

ELICA PHILOSOPHY

Written by
E. Sendova and P. Boytchev
e-mail: jsendova@mit.edu, pavel@elica.net

Rev 1.1
July, 2003

Contents

1.	WHY LOGO.....	3
2.	ELICA PHILOSOPHY.....	4
2.1	THE LOGO CULTURE AND SPIRIT	6
2.2	EXPERIMENTATION AND HYPOTHESIS GENERATION	6
2.3	LANGUAGE REPRESENTATION.....	7
2.4	MICROWORLDS.....	9
2.5	TOOLS AND TECHNIQUES.....	11
2.6	PROGRAMMABILITY	11
2.7	ANTITHESIS.....	12
2.8	SPIRIT	12
3.	PRE-ELICA HISTORY.....	14

1. Why Logo

It is a big problem to decide what computer language to use as a basis for leaning and experimenting. The trade market evaluates languages according their speed of compilation (or interpretation), number of users, street price etc. But as it always happens these scores affect only the trade-oriented decision. Many languages are claimed to be good, but none of them is simple enough to be used as a start up language.

During the design phase Elica needed a base language which has simple syntax, intuitive semantics, acceptable speed and big popularity among computer users and soon-to-become-users. Right, you guessed it - LOGO suits perfectly. It satisfies all the requirements mentioned above.

The language Logo like any other language has no world-wide standard specification. In fact, Logo is not a programming language, but a set of Logo dialect languages, combined under the general name Logo. Programs written on one Logo implementation can differ more or less from the same program tuned for another one.

Elica is based on Elica Logo - this is a special dialect of Logo, entirely oriented to serve Elica needs. Although Elica Logo is a dialect it can be treated as intersection between all other Logo dialects, and thus it might be closed to the imaginary standard. It has the absolute minimum of reserved words and the amazing feature of expandability.

Elica is an Integrated Development Environment, which programming language is Logo. The dialect of Logo used is called Elica Logo. The basic elements of Elica Logo are quite similar to those of other Logo implementations. Elica Logo uses well-known reserved words for basic commands.

But Elica Logo has some advanced features, which include rules (for automatic response for change events), objects, virtual properties and event handlers. The advance features also include defining and using user libraries and the possibility for the user to declare and modify the interactive use of the system.

Elica Logo is not a full functional Logo programming language, because, for example, it does not know how to process basic Logo data types: numbers, words and lists. However, Elica Logo together with the Elica Logo Library form a full-functional Logo implementation.

2. Elica Philosophy

Elica is the most recent development of a series of educational language-based computer environments for creative learning in a constructionist's style. Work in all these environments helps learners to acquire skills needed to participate with understanding in the construction of what is new. Inspired by *the Logo culture and spirit*¹ they facilitate the *experimentation and hypothesis generation*², and *open-ended exploration*³. In addition, the *language representation*⁴ of the learner's activity provides an occasion for turning the thinking process itself into an object of analysis and explorations - something that is very powerful from an educational point of view.

The concepts of Elica provide a perfect balance between simple syntax, clear semantics, acceptable speed and amazing results:

- Elica is based on a limited set of *Logo commands*⁵ providing a complete and rapid take-off towards very complex programs. On one hand Logo takes advantages from both worlds – imperative and functional programming. On the other, the *unique style*⁶ of introducing objects to Logo environment discovers new horizons by enhancing their strength, usefulness and flexibility.

¹ For details see 2.1

² For details see 2.2

³ For efficient learning, an exploratory phase should precede the phase of verbalization and concept formation and, eventually, the material learned should be merged in, and contribute to, the integral mental attitude of the learner. (in *On Learning, Teaching and Learning and Teaching*)

⁴ For details see 2.3

⁵ Since the language (Logo) is a procedural programming language, it introduces the students to ideas about procedural abstraction that are important not just in the world of computer science, but in many other disciplines (B. Harvey, *Computer Science Logo Style*, Cambridge, MA, MIT Press, 1985).

