does not take courses in classrooms: Elementary Talking, Intermediate Talking, Advanced Talking. The recommended method is to learn to talk in the midst of the family, learning the words that the child requires to deal with situations such as asking for food and toys. The fact that most children of normal intelligence learn to speak the mother tongue and do it well, is strong evidence that the mind of a child is designed to learn and there is no need to organize knowledge in a special way for the child to assimilate it. Papert opposes the theories that preach that there are children for which mathematics are negated. He considers that all children are latent mathematicians if only they are placed in a proper environment. All of them are capable of learning. He uses as an example of the importance of the environment in learning, the acquisition of French in American schools; and describes the failure of American children when trying to learn French in school by taking lessons versus the success when the same children learn it by traveling to France and staying there for a few months. He explains that the great difference has to do with the need to speak the language every day, 24 hours a day, simply to carry on with life in contrast to trying to learn it by memorizing grammatical rules and inventing compositions two hours a week in a French as a second language formal course in school.

Papert is so much against school education that he predicts its demise. He explains that in school education the teacher is constantly judging the work of the students giving them passing or failing grades: a tick or a cross. What Papert considers correct is to measure how close is a piece of work to achieving the objective set by the student, then discovering the point at which the program deviates from what is expected, in order to introduce a change that improves the final result. It is a process of successive approximations that takes the proposed solution closer and closer to the goal. It is too simplistic to judge a program as right or wrong; it is much better to say that it still has some "bugs" and that it can be improved by their removal.

In this paper we will exhibit three examples of the use of a Logo environment applied to learning geometry, music and Spanish grammar. Space does not allow us an extensive presentation of the Logo language. Nevertheless, we provide a brief introduction to the language so that the programs that we will include in the paper are better understood. This allows the reader a taste of the possibilities of the Logo educational environment.

2. THE LOGO LANGUAGE

Before we introduce some concepts it is convenient to describe a simple session using LogoWriter. The following and the examples in the paper are best understood if you have a copy of the software available on a computer. If you don't have a copy of LogoWriter, try borrowing one from a library. Once we execute the program (called LogoWriter) and we choose a NEW PAGE from the initial menu, we are presented with a screen which has a pink horizontal line near the bottom of the page and below the line there is a blinking cursor. The area below the line is called the command area and the one above is called the work area. (See Fig. 1) An image of a turtle as seen from above appears in the center of the work area. If we type FORWARD 50 and press the largest key of the keyboard, placed on the right border of the largest portion of the keyboard, (we will call this the <RETURN> key) we can see that the turtle moves vertically in an upward direction about halfway towards the top border. Now type RIGHT 90 and press <RETURN> (every line you type must be followed by <RETURN> which signals the language to execute the instructions on the line.) What you see is the turtle rotating 90 degrees to the right and facing towards the right edge of the screen. Now type FORWARD 50 and the turtle moves horizontally in the direction it is facing a distance equal to the one it moved upwards previously. Type RIGHT 90 and the turtle rotates facing towards the bottom of the screen. Type FD 50 and the turtle moves downward the same distance it moved upward previously.

Notice that FD is an abbreviation of FORWARD to economize on typing. Type RT 90 (notice RT is the abreviation of RIGHT) and the turtle rotates to face the left border of the screen, because the order to turn right is relative to the orientation of the turtle and not to the user. (Imagine yourself to be the turtle, that is what we ask the children to do). Finally type FD 50 and you have completed a square. If you add a final RT 90 the turtle will face upward which is the orientation with which it started.

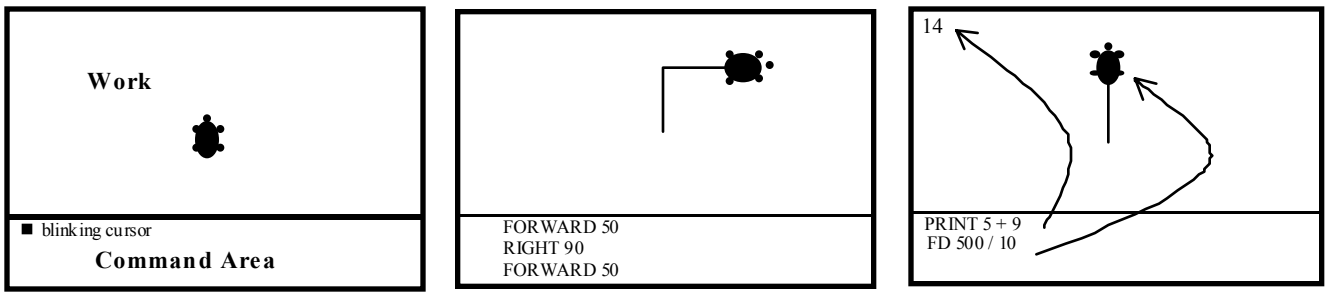

Figure 1.The disposition of the screen in LogoWriter.

Play around for a while with the language. Make the turtle turn angles different from 90 degrees. Try the instruction BACK (abbreviated BK) to move the turtle in reverse. The instruction CG (meaning clear graphics) erases what you have drawn on the screen and returns the turtle to the center of the work area facing upward. The instruction PU (meaning pen up) raises the pen and the turtle moves without drawing; to draw again the instruction PD (meaning pen down) lowers the pen and the turtle draws as it moves. To make the turtle invisible use HT (meaning hide turtle). To make the turtle visible use ST (meaning show turtle).

With LogoWriter you can calculate. Type PRINT 5 + 9 (notice the spaces separating PRINT and 5; 5 and +, and + and 9) and you will see in the work area the number 14 (which is the result of 5 + 9). Type FD 500 / 10 (notice the spaces separating FD and 500, 500 and /, and / and 10) and you will see the turtle move in the direction it is facing a distance equal to the 500/10, or 50, the same as it moved when drawing the square above. Play some more with the language. Try subtraction with the symbol "–", multiplication with "*". To erase symbols typed with PRINT (abbreviated PR) use CT (meaning clear text).

The Logo Language is one of the tools suggested as a programming language for the environment of the Logo Philosophy. As a language is derivative of LISP, a language utilized by the Artificial Intelligence community, Logo has some powerful elementary processes and instructions geared mainly to learning mathematics, especially geometry. There are orders that deal with Cartesian coordinates, change the color of the turtle and of the lines drawn by it, perform arithmetical calculations and evaluate the most common elementary mathematical functions. There are conditional instructions that allow complicated logical decisions. There are also instructions to process strings and lists of strings and lists of lists. Some versions of Logo have instructions to process text, generate music, play various instruments, handle multimedia, manipulate several turtles at the same time, and handle processes by means of which animation can be produced.

To help the reader understand the rest of this paper we provide a very brief introduction to the Logo language for a PC, a version called LogoWriter [3]. [4] is a short introduction to the language written by the author. In LogoWriter the instructions can be written either in capital or small letters or any combination of them. Table I shows the meaning of some of the more common instructions. The instructions are classified in two categories: commands and functions. The commands perform some action such as moving a turtle forward or writing a name or number in the screen; functions on the other hand are first evaluated and then they transmit their value to another function or command to its left in the text of the program as a part of an arithmetical o symbolic expression. Both commands and functions may have parameters which may be numbers, words or lists, a feature that gives Logo great flexibility. All these points will be clarified through examples.

Table I. Some elementary instructions in logowriter.

INSTRUCTION	ABREV	ACTION AND TYPE
Forward 50	Fd 50	The turtle advances in the direction pointed by its head the number of turtle steps indicated (50 for the example.) Command
Back 50	Bk 50	Same as the previous but in the opposite direction. Command
Right 30	Rt 30	The turtle rotates to the right using the pen as axis the number of degrees indicated (30 for the example.) Command
Left 90	Lt 90	Same as the previous but to the left. Command
	St	Shows the turtle (makes it visible.) Command
	Ht	Hides the turtle (makes it invisible.) Command
	Pd	(Pen down) The turtle is ready to draw as it moves. Command
	Pu	(Pen up) The turtle is ready to move without drawing. Command
	Cg	(Clear Graphics) Erases the drawings in the screen and places the turtle in the center of the work area facing up. Command
Make “x 25		Assigns the variable named x the value 25. Command
Print :x Print “Peter (Print :x :y)	Pr :x Pr “Peter (Pr :x :y)	Writes in the work area the value of object x. Command Writes in the work area the name Peter. Command Writes in the work area the values of objects x and y. Command
Sin 45		Calculates and outputs the value of Sine of 45 degrees. Function.
Cos 45		Same as previous for the Cosine. Function.
Arctan 2.5		Calculates and outputs the value of arctan of the quantity indicated (2.5 for the example.) The result is given in degrees. Function
First :a		Outputs the value of the first element of the list a. Function
Last :a		Same as above for the last element. Function
Butfirst :a	Bf :a	Outputs the list obtained from a after removing the first element. Function
Butlast :a	Bl :a	Same as previous removing the last element. Function
		In the last four instructions instead of lists, words can be used, in which case reference to characters, rather than to elements, is meant.

Putfirst :a :x	Pf :a :x	Builds and outputs the list x to which it attached a as first element. Function
Putlast :a :x	Pl :a :x	Same as previous except a is attached as the last element. Function
Slow turtle		Slows down the speed at which the turtle moves. Command
Item :n :x		Extracts and outputs the nth element in the list x.
Empty? :x		Outputs true if list x has no elements, else it outputs false. Function
Round :a		Outputs the integer closest to the number a. Function
Setsh 25		Gives the turtle the shape indicated (25 in the example) from the table of the shape file of LogoWriter. Command
Random :n		Outputs a random integer selected from a population uniformly distributed between 0 and n – 1. Function
Tone :f :d		Outputs a sound with frequency f and duration d. Command
Search :a		Searches the text in the work area for the string stored in variable a and positions the cursor on the first one it finds. Command
Cut		Eliminates the text that is marked in the work area and places it in a special storage area. Command
Paste		Places in the work area, from the cursor on, whatever is stored in the special storage area mentioned in the previous point. Command
If exp1 [list]		If exp1 is true, Logo executes the instructions in list, else execution passes to the next instruction. Command or Function depending on the contents of list.
Repeat 10 [list]		Repeats the number of times indicated (10 in the example) the execution of the instructions in list. Command

Among the fundamental concepts in Logo is the concept of a *variable* and the concept of a *procedure.* With variables, through *names* one can handle quantities, words or lists that can change. As in other computer languages, numerical variables can be considered as little boxes with names that store numbers which can change according to circumstances. For example if one is manipulating squares, in a variable whose name is *side* one can store the length of the side of a specific square. The area of the square will then be the product of the value of *side* multiplied by it self. In Logo this product is written: side * :side. The need to introduce the symbol ":" stems from the fact that in a variable we can also store words, or lists and it is necessary at all times to distinguish between the word by which a variable is known, that is the name, and the number, word or list that is stored in the box under that name. When we refer to the name we use as a prefix the opening double quotation" without closing it, and when we refer to the value (or *content* of the box) the prefix used is ":".

There are in modern computing languages mechanisms such that when different persons write procedures, they can use whatever names they see fit without concerning themselves whether those names are used elsewhere giving rise to the possibility of confusion The appellatives *local variables* and *global variables* are often associated with this mechanism. In LogoWriter the variables appearing in the first line of a procedure (also called *inputs*) are local variables that are only valid while the procedure is being executed and they disappear when execution ends. All other variables are global.

Logo also allows us to introduce new instructions which we can give a name by combining the language's original instructions (called *primitives, parameters* or *arguments*) and defining *procedures.* The first line of a procedure starts with the word **to,** followed by the name we give to the procedure, followed by none, one or several inputs or parameters. In LogoWriter there is a restriction for the name of a procedure: it must not coincide with a primitive. All procedures end with a line with the instruction **end.** Once a procedure has been defined it can be used elsewhere as if it were a primitive. As it is done with primitives, procedures can have parameters to provide values for variables and thus make them more flexible. As an example of a procedure we can exhibit the following to draw squares of different sizes:

```
TO SQUARE :SIDE
REPEAT 4 [ FD :SIDE RT 90 ]
END
```

Here we have introduced a new instruction REPEAT :n whose effect is to repeat n times the execution of all the instructions that are enclosed in the pair of brackets opening after the parameter n. The value of the procedure input :SIDE is taken as the value assigned to the variable SIDE that appears in the second line of the procedure.
If we now write SQUARE 50 in the command area, a square with a side of 50 "turtle steps," identical to the square drawn previously is drawn in the work area. If we write SQUARE 75 a square of 75 turtle steps per side is drawn. Another way to use a procedure is to include its name and parameters in a second procedure; the square will be drawn at the moment the name is encountered in the execution of the second procedure. (In order not to complicate the discussion we are ignoring some things such as the color of the drawing, the color of the background, and so forth.)
Logo's syntax is very simple, any legal instruction must start with a command. Several commands can be written in a line separating them with one or more blank spaces. Blank spaces can be inserted anywhere that is desired except in the middle of a primitive, word or number. If a command has inputs or parameters one can provide numbers, names or lists or, if desired, functions that report numbers, names or lists. The functions will be evaluated and the value treated as if it had been written. Since Logo has binary arithmetic operators such as "+", "-", "*", "/", "=", ">", "<", (all of which are functions or relations with two arguments), as well as logical operators such as AND, OR, NOT, the expressions with operators can be treated as functions of the arguments. For example, when writing the expression

```
fd 100 * (sin (:x + :y) + cos(:x + :y))
```

When executing the expression the values of the variables x and y are added, giving a number, the sine of this number is evaluated giving another number; the steps are repeated for cosine and a third number is calculated; the second and third numbers are added giving a fourth number which is multiplied by 100 and the result which is also a number, is used as an input to the instruction **Fd.** Therefore the turtle advances a certain distance equal in turtle steps to the last number calculated. Parentheses are used to change the priority of the operations.

The priority of the operations is a possible source of confusion, especially when the logical operators are used. Multiplication and division have the same priority, which is greater than that of addition and subtraction. Therefore :x + :y / 2 should be interpreted as :x + (:y / 2) and not as (:x + :y) / 2. In case one wants the latter, one has to introduce parentheses to force, first, the addition and then the division. While we acknowledge having left many topics untouched, we recommend the reader consultation of the language manuals to clarify the points not covered in these brief comments. (For more details on the topics of this sectio see [3], [4])

2. MICROWORLDS

One of Papert's central ideas is the concept of a *microworld,* which is an ideal computational educational environment in which certain entities are defined together with their properties, and inside which the student can manipulate the entities to achieve certain objectives. The most popular microworld is the geometrical world of the turtle. The defined objects are the turtles. LogoWriter has four turtles and has instructions which can be addressed to any of them, several or all of them. The instructions to do this are **tell** and **ask**. Each turtle is characterized by its state, formed by a list of values including the coordinates of the center of the turtle (place where the turtle has the pen,) its heading, the form of the turtle, the color of the turtle, whether the turtle has its pen down or up, whether it is visible or not, etc. The turtles can interact with each other, for example one can chase another, and actions can be taken depending on events such as: the moment in which a turtle crosses the line drawn by another turtle. These facilities allow interesting dynamic processes in the microworld of the turtles. A very different microworld is provided by the instructions to handle lists, words and text in the screen's work area. In this microworld one can work with

topics such as Spanish Grammar. An example is the conjugation of regular and irregular verbs of Spanish. Lists of verbs whose conjugation follows the same rules can be assembled and the rules can be programmed into programs that automate the conjugation of families of verbs in different tenses. One can build an artificial expert that can print or display how a given verb is conjugated in a particular tense. A third possible microworld in LogoWriter can handle some simple musical topics. Through the use of the primitive **Tone** and an elementary knowledge of the frequencies of the notes, one can generate simple melodies or experiment with no conventional musical scales like the one used in Sound 13 or random music. It is possible to convert a piece of the keyboard into a simple musical instrument in which the notes are played by pressing certain keys. Although it is not possible to compete with specialized programs for musicians that include sequencers, synthesizers and so forth, it is possible with such a program to teach some elementary facts about music. (See [5] and [24]. For more information on the topics of this section see [15]).