⁶ In a microworld the central technical actors are computational objects. Each object is a conceptual building block instantiated on the screen, which students can construct and reconstruct. To be effective they must evoke something worthwhile in the learner, some rationale for wanting to explore with them, play with them, learn with them. They should evoke intuitions, current understanding and personal images - even preferences and pleasures... One essential property of these objects is that the learner is afforded with the opportunity to move smoothly between different meanings derived from actions and language, and simultaneously to build new meanings. In much the same way that object-oriented programming facilitates the development of software, object-oriented theories can facilitate the development of theories by a beginner. (in S. Papert, *the Children's machine: re-thinking school in the age of the Computer*, NY:Basic books, Inc.)

- Elica provides fundamentals for building complex *microworlds*⁷ representing various knowledge domains. The philosophy behind it is to make the style of work in a specific microworld in full harmony with the proper style of work in the domain it is designed for. This means that
- Elica provides the whole range of *tools and techniques*⁸ from easy means for making sketches up to means enabling construction of arbitrarily complex tools. Such continuity might be best achieved by integrating *direct manipulation interface*⁹ with full *programmability*¹⁰. Thus
- Elica breaks down the *antithesis*¹¹ between programmability and direct manipulations and exploits the strength of both.

When tuned to a specific domain:

- Elica can bring into the classroom the *spirit*¹² of the subject matter as a science with its principles of questioning, investigating, hypothesizing and discovering. In addition
- Its rich linguistic medium provides clarity, objectivity, understanding, and a ground on which learners can develop their own domain-oriented vocabularies.
- Using Elica could enlarge the spectrum of the traditional school curriculum and enable more students and teachers to experience the authentic scientific spirit of various knowledge domains in which there are open problems, and in which formulating problems plays an important role. Educators, on their part, can search for new educational domains.

Thanks to these features Elica has the potential of redrawing the boundaries of instruction, learning and creative achievements - it can provide with insight:

⁷ For details see 2.4

⁸ For details see 2.5

⁹ It is common today to use computer interfaces that replace typing instructions by combinations of pointing, clicking, dragging, pull-down menus and so on and so forth. I think this is overdone and often done badly. The design of dynamic geometry programs is usually flawed in this respect, probably of a deep-seated antagonism to anything that suggest programming as a mode of using software. They consequently lost the chance to present the thing-like aspect of geometric operations performed by pull down, click and drag. (S. Papert, *An exploration in the Space of mathematics education*, IJCMML, Vol. 1, No.1, 1996 p. 101)

¹⁰ For details see 2.6

¹¹ For details see 2.7

¹² For details see 2.8

- novices who would like to obtain fast results in a specific domain
- researchers working within that domain.
- teachers, who would like become part of the research teams, not just reality-check for researchers.

2.1 *The Logo Culture and Spirit*

What is the **Logo spirit**? And why is this spirit so rarely found in computer work without Logo? An answer given by many Logoists is in the form of definition: Logo is a programming language plus a philosophy of education and this latter is most often categorized as constructivism or discovery learning.

But while the Logo spirit is certainly consistent with constructivism, there is more to it than any traditional meaning of constructivism and indeed more to it than **education**... The right answer to **what is Logo** cannot be **An X plus a Y**. It is something more holistic and the only kind of entity that has the right kind of integrity is a **culture** and the only way to get to know a culture is by delving into its multiple corners.

The acceptance of negatives is very characteristic of the Logo spirit: what others might describe as **going wrong** Logoists treat as an opportunity to gain better understanding of what one is trying to do....Of course rejecting *right* vs. *wrong* does not mean that *anything goes*. Discipline means commitment to the principle that once you start a project you sweat and slave to get it to work and only give up as a very last resort. Life is not about knowing the right answer - or at least it should not be - it is about getting things to work¹³!

2.2 *Experimentation and Hypothesis Generation*

What you have been obliged to discover by yourself leaves a path in your mind which you can use again when the need arises. (*G.C. Lichtenberg: Aphorismen*)

"Teaching to think" means that the teacher should not merely impart information, but should try also to develop the ability of the students to use the information imparted; he should stress know how, useful attitudes, desirable habits of mind.