3. THE COMPUTERIZED EDUCATIONAL PROJECT

One way to use microworlds to help in the teaching-learning process is to work with *computerized educational projects*. The experience of the author and other researchers see [6] differs from what Papert had been claiming: it is sufficient to teach a few Logo instructions and the students will by themselves discover mathematical theorems and other facts. Experience indicates that it is necessary to define specific learning-objectives and provide teacher supervision to maintain the students on a road that leads to the objectives, helping then to get unstuck when problems arise. We do not recommend to go to the extreme of the lecture in which the teacher is practically the only one who speaks during class. Nor do we recommend that the teacher solve the totality of the problems that arise for each student. The right amount of help seems to be to redirect them whenever the direction steers far from the direction of the goal that has been set for the lesson, to get them unstuck when they no longer seem to be able to progress towards the goal by providing hints and suggest ideas to test or materials to read. The computerized educational project when properly formulated so it corresponds to the students' level of knowledge and abilities can be a practical way to implant Logo as a computer-aided learning environment in various topics.

In a computerized educational project one starts by defining a problem to be solved in a narrow topic. For example to turn a piece of the computer's keyboard in a little piano of an octave and a half range. The demonstration that the objective has been achieved will consist in the fact that when one presses the designated keys in proper order the familiar musical scale will sound. We are only too happy to admit that the quality of sound of the device will be far from that of a real piano. It will not be able to sustain a note as in a real piano, but we will be able to recognize some simple melodies. The project will force students to research several items such as the frequencies of the notes of the white and black keys of a piano. This may require a visit to the school library in search for a book on the physics of musical sounds. Participants will also be required to learn enough about the computer language to be able to detect which key is being pressed in order to be able to produce the right sound. The students may learn about lists or arrays and procedures to coordinate the whole operation. In short, students will be faced with learning and thinking.

In contrast with other ways to encourage learning for which the teacher reviews the work and grades it, computerized educational projects generally have a ready-built evaluation instrumentation since the student is able to see for himself if the objective is achieved. For example, in the case of using the turtle to draw a house with a given shape, the student can evaluate without help if the final drawing resembles the one he had set his mind to. No more crossing to indicate that an answer is wrong. In a computerized educational project a piece of work is neither right nor wrong, it may resemble to a lesser o grater degree the objective that was set. When it does not, the thing to do is analyze the program to detect the first point at which it starts to differ significantly from the model desired in order to provide corrections to the program followed by test runs that will bring the final product into an acceptable resemblance. Repetition of the process in general brings convergence. When convergence fails it is time for the teacher to intervene and suggest an instruction or idea, or suggest some reading material or consultation with outside experts. What has been mentioned is very similar to the way problems are solved in the professional practice of scientific research and technological development. The experience acquired by the students will help them to assemble a kit of tools for their future careers. (For more information on the topics of this section see [19].

4. LEARNING GEOMETRY WITH LOGO

Most material written about Logo emphasize its possibilities for learning geometry at an elementary level [14]. [7] is a notable exception. They tend to concentrate on drawing simple figures such as equilateral triangles, squares, polygons, stars, and recursive variations. Since there are many publications treating these topics, in this paper an application at a higher level (high school) will be given. G. Polya, an important 20th Century mathematician and educator, who wrote several books on the teaching - learning process in mathematics [27]. considers that mathematical discovery has many of the ingredients of experimental sciences such as physics. In spite of the fact that the central topic of mathematics is proof, in general, theorems are first conjectured, then tested in particular cases and finally when the discoverer is convinced of its truth a proof is attempted. There is a nice little theorem in Geometry known as Von Aubel's Theorem that says: "Given a convex quadrilateral, if upon each of its sides a square is constructed on the exterior of the quadrilateral, the lines joining the centers of the squares corresponding to opposite sides have equal lengths and are orthogonal to each other." [29]. Fig. 2 illustrates the theorem. Before attempting a proof of the theorem, it is a good idea to ask the students to test it for several particular instances. This can be done by hand using conventional geometric instruments. Logo lends itself to automate the tests since it has instructions to both draw figures as well as to "make measurements" of lengths and angles.

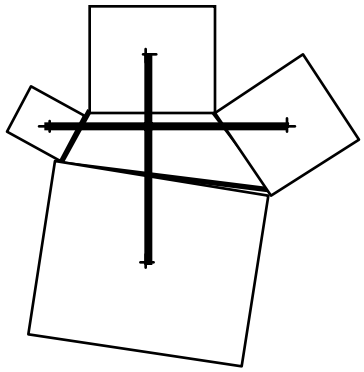

Figure 2. Von Aubel's Theorem

It turns out that the theorem, although enunciated for convex quadrilaterals, can be proved for any quadrilateral, as long as the squares built lie on the same side of the perimeter when adjacent vertices are traversed in order in a specific direction [28]. Hence, the testing will be done for arbitrary quadrilaterals. The objective of the project is to write a LogoWriter program that will generate random quadrilaterals (which will fit in the work area of LogoWriter) and draw the squares constructed on the sides and the lines joining the centers of the squares corresponding to opposite sides; then measure the lengths of these lines and their headings to ascertain that they are equal and orthogonal.

We exhibit a LogoWriter program that does this

```
to vonaubel
ht
make "x1 40 - random 80 make "x2 40 - random 80 make "x3 40 - random
80
make "x4 40 - random 80 make "y1 40 - random 80 make "y2 40 - random
80
make "y3 40 - random 80 make "y4 40 - random 80
make "x5 :x1 make "y5 :y1
cg ct pu setx :x1 sety :y1 make "i 2
```

```
repeat 4 [ seth towards (list thing (word "x :i) thing (word "y :i))
    pd setc 1 make "d distance (list thing
    (word "x :i) thing (word "y :i)) square
    :d setc 2 ad :d / 2 pu rt 90 fd :d / 2
    make (word "w :i - 1) xcor make (
    word "z :i - 1) ycor
    bk :d / 2 lt 90 pd fd :d / 2 make "i :i + 1]
pu setx :w1 sety :z1 seth towards (list :w3 :z3) make "r1 heading
pd setc 3 make "h1 distance (list :w3 :z3) fd :h1
pu setx :w2 sety :z2 seth towards (list :w4 :z4) make "r2 heading
pd make "h2 distance (list :w4 :z4) fd :h2 pu
(pr "lengths round :h1 round :h2)
make "r :r2 - :r1
if ((abs (:r2 - :r1)) > 180) [make "r 360 - abs :r]
(pr "|difference in headings | abs round :r)
end

to square :side
repeat 4 [fd :side rt 90]
end

to abs :x
if :x < 0 [output (-1) * :x]
output :x
end
```

The rough explanation of the program is as follows: The main procedures name is **vonaubel.** It calls two auxiliary procedures: **square** with one input (the length of the side) which draws a square, and **abs** with one input, which reports the absolute value of its input.

The first thing that is done is to hide the turtle (**ht**). Then 8 random integers between 0 and 79 are generated to assign to x1, x2, x3, x4, y1, y2, y3, y4, which are the x and y coordinates of the 4 vertices of the quadrilateral, random values between –39 and 40. The numbers are designed so that the vertices are well within the work area of LogoWriter with room to draw the squares. In order to avoid exceptions in the algorithm, vertex 5 is given the same values given to vertex 1. Next the graphics and text are cleared in the work area, the pen of the turtle is lifted, and the turtle placed (without drawing since the pen is lifted) in the first vertex of coordinates x1, y1. A counter named i is initiated at value 2 and an iteration including all the instructions inside the brackets is performed. In each of the 4 cycles of the iteration the following is done: The pen is lowered so the turtle can draw; color 1 is chosen (so the quadrilateral, the squares and the lines joining the centers of squares of opposite sides are drawn in different colors) the heading of the turtle is made to face the next point (in the first cycle to point x2, y2). The distance from the turtle to the next point is stored in variable d; a square with side d is drawn (in color 1); now the color is changed to color 2 and the turtle is advanced d/2 (half the length of the side of the quadrilateral that lies between vertices (x1, y1) and (x2, y2)); when the turtle gets to the half point its pen is lifted so that it stops drawing, the turtle is turned 90 degrees to the right, advanced a distance d/2 (to get it to the center of the square) and when it gets there we store in variables w1 and z1 the x and y coordinates of the turtle, which correspond to the coordinates of the center of the square. Next we backtrack by moving the turtle backward a distance d/2 and turn it 90 degrees to its left. Now the pen is lowered (it still has color 2) and the turtle is advanced a distance d/2 completing the first side of the quadrilateral. Finally before going on to the next cycle the counter "i" is increased by one. The instructions are repeated 4 times. Since we started the iteration with a value for "i" equal to 2, in the last cycle i will have the value 5, but since we started by copying the point (x1, y1) into the point (x5, y5), no special instructions are needed for the last point. By the time we finish the iteration we have in storage the centers of the squares in points (w1, z1), (w2, z2), (w3, z3), (w4, z4). Now to draw the lines between centers of squares of opposite sides we lift the pen of the turtle to avoid drawing and place the turtle in point (w1, z1) and set its heading towards the point (w3, z3);

we store the heading in variable r1. We put the pen down and change the color to *color 3,* calculate the distance from the turtle position to point (w3, z3) and store it in variable h1, and proceed to advance the turtle the distance h1. We do the same operation for points (w2, z2) and (w4, z4), storing the heading in r2 and the distance between the points in h2. Finally we print in the work area the message "lengths" followed by the values of h1 and h2, rounded to the next integer, and the absolute value of the difference in headings | r2 – r1 |. Since headings are expressed from 0 to 359.9999 degrees, we first check if the absolute value of the difference in headings is larger than 180 degrees, in which case we change the difference to 360 minus the absolute value of the difference, so that the value of the difference will always be less than or equal to 180 degrees.

This project keeps high school kids occupied for several one-hour class sessions till they get the program to work correctly. Notice that several special functions of LogoWriter such as **towards** and **distance** have been called into use. In one place there is a sophisticated trick to handle the "subscripts" of the "arrays" x, y, w, z. Logo allows the name of a variable to be built with functions that handle symbols. The instruction

```
make (word "x :i) xcor
```

is an extension of the simpler instruction

```
make "a 5
```

which simply assigns the value **5** to the variable named **a**. In the first version the first argument of the function **make** is another function that concatenates the letter **x** with the current value of the variable named **i**. By varying the value of i in the iteration we can change the name of the variable to which the assignment is made and thus obtain the same effect as if we were handling an array, in spite of the fact that LogoWriter does not support arrays. It is unlikely many students will write their programs like the one we exhibit here (although I say this with some reticence; many of my students are far better programmers than I ever was; young people seem to have a special talent for things having to do with computers). In general there will be a wide variety of styles. Some students will not use iteration, instead they may write the instructions repeatedly for each one of the sides of the quadrangle and may even draw each square independently without resorting to a separate procedure. Some will not even name the variables in an organized fashion, maybe calling the coordinates of the first point (a, b) and of the second point (p, m). It is up to the instructor to give them hints and ideas and provide needed information at the proper times. What I do assure is that if the project is carried to completion the theorem and its significance will be learned together with many geometrical facts. Also tools will be developed for the programming language. They will help the students to solve other geometrical problems, and in some cases other types of problems.

By running the procedure **vonaubel** repeatedly a student experiments with many cases of figures to which the Von Aubel Theorem applies. Many other geometrical theorems can be treated in a similar way. The teacher can challenge the students to draw a quadrilateral that does not obey the Von Aubel Theorem, and they will be surprised that they can not (obviously the same challenge can be raised for any other theorem within its range of validity with the same result.) In Fig. 3 we exhibit two typical results of runs of **vonaubel.** The quadrilateral on the left is not convex. The one on the right is convex but the vertices circle the center of the figure clockwise, hence the squares on the right side are really drawn "towards the inside of the quadrilateral."

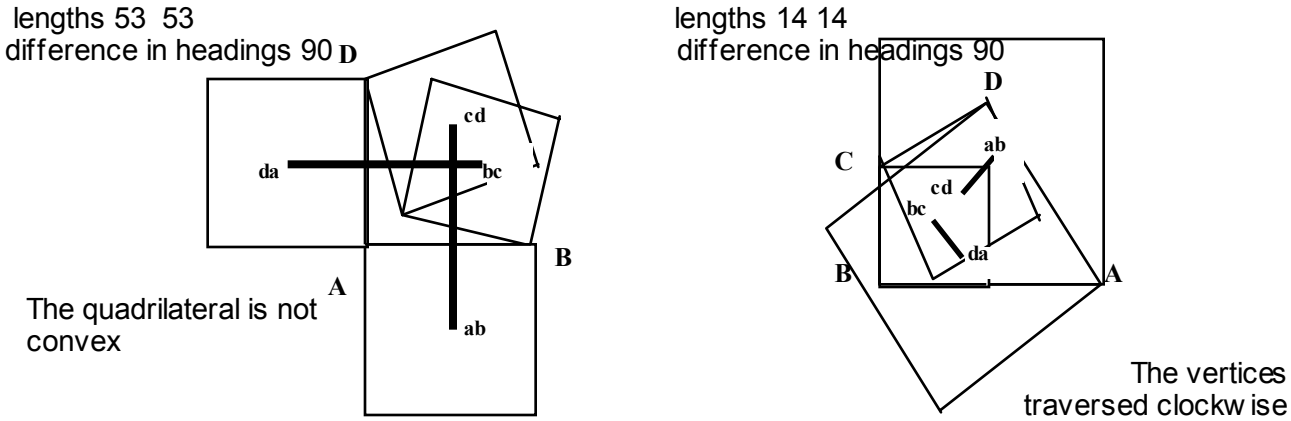

Figure 3. The program does not write the letters, they were added for clarity. See legend for Fig. 4. Although the program draws in color, for black and white reproduction the quadrilateral and the two lines between the centers of opposite squares have been emphasized.

Since the theorem is true for any quadrilateral, as part of the strategy to drive home the generality of the theorem, the teacher can suggest applying it to a triangle considered as a quadrangle, one of whose sides has zero length. Another suggestion can be to apply the theorem to a quadrangle that has collapsed into a broken line and into a straight line. These three cases are illustrated in Fig. 4.