Mathematical thinking is not purely "formal", it is concerned also with generalizing from observed cases, inductive arguments, recognizing a mathematical concept in a concrete situation or extracting this notion from it. Let us teach proving, but let us teach also guessing.

¹³ S. Papert, *What is Logo? And Who Needs it?* in Logo Philosophy and Implementation, LCSl, 1999

For efficient learning, the learner should discover by himself as large a fraction of the material to be learned as feasible under the given circumstances. This is a very old principle. It underlies the idea of Socratic method.

Let the students actively contribute to the formulation of the problem. In the work of the scientist, formulating the problem may be the better part of a discovery, the solution often needs less insight and originality than the formulation. Thus, letting your students have a share in the formulation, you not only motivate them to work harder, but you teach them a desirable attitude of mind.

The best motivation is the student's interest in the task, yet there are other motivations which should not be neglected. A little practical trick is before students do a problem to let them guess the result, or part of the result. The boy who expresses an opinion commits himself - his prestige and self-esteem depend a little on the outcome, he is impatient to know if his guess will turn out right or not, and so he will be actively interested in his task and in the work of the class.

Do not give away your whole secret at once - let the students guess before you tell it - let them find out by themselves as much as is feasible. Voltaire "Le secret d'etre ennuyeux s'est de tout dire" - the Art of being a bore consists in telling everything.

Letting pupils do some preliminary exploration may whet their appetite for the formal solution.

Abstract understanding is like viewing terrain through a satellite map, while examples show what the land is really like under your feet. Research benefits from both approaches. Computation can help uncover surprising connections, and it can uncover fruitful areas for study. Computations and examples enrich and guide research as they do teaching. At a time when mathematicians are returning to computation, computers and symbolic computation programs are giving mathematicians an exciting opportunity to expand their research capabilities¹⁴.

2.3 *Language Representation*

The development of a language makes possible the guidance of a pupil learning from the particular to the general.

- Reading comes naturally; the literature is much deeper and wider. In order our conversations about mathematics to be at least as deep as our conversations about Beethoven and Shakespeare *we need a language to talk about and express our ideas about mathematics*¹⁵.

¹⁴ in *Computer and Conjecture*, Susan Landau, Notices, vol. 46, Number 2, February 1999)

¹⁵ from an Eurologo'93 discussion

- We have thought of learning to participate in mathematical activity as *learning to articulate structure and relationships*, and we have sought to describe conceptual worlds that possess this expressive power.
- The *language* is the crucial element - it provides a way to construct relationships which renders the construction visible... In short, the computational world can be autoexpressive - it can contain the elements of the language to talk about itself. At root, it is the language, the program, that allows the most obvious link between computational and mathematical discourses¹⁶.
- They say that using a programming language means to have the courage of admitting your own mistakes. On the other hand mistakes are welcome in a language based environment for two reasons at least:
 - they reflect very well the misconceptions of the learner since the way of verbalizing his/her understanding about a certain concept are recorded and therefore can be reflected upon.
 - they enable teachers to demonstrate different styles of debugging - a process much more educationally powerful than demonstrating just an error-free solution.
 - though the details, and the need for precision, may be specific to mathematical communication, similar expressive skills are important in communication in a broader sense – one must be able to express one's overall meaning in a non-technical way, and one must be able to add precision in a variety of ways¹⁷.
- Mistakes are the steps to success, provided you know how to handle them
- In computer jargon we use the term 'bug', and I strongly suggest this word rather than 'wrong' or 'mistake' or 'error". Bug has a more positive sound and encourages more positive attitude. In school if you get something wrong you get a bad grade and that's often the end of it. The kind of thinking called 'debugging' gets you on the track of bringing it closer to what you wanted to achieve. What is great here is that thinking again leaves you understanding what happened, and understanding what happened leads you to understand how to make something different happen. Your scheme didn't 'just not work'. It did exactly what you told it to do. *Recognizing the discrepancy between what you actually told it*

¹⁶ in Noss, R. and Hoyles, C. (1996). *Windows on Mathematical Meanings*, Kluwer Academic Publishers, p.227

¹⁷ E. Sendova, B. Sendov - Harnessing the power of programming to support explorations in Euclidean geometry, in *International Journal of Continuing Engineering Education and Life-long Learning*, vol. 9, Nos.2/3/4 1999 pp. 183-200

and what you meant to tell it is a big step towards getting it to work and an excellent exercise in formal language¹⁸.