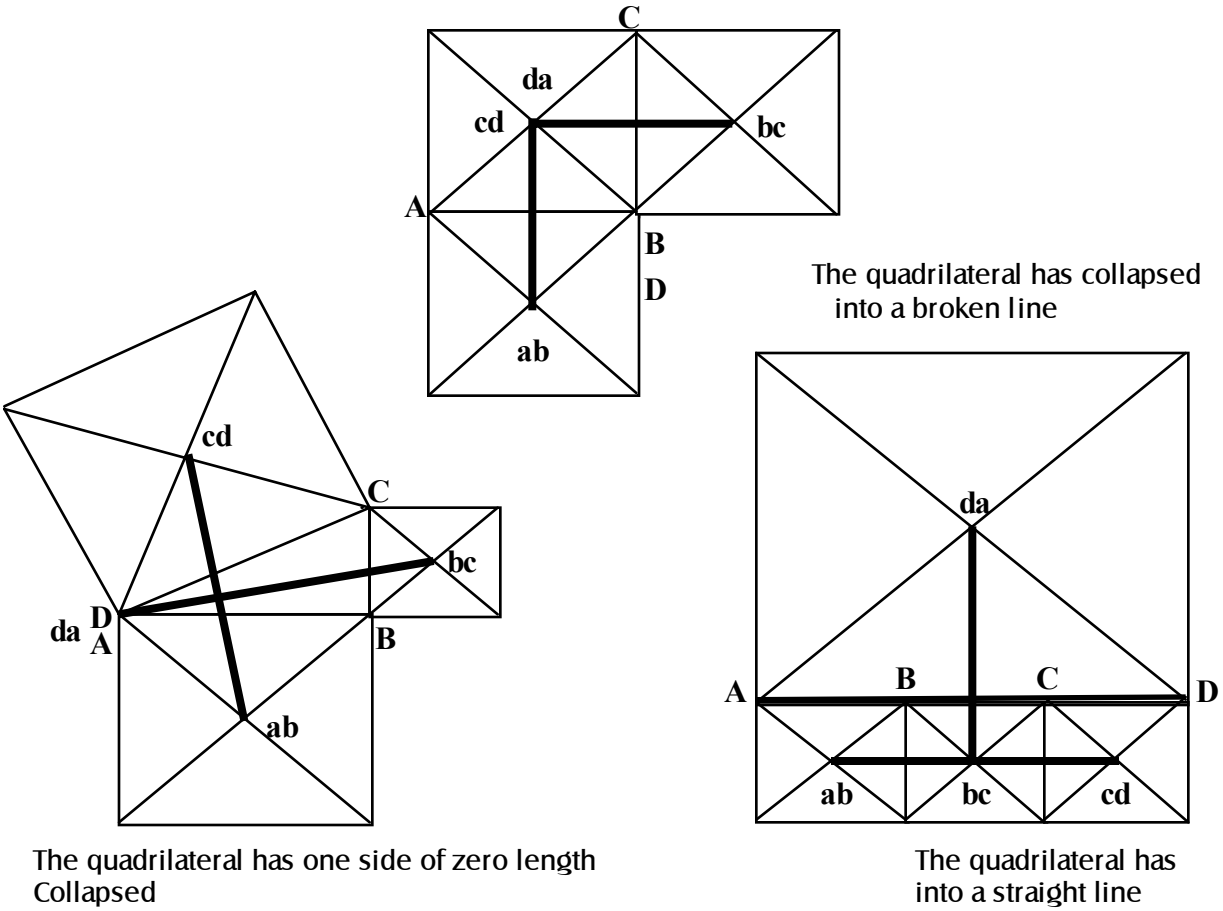

Figure 4. The vertices are A, B, C, D. The centers of the squares are ab, bc, cd, da for the squares corresponding to sides AB, BC, CD, DA respectively. All squares are built on the right side when traversing the vertices of the quadrilateral in order ABCDA.

5. A MICROWORLD FOR MUSIC

In order to show a microworld that is not for geometry, we now present one for the teaching of the physical facts of music. The project will give us the opportunity of introducing some ideas related to lists and their handling. The objective of our computerized educational project is to build a little piano with a portion of the two lower rows of the letter (and some punctuation marks) portion of the keyboard of a PC. The lower row will correspond to the white notes of the piano and the second row to the black notes. We will be able to include one octave and a half of a piano.

LogoWriter has an instruction **Tone** that takes two inputs *frequency* and *duration*. Frequency is a number between 37 and 9999.9999 and duration is an integer between 0 and 255. The instruction sounds out a musical note. The frequency is the number of cycles per second (also called *hertz*) of the vibration producing the note and determines how high is the pitch of the sound; the higher the frequency the higher the pitch. The duration measures the length of the time period during which the sound will persist in units of 1/20 of a second. Using this instruction it is possible to play melodies; all that is necessary is to know the frequencies of the musical notes. Some books and computer manuals have tables of the frequencies of the notes. Here we will deduce the frequencies using frequencies and mathematics to illustrate that an educational project often requires the students to solve intermediate problems including doing research at a library.

Those who have attended a concert will have noticed that before they start one of the musicians plays a note for the others so they can all properly tune their instruments, especially the strings. When there is a piano, the note struck is an A whose frequency is 440 hertz. The A struck is the one immediately below "middle C" which is the C nearest the center of a standard piano. Since the times of J. S. Bach, who popularized the "well tempered scale," the notes of the chromatic scale (which is the sequence of white and black notes in order of their position) are separated by an equal interval. This interval is called a *semitone*. The semitone is characterized by the fact that the frequency of any note is the frequency of the previous note multiplied by a factor K which remains constant for all the chromatic scale. We are going to deduce the value of K from a few physical and mathematical facts.

The frequency of any note is equal to double the frequency of the note with the same name one octave below. If we count both white and black notes an octave includes 12 notes, or what is the same 12 semitones. Thus the notes repeat their names in a piano keyboard in a very similar way the hours of the clock repeat. We know the frequency of the A given for tuning which is 440. The next note, an A# (pronounced *A sharp* (another name for the same note is *B flat* or B*b*), will have a frequency of $440\times K$. The next note B will have a frequency of $440\times K\times K = 440\times K^2$. The next note C will have a frequency $440\times K^3$. If we continue in the same way until we get to the A of the next octave we have that the frequency of that A is $440\times K^{12}$. But since we know that the frequency of a note with the same name an octave higher is double the original frequency, another expression for the frequency of the higher A is 2×440. Equating the two expressions for the same frequency we obtain $440\times K^{12} = 2\times 440$. Canceling the 440 on both sides of the equation we get $K^{12} = 2$. In order to determine the value of K we must solve this equation. Those having access to a language like BASIC, FORTRAN, or C, or simply a scientific calculator, could immediately give the answer by extracting the twelfth root of two. Regretfully LogoWriter does not have a 12^{th} root extractor, which may be a lucky situation, because it gives us the opportunity to illustrate how a sub problem must be solved to reach the goal of a project, giving the teacher the opportunity to teach some useful mathematics in a situation in which the students are motivated, because they want to play some melodies on the little piano.

To solve the equation $K^{12} = 2$ we will use a method worth knowing because of its wide applicability: the *Method of Succesive Approximations.* We recall that to find the twelfth root of a number is to find a second number such that when it is raised to the twelfth power (that is, multiplied by itself 12 times) gives us the original number. In our case, what we want is to find a number that when multiplied by itself 12 times gives us 2. We could try several numbers until we find one that satisfies the criterion. In order to

save time we notice that 1 multiplied by itself 12 times gives 1, while 2 multiplied by itself 12 times gives a large number (actually 4096). Therefore, the number we are looking for must be between 1 and 2, probably much closer to 1. Let us try 1.1. In Logo we could write

```
(print 1.1 * 1.1 * 1.1 * 1.1 * 1.1 * 1.1 * 1.1 * 1.1 * 1.1 * 1.1 * 1.1 *
                                1.1)
```

and realize that is it too large (3.1386). However, when we try another number we will have to do a lot of typing, so it is much better to write either a procedure or a line like

```
(print :n * :n * :n * :n * :n * :n * :n * :n * :n * :n * :n * :n)
```

which can be executed repeatedly by simply pressing the <RETURN> key while the cursor is somewhere on the line. We could give a value to the variable n by typing a line such as

```
make "n 1.1
```

and pressing the <RETURN> key. Once we see the result we can decide if the value tried is too large or too small and change it accordingly. Since 1.1 turned out to be too large we next try 1.05 and obtain for its 12^{th} power 1.796. This is too small, hence we try something somewhat larger, say 1.06 to obtain for the 12^{th} power 2.0123 which is fairly close. We may try to get even closer so we diminish the last tried number a little and try 1.059. We obtain for the 12^{th} power 1.9895. Too small, so we augment it a bit to 1.0595 for which its 12^{th} power is 2.001 (very close). We may still try to improve the last one and try 1.0594 to obtain for the 12^{th} power 1.9986 which is not as good as the previous one, so we settle for 1.0595.

We have found that students like this process and when they run out of decimals in Logo (LogoWriter works with a precision of 4 decimals) they often take out their scientific calculators and continue the process to get 1.059463 or something even better. As a teacher you better be prepared to know that the value of K to 15 decimals is 1.059463094359295 or risk loosing face to some smart student. The situation is a welcome change to most mathematics classes in which the students want to get out as soon as the bell rings. We have often encountered groups of students who give up their recess to continue working on a project such as this one.
Returning to the music, once we know the value of K we can calculate the frequencies for any note on the piano we like. We could do it one by one, or even better, use Logo to calculate all the ones we are interested in in a single stroke. Without going into too many details, here is a procedure that outputs a list of frequencies of the succesive white and black notes starting with 440 hertz A.

```
to calcfreq :list :n :nn
if :n > :nn [op bf bf bf :list]
op calcfreq (lput round (last :list) * 1.0595 :list) :n + 1 :nn
end
```

In the procedure **calcfreq** three inputs are required: the list **list** where the frequencies will be stored, the number **nn** of frequencies that will be calculated, and a numerical counter **n** to control the number of frequencies that have been calculated up to a point in the process. In the second line of the function procedure (a procedure is a function procedure if it outputs with the instruction **output** abbreviated **op** a Logo object – a number, word or list -- at the end of its execution.) there is a conditional instruction which tests if we have calculated the number of frequencies we wanted in order to stop the process (a process stops as soon as an **output** instruction is executed.) and output the result. We have decided to make the leftmost key in the lower row of the keyboard a C, but our base frequency for our calculation is for an A (440 hertz), therefore we will calculate all the frequencies starting with that of the A note, but will output the list without the first three numbers (corresponding to notes A, A# and B). For this reason we output the *butfirst* of the *butfirst* of the *butfirst* of the list where we stored all the frequencies. The last line before

the end line is a *recursive* call to the function being defined, something often done in Logo, (see [12]) which assumes we somehow have a function that can calculate the next frequency based on the previous ones, so we output the result of applying the function to the n + 1 stage in the process, by putting at the end (instruction **lput**) an element calculated by multiplying by 1.0595 by the last element (instruction **last**) of the previous list **list** rounding it to the nearest integer (instruction **round**) keeping the same limit **nn** on the number of frequencies for the whole process. Since in each stage the counter increases by one, eventually it will become greater than nn, and the second line of the procedure will stop it and report the list where the frequencies have been accumulating.