Honing one's ideas in just about any domain requires that there be a way of putting the ideas 'out there' for one to scan and edit and rearrange and debug. There is much that one can learn about writing, painting, programming, composing music, or building furniture by reading and thinking about principles, theory, facts, and special techniques in each of the domains. But one gains skills in each area by doing the things, getting feedback (from reality testing or from expert critics), and debugging. One learns to write by writing, seeing if it 'works', and editing. A text-editor does not suggest projects, does not tutor, and may provide no feedback. If one brings no idea of the blank page nothing happens, but if one does fill the blank page with writing, the editor makes the editing stage easy. In a similar way a programming language has no agenda and provides no initial information. If one brings no idea to it, nothing happens, but if one does write a program, one can see if it works - testing the results against some reality of personal interest - and editing that idea accordingly.

Like the word-processor in its text domain, or logo in the general-programming-language domain, a dynamic geometry tool in its mathematical domain, is an idea editor. It is a blank page until the user fills it with an idea, and then it represents that idea in a form that the user can examine, manipulate, get feedback from, and 'edit'¹⁹.

2.4 Microworlds

The notion of microworld was first used by AI specialists to describe "a small, coherent domain of objects and activities implemented in the form of a computer program"²⁰. As Weir points out (ibid. p. 13) microworlds are environments which, "quite unlike the traditional classroom, are clearly in the discovery-learning tradition".

Noss and Hoyles emphasize that "the microworlds were born in the AI community as a way of capturing the notion of problem solving within an arena sufficiently constrained that the computers might be able to achieve a solution"²¹. They point out the contribution of Papert who made a small but

¹⁸ Seymour Papert, *The Connected Family*, Longstreet press, Atlanta, Georgia, 1996

¹⁹ E. Paul Goldenberg, *Bringing back Formal language: A Use to Counter my Worries about Computers in the Mathematics Classroom*, in R. Nikolov, E. Sendova, I. Nikolova and I. Derzhanski (eds.) EUROLOGO'99, Proceedings of the Seventh European Logo Conference, Sofia, Bulgaria 22-25 August 1999

²⁰ Weir, S. (1987). *Cultivating Minds: A Logo Casebook*. London: Harper&Row, p. 12

²¹ Noss, R. and Hoyles, C. (1996). *Windows on Mathematical Meanings*, Kluwer Academic Publishers, p. 63

significant change to the idea - the simple and constrained arena became part of *a knowledge domain* with epistemological significance.

DiSessa also stresses the epistemological basis of a microworld, pointing to the importance of making discovery part of mathematics education²².

Groen and Kieran consider microworlds as "concrete embodiments of a domain of mathematics", as "mini-domains of Piagetian mathematics" (i.e. a mathematics viewed as a key facet in general intellectual development)²³.

Feurzeig defines microworld as "a clearly delimited task domain of problem space whose elements are objects and operations on objects which create new objects and operations"²⁴.

In her detailed analyses of microworlds as representations Edwards²⁵ proposes a *structural* definition, which focuses on designed elements shared by the environments, and a *functional* definition which highlights the commonalities in how students learn with microworlds.

It is interesting that the notion of *microworld* evolves even in the works of one and the same team of authors. For example, Hoyles and Noss in their early works identify the set of Logo procedures focusing on a specific mathematical topic as "the microworld". Later on they propose a definition which includes several components, stating that "a microworld cannot be defined in isolation from either the learner, the teacher or the setting..."²⁶. Tracing the genesis of the idea of microworlds they conclude that "at the core of the microworld there is a model of a knowledge domain to be investigated by interaction with the software. Exploration is necessarily constrained but in ways designed to promote learning; knowledge is not simplified, it is recognized as complex, interrelated and evolving in action"²⁷.