It is unlikely the students will on their own write a procedure such as the one exhibited which uses recursion. They may calculate the frequencies one by one multiplying the previous one by the factor 1.0595. That is perfectly all right, it means they have understood the idea. The procedure we exhibit is for honor students who want to learn the more advanced features of the Logo language. The teacher will decide whom to encourage to write such a procedure.

In order to get a printed list for 20 – 2 = 18 frequencies (which is the number of keys in the piano) starting with the one of middle C (we order a calculation of 20 frequencies in addition to the initial frequency 440 but we discard the first 3 frequencies) we type in the command area of LogoWriter

```
(print calcfreq (list 440) 1 20)
```

and we obtain the following in the work area of LogoWriter: (only the numbers appear, we have added the names of the notes and of the keys below, running ahead of ourselves, for easy reference.)

frequencies `523 554 587 622 659 698 740 784 831 880 932 987 1046 1108 1174 1244 1318 1396`

notes	**C**	**C#**	**D**	**D#**	**E**	**F**	**F#**	**G**	**G#**	**A**	**A#**	**B**	**C**	**C#**	**D**	**D#**	**E**	**F**
keys	**<**	**a**	**z**	**s**	**x**	**c**	**f**	**v**	**g**	**b**	**h**	**n**	**m**	**k**	**,**	**l**	**.**	**–**

Once the frequencies have been calculated and the notes and keys they correspond to have been identified, the project's next step is to write a program for the operation of the little piano. If we examine the list of frequencies, notes and keys above, we will notice that the keys are arranged somewhat like the notes of a piano, that is, the black notes are interleaved. For example the F which corresponds to the note F# is above and between the keys C and V which correspond to the notes F and G. There are some keys on the second row that do not correspond to notes, for example, the key D which is in a place where a piano does not have a black note. These keys will produce no sound in our little piano, as will all other keys, for example those in third and fourth rows.

The program which will play the notes is the following:

```
to piano
make "note readchar
if :note = "< [tone 523 5]
if :note = "a [tone 554 5]
if :note = "z [tone 587 5]
if :note = "s [tone 622 5]
if :note = "x [tone 659 5]
if :note = "c [tone 698 5]
if :note = "f [tone 740 5]
if :note = "v [tone 784 5]
if :note = "g [tone 831 5]
if :note = "b [tone 880 5]
if :note = "h [tone 932 5]
if :note = "n [tone 987 5]
if :note = "m [tone 1046 5]
```

```
if :note = "k [tone 1108 5]
if :note = ", [tone 1174 5]
if :note = "l [tone 1244 5]
if :note = ". [tone 1318 5]
if :note = "- [tone 1396 5]
if :note = "q [stopall]
piano
end
```

The procedure **piano** is very simple. The idea is to detect which key is pressed by the user. This can be done with the instruction **readchar**, which is a function with no inputs which outputs the character associated with the key pressed. We start by storing in the variable **note** the character being pressed. The rest of the procedure is a sequence of **if** instructions that compare what was stored in **note** with each of the characters corresponding to the keys that stand for a note. If one of them coincides, then the instruction **tone** sounds out the corresponding note with the proper frequency and a duration of ¼ second. If none of the **if** instructions is *true* and if the "q" key has not been pressed, the procedure calls **piano** again (that is, it calls itself) which starts the process all over again and the character associated with the next key pressed is stored in **note** and checked against all the keys representing notes over and over, until the user presses the "q" (for *quit*) key, which indicates to the program that the user wishes to stop with the instruction **stopall.**

With the piano we have built we can play melodies for which we know the notes. For example the "Mexican Hat Dance" requires us to press in sequence the following keys:
<, c, <, c, <, c, <, c, v, c, x, c, v, <, x, <, x, <, x, <, x, c, x, z, x, c.

The manner in which we completed the project is practical although not very elegant with respect to the programming, since there is a line of code for each note in the piano. If the piano had been larger the code would be longer. For more advanced students we now show more compact procedures to be used with **calcfreq** shown above..

```
to piano2 :freqlist :keylist
make "note readchar
tone bringfreq :freqlist :keylist :note 5
piano2 :freqlist :keylist
end

to bringfreq :freqlist :keylist :note
if empty? :keylist [op 9999]
if :note = first :keylist [op first :freqlist]
op bringfreq bf :freqlist bf :keylist :note
end

to storekeys
op (list "< "a "z "s "x "c "f "v "g "b "h "n "m "k ", "l ". "-)
end
```

The way the procedures **piano2, bringfreq** and **storekeys** work is as follows:

The procedure **piano2** accepts two inputs both of which are lists: *freqlist* and *keylist*. The first one stores the frequencies of the notes in the piano and the second one the keys that activate said notes. In the second line of procedure **piano2** the pressing of a key is detected and the corresponding character is stored in the variable *note.* In the next line the primitive **note** is invoqued using for its first input (the one referring to frequency) the output produced by a call to the procedure **bringfreq** with its three inputs *freqlist, keylist* and *note,* and for its second input (the one referring to duration) the constant 5. The next line is a call of **piano2** to itself, hence the next keypress is processed. The procedure **bringfreq** has

three inputs *freqlist, keylist* and *note* and according to its use in **piano2** should output the frequency associated with the character stored in *note.* What the procedure **brinfreq** does is to look for the character stored in the variable *note* and output the corresponding frequency in *freqlist.* The lists *freqlist* and *keylist* are paired so each element in one corresponds to the element in the same position in the other. The way to look for the character in *note* in *freqlist* is to start by looking at the first element of *freqlist*, if it does not match then the search is transferred to the list *bf :freqlist* which is the previous list from which the first element has been removed. This is accomplished in a typical Logo style by a recursive call of the procedure to itself except that the first and second inputs are *bf freqlist* and *bf keylist.* This is done in the fourth line of **bringfreq.** The second and third line are conditional instructions to take care of the possibility that the key pressed does not correspond to any of the keys stored in *keylist,* (this is detected when *keylist* becomes empty) in which case the procedure returns a large number (9999) that would produce a sound that can not be heard and is therefore the equivalent of silence; and to the case in which the key has been found and the procedure outputs the corresponding frequency (which, because of the way it is looking for it must correspond to the first element of *freqlist* in the most recent call to **bringfreq.)** The procedure **storekeys** simply outputs the list of characters associated with the keys of the keyboard used as notes in the same order as **calcfreq** outpts the frequencies.

To operate the piano with the new procedures the user must type

```
Piano2 calcfreq (list 440) 1 20 storekeys   (1)
```

In the new procedures we did not provide for the possibility of stopping the piano by pressing the "q" key. For this version, to stop the program, the user should use the <Control-Interrupt> keys. Except for this detail both versions work in the same way.

The interaction betwee the procedures in the second version of the piano software give us the opportunity to say something about the syntax of Logo. We said previously that any Logo program must start with a command. The procedure **piano2** is a command. It has two inputs, both of which are lists. In line (1), the first input is provided by the output of a function call to **calcfreq.** If we turn a few pages back to the listing of **calcfreq** we can see that it outputs a list and that it takes 3 inputs: a list and two numbers. The list in line (1) is (list 440), and the two numbers are 1 and 20. The second input to **piano2** is provided by the output of a call to the function **storekeys.** Examining the listing of **storekeys**, we see that it has no inputs and that it outputs a list. This completes the analysis of the syntax of line (1). In order to simplify the analysis of the syntax of Logo one can use a pictorial representation of command and function calls. Fig. 5 shows a diagram associated with the calls of line (1). Each box representing a command or function has the inputs flowing from above and the output flowing out below. Functions may or may not have inputs but must always have outputs which must be inputs of commands or other functions. Commands may or may not have inputs but they may not have outputs. The boxes in a correct diagram must fit together for the syntax to be correct. Such things as constants and primitives can be thought of as boxes (with no name in the case of constants) that output something which will be the input of a command or function. The bottom boxes of a diagram like Fig. 5 must be commands, since the flow must stop somewhere.

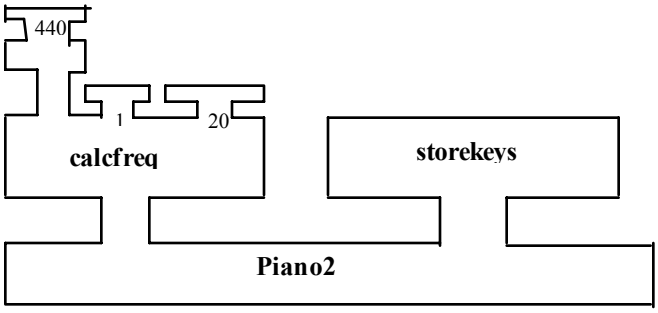

Figure 5. A schematic diagram of the manner in which the pieces of the program Piano2 fit together and communicate with each other.

6. A MICROWORLD FOR LEARNING SPANISH GRAMMAR

We now present two procedure which are a part of an educational project to automate the conjugation of Spanish verbs. Being a large project we only show a small piece to give the reader an idea of the possibilities. The first procedure conjugates in the present tense of the indicative mode the regular verbs of the first conjugation whose paradigm is the verb *amar.*

```
to conjugatepir1 :p
if not equal? word (last bl :p) last :p "ar [(pr [the verb ] :p
[does not end in
     ar.]) stop]
make "t1 "o make "t2 "as make "t3 "a make "t4 "amos make "t5 "áis
make "t6    "an
make "y mul mul :p
(pr "yo (word :y :t1))
(pr "tu (word :y :t2))
(pr "él (word :y :t3))
(pr "nosotros (word :y :t4))
(pr "vosotros (word :y :t5))
(pr "ellos (word :y :t6))
end
```

The use of the procedure **conjugatepir1** (whose name stands for **conjugate** in **p**resent **i**ndicative **r**egular conjugation **1**) is as follows: The infinitive of a regular verb of the first conjugation is selected, the verb must end in "ar." Examples are *amar, pernoctar.* The following is typed in the command center of LogoWriter.

```
conjugatepir1 "pernoctar
```

The program then writes the conjugation of the verb in present indicative for the six persons as follows:

```
yo pernocto
tu pernoctas
él pernocta
nosotros pernoctamos
vosotros pernoctáis
ellos pernoctan
```

The procedure starts by checking that the verb used as input actually ends in "ar". In case this is not so, it writes a message informing the fact to the user and stops. Next it uses the variables t1, t2, t3, t4, t5, t6 to store the endings for the conjugation in the tense and mode in question. In line 5 the root of the verb used as input is stored in the variable *y.* This is done by taking the input (infinitive) and deleting the last two letters. Finally through **print** statements the pronouns, root and endings are concatenated and displayed in the work area of LogoWriter. The procedure has no more intelligence than the one described, therefore it will take any word ending in "ar" and print a conjugation of said word as explained. Therefore in order that the program does not print garbage it is convenient to first check that the verb is in a list of regular verbs of Spanish. This will be illustrated with the conjugation of irregular verbs of a certain family which we take up next.

Many irregular verbs in Spanish are very similar in their conjugation; for that reason they are grouped in families that follow the same rules. To illustrate the techniques we will work with a group of irregular verbs which we will call group 6. We use the following procedures:

```
to verbs
make "verbsi6 [abuñolar aclocar acordar desacordar acornar descornar
mancornar aforar desaforar agorar alongar almorzar amolar apercollar
apostar avergonzar azolar colgar descolgar contar descontar recontar
degollar denostar descollar desflocar desmajolar desollar desosar
desvergonzar dolar emporcar enclocar encontrar encorar encordar
desencordar encovar engorar engrosar desengrosar entortar follar
afollar forzar esforzar reforzar holgar hollar rehollar moblar
amoblar desamoblar mostrar demostrar poblar despoblar repoblar
probar aprobar comprobar desaprobar improbar reprobar recordar
recostar regoldar renovar resollar rodar enrodar sonrodar rogar
solar asolar sobresolar soldar soltar sonar asonar consonar disonar
malsonar resonar soñar trasoñar tostar retostar trocar destrocar
trastocar tronar atronar retronar volar trasvolar volcar revolcar]
End
```

This procedure only loads the different groups of verbs as lists with names for each list. We only exhibit one group ***verbsi6*** (That stands for ***verbs i***rregular group ***6***) Each group is stored as a list. For the complete project all the verbs of Spanish must be classified and loaded as lists. To do the conjugation of a particular tense and mode a procedure similar to the following, given for present of indicative is written:

```
to conjugatepii6 :p
if not member? :p :verbsi6 [(pr [the verb] :p [is not conjugated
with model 6.])
       stop]
make "t1 "o make "t2 "as make "t3 "a make "t4 "amos make "t5 "áis
make "t6 "an make "y mul mul :p
ct pr :y top replace "o "ue top
select eol make "y1 selected ct
(pr "yo (word :y1 :t1))
(pr "tu (word :y1 :t2))
(pr "él (word :y1 :t3))
(pr "nosotros (word :y :t4))
(pr "vosotros (word :y :t5))
(pr "ellos (word :y1 :t6))
end
```

The workings of the procedure **conjugatepii6** is very similar to the one given previously for regular verbs except for some minor differences. Here we show how it is checked that the verb belong to the family (verbsi6) for which the procedure is designed. The handling of the endings is handled in the same way. Because this family of verbs changes its root for some of the persons in present indicative, and additional variable y1 is introduced. To handle the change in the root, the procedure looks for the appearance of the first "o" in the infinitive (as a matter of fact all the infinitives of the verbs of group 6 have only one "o") and changes it to "ue". All this is done with LogoWriter text processing primitives. Among these primitives are: **top** which takes the cursor to the beginning of the work area. The primitive **replace,** takes two inputs; searches the text in the work area (starting from the cursor's position) and locates the first appearance of the first input and replaces it with the second input. The primitive **select** starts marking the text at the position of the cursor and marks all the positions the cursor takes under instructions that move the cursor until an **unslect** instruction or any other editing instruction that modifies the text such as **replace, print, ct** is executed. The primitive **selected** outputs whatever portion of the text is marked**.** The primitive **eol** takes the cursor to the end of the line. (If marking is active it marks all the text in between.)

As an example of the operation of **conjugatepii6** we show the result of typing in the command area of LogoWriter the instructions

```
Verbos conjugatepii6 "trocar
yo trueco
tu truecas
él trueca
nosotros trocamos
vosotros trocáis
ellos truecan
```

In the two examples treated only the present tense of the indicative mode was included. For each tense of every mode procedures would have to be written which would include the appropriate terminations as well as the list of verbs belonging to the group. Consideration should be given to the fact that some verbs are defective and can not be conjugated for certain persons and certain tenses. To do it for the full range of the Spanish language is somewhat laborious but is well within the scope of a typical college student as long as she has information like the one in [10]. The person who completes this project in its entirety will certainly be very knowledgeable in the conjugation of Spanish verbs. (For more information on the topics of this section see [9], [10])

7. CONCLUDING REMARKS

Through the exhibition of three computerized educational projects in different microworlds we hope to have shown the possibilities of Logo in providing a learning environment. Program listings in LogoWriter have been provided. Although there are more modern versions of Logo available in the market, it is still felt that the purpose of the paper is well served by LogoWriter among other things because there are still many schools using Apple and Pre Windows PC computers, particularly in developing countries. The philosophy of learning by teaching a dumb machine to solve a problem, that is by programming the solution, forces the programmer to master the topic, because the programs will not function correctly unless thay are very well thought out and are given a touch of generality. In contrast with other educational philosophies such as programmed instruction and drill and practice programs in Logo it is the student that programs the machine and not the opposite. Recent versions of Logo have even more facilities than the ones shown here. For example *Object Logo* [8] is an Object Oriented Language that can do fraction arithmetic, essentially unlimited floating point precision arithmetic, complex number arithmetic, handle an essentially unlimited number of turtles and other objects such as printing and drawing windows, handling of buttons and sliders to control variables, etc. *Microworlds* [11] is an object oriented package that runs under *Windows,* manages menus, dialog windows and icons with the mouse. It has many instructions including some for color animation, and facilities to work multimedia, including voice recording, showing video, editing of graphics with a very complete set of colors to design forms and background images. Logo Gráfico [13] developed in Pascal in Argentina is also object oriented, run under Windows has a Pascal like syntax with constructions such as **while** and **until**, very good color graphic animation facilities with extra large turtles and turtle forms called **actors.** It also has special instructions for teaching physics such as object that can be accelerated and follow the laws of mechanics in movements such as simple harmonic motion.

Another learning environment that has been introduced by the promoters of Logo has been educational robotics Lego-Logo wich includes extensions of the Logo language to manipulate lights, motors, switches, buzzers, which together with bands, pulleys, gears, drums and other mechanical components allow the users to build all sorts of devices such as simple toy-factories with automated transportation of raw materials and finished goods, which students like because they are a lot of fun and can aid in the teaching of technology and physical science. There are several of these systems in the marketplace.

There is a considerable body of literature on Logo ranging from books for different educational levels [14],[16], [25], magazines and journals of professional societies [21], as well as research articles, information spreading articles, with ideas to try in the classroom and evaluations of the effect of Logo in learning. There is a special interest group on Logo in the International Society for Technology in Education (ISTE, [20]) and conferences and symposia are held periodically on Logo in different parts of

the world. Good sources of information on Logo are the Logo Foundation [17] located in New York, as well as the companies Logo Computer Systems, Inc., (LCSI, [23]), located in Montreal, and Terrapin Software, Inc [18], located in Portland, ME. There is also Fundaustral [26], located in Buenos Aires, Argentina, the Omar Dengo Foundation located in San José, Costa Rica, which has many years experience in the use of Logo as a learning environment. Finally, the Media Lab at MIT [22] is the center where many of the most active researchers on Logo, including S. Papert work.

8. REFERENCES

[1] Papert S., 1981 Mindstorms: Children, Computers, and Powerful Ideas, Basic Books, New York, 1980. There is a Spanish translation: *Desafío a la Mente: Computadoras y Educación*, Ediciones Galápago, Buenos Aires.

[2] Papert S. 1993, The Children's Machine: Rethinking School in the Age of the Computer, Basic Books, New York.

[3] Logo Computer Systems, Inc, LogoWriter: Guía de Referencia, Macrobit Editores, México, 1990.

[4] Murray-Lasso M. A., Introducción a LogoWriter: Un Lenguaje Educativo, Memoria del X Simposio Internacional de Computación en la Educación, Sociedad Mexicana de Computación en la Educación, México, October 1994, pp. 41 – 69.

[5] Murray-Lasso M. A., Introducing Computers to Teenagers through Music The Computing Teacher: Journal of the Society for Technology in Education, February 1993, pp. 8 – 12.

[6] Hoyles C. and Noss R., 1992 Learning Mathematics and Logo, MIT Press, Cambridge, MA.

[7] Abelson H. and diSessa A., 1981 Turtle Geometry: The Computer as a Medium for Exploring Mathematics, MIT Press, Cambridge, MA.

[8] Paradigm Software, Inc., Object Logo, Paradigm Software, Inc., Cambridge, MA, 1990.

[9] Murray-Lasso M. A., La Enseñanza de la Gramática Española con Ayuda de la Computadora, Memorias del Segundo Seminario Internacional La Implantación de la Computadora en la Educación Latinoamericana, Secretaría de Educación Pública de México, México, April 1991, pp. 181 – 195.

[10] Alsina R., 1990Todos los Verbos Castellanos Conjugados, (4ª. Edición,) Ediciones Teide, Barcelona,.

[11] Logo Computer Systems, Inc., Las Técnicas Micromundos, LCSI, Montreal, 1994.

[12] Murray-Lasso M. A., Introducing Recursion Gently through Logo, Memoria del XI Simposium Internacional de Computación en la Educación, SOMECE, Puebla, October 1995, pp. 358 – 370.

[13] Grant C. R. y de Antueno E. A., 1996 Logo Gráfico Versión 4. Manual del Usuario. Descripción de Primitivas. Apéndices, Fundaustral, Buenos Aires.

[14] Yoder S. y Moursund D., 1996 Introduction to Microworlds: A Logo-Based Hypermedia Environment, (Second Edition), International Society for Technology in Education, Eugene, OR.

[15] S. Papert, 1980 Commputer-Based Microworlds in The Computer in the School: Tutor, Tool, Tutee, R. P. Taylor (Ed.,) Teachers College Press, New York.

[16] Harvey B., 1987 Computer Science Logo Style, 3 Volumes, MIT Press, Cambridge, MA, 1985, 1986.

[17] Logo Foundation, 250 West 57th. Street Suite 2603, New York, NY, 10107-2603, U.S.A.

[18] Terrapin Software, Inc., 400 Riverside Street, Portland, ME 04103, U.S.A.

[19] Murray-Lasso M. A., El Proyecto Educativo Computarizado: Un Ejemplo en Logo Memoria de SOMECE '93: Simposio Internacional de Computación e Educación, Sociedad Mexicana de Computación en la Educación, Querétaro, July 1993, pp. 101 – 103.

[20] International Society for Technology in Education (ISTE,) 1787 Agate Street, Eugene, OR, 97403-1923.

[21] Logo Exchange: Journal of the ISTE Special Interest Group for Logo-Using Educators.

[22] MIT Media Lab E15-309, 20 Ames Street, Cambridge, MA 02139, U.S.A.

[23] Logo Computer Systems, Inc. (LCSI,) 3300 Cote Vertu, Suite 201, Montreal, Quebec H4R 2B7, CANADA.

[24] Murray-Walpole G., Murray-Lasso M. A., y Murray-Walpole V., Cómo Ejecutar Música Clásica por Computadora sin Previo Entrenamiento, Memoria del Tercer Simposio Internacional La Computación y la Educación Infantil y Juvenil, Academia de la Investigación Científica, Puebla, October 1986, pp. 34.1 – 34.11.

[25] de Antueno E. A., Naveira A. M. y Thompson H. H., 1984 Computación en el Colegio – LOGO: Espejo de la Mente, Editorial Informática Educativa, Buenos Aires.

[26] Fundaustral, F. L. Beltrán 267 – 1406 Buenos Aires, ARGENTINA.

[27] Polya G., 1981 Mathematical Discovery, Vols. I and II, (Combined Edition), John Wiley & Sons, Inc., New York.

[28] Murray-Lasso M. A., Aplicación de los Números Complejos en la Enseñanza de la Geometría Plana Ingeniería: Investigación y Tecnología, Vol. 1, No. 5, October – December 2000, pp. 215 – 223.

[29] Kelly P., Von Aubel's Quadrilateral Theorem, Mathematics Magazine, January 1966, pp. 35 – 37

Author Biography

Marco A. Murray-Lasso

Born in Mexico City, MEXICO September 1, 1937
Mechanical-Electrical Engineer, 1960, National University of Mexico, Mexico City.
Master of Science in Electrical Engineering, 1962, Massachusetts Institute of Technology, Cambridge, MA.
Doctor of Science, Automatic Control, 1965, Massachusetts Institute of Technology, Cambridge, MA.
Professor of Systems Engineering, Graduate School of Engineering, National University of Mexico.
Member of the Academy of Engineering (Honor Member, Founding President of the National Academy of Engineering), Mexican Academy of Technology (Founding Member), Mexican Academy of Informatics, Mexican Academy of Sciences (former Secretary General), Mexican Academy of Sciences, Arts, Technology and Humanities (Founding Member). Former President of the Council of Academies of Engineering Technological Sciences (CAETS).