²² diSessa, A. (1987). *Artificial Worlds and Real Experience*. In R.W. Lawler & M. Yazdani (Eds.), *Artificial Intelligence and Education. Volume One: Learning Environments and Tutoring Systems*, New Jersey: Ablex Publishing Corporation, p. 66.

²³ Groen, G., & Kieran, C. (1983). In *Search of Piagetian Mathematics*. In H. Ginsberg (Ed.), *The Development of Mathematical Thinking*, New York: Academic Press, p. 372.

²⁴ Feurzeig, W. (1987). *Algebra Slaves and Agents in a Logo-Based Mathematics Curriculum*. In R.W. Lawler & M. Yazdani (Eds.), *Artificial Intelligence and Education. Volume One: Learning Environments and Tutoring Systems*, New Jersey: Ablex Publishing Corporation, p. 51

²⁵ Edwards, L. (1995). *Microworlds as Representations*. In A. DiSessa, C. Hoyles, & R. Noss (Eds.), *Computers and Exploratory Learning*, Berlin: Springer-Verlag, p.127.

²⁶ Hoyles, C. And Noss, R. (1987) *Synthesizing mathematical conceptions and their formalization through the construction of a Logo-based school mathematics curriculum*, *International Journal of Mathematics, Science and Technology*, 18/4, p. 587

²⁷ Noss, R. and Hoyles, C. (1996). *Windows on Mathematical Meanings*, Kluwer Academic Publishers, p. 65

For us the fundamental aspect of a computational microworld is that it should be a medium where the learner's intuition is evoked and his/her current understanding of the phenomena (specific for the domain) is enhanced during the process of learning via programming. Furthermore, the microworld should provide a rich linguistic medium in which learners can develop their own domain-oriented vocabularies.

2.5 Tools and Techniques

Logo provides an interactive environment in which thinking may be expressed and reflected upon. Problems are explored and solve WHILE interacting with Logo. If problems are loosely defined you may not know ahead of time what tools you will need. Therefore it's best to have what you need to make tools as you go.²⁸

The one aspect of computers that distinguishes them from other devices that have served as either an educational medium or tool is their capacity to serve as tool-makers. The most fundamental examples of such are the general purpose programming languages (Logo, Pascal) and the more narrow purposed educationally oriented languages such as Function machines²⁹.

An old saying goes something like this: If a person has only a hammer, the whole world looks like a nail. Adding new tools to the carpenter's toolkit changes the way the carpenter looks at the world. In this context new computational paradigms can change the way computer users think about the world.

A new computational paradigm (parallelism in StarLogo) - (maybe dynamic reevaluation of the variables, dynamic change of a construction reflecting the connections and dependencies among the objects)³⁰

2.6 Programmability

If we are to shape software with the aim of revisioning mathematics, there must be an initial stage of resistance to tried and tested approaches, followed by experimenting and questioning. Without resistance and play, software use will almost inevitably become trivialized; used to reproduce rather than revision.

The novelty in exploration lay in resisting the temptation merely to shape the medium to the practice, and instead allow new mathematical possibilities to flow from their dialectical interaction. And this is precisely what carefully

²⁸ M. Tempel, Brian Silverman, *Creating a Logo Tool Box* -. LCSl

²⁹ Feurzeig & Richards, 1991, James Kaput, *Technology and mathematics education*

³⁰ Michel Resnick, *New paradigms for computing. New paradigms for thinking*, in A.A. diSessa, C. Hoyles, R. Noss (eds.) *Computers and exploratory learning*, Nato ASI Series, p. 31

designed computational environments should provide - a link between programming and the central ideas in Euclidean geometry.

...A central principle is that: a computational environment should be a place where the learner's intuition, her current explanations for phenomena, are evoked during the process of learning about some subject matter via programming activity³¹.

It is true that programming is hard but the efforts are well worthwhile if the computer is used as a medium for expressing mathematical ideas and not simply as a tool for learning well known facts.

2.7 *Antithesis*

What are advantages/disadvantages and the implications for learning in a language-based microworld as opposed to "point and click" method of interaction has been a matter of heated discussion³².

Historically direct manipulations interfaces and interactive programming environments have often been viewed as antipodal - the former being ideal for extra-linguistic tasks, and the latter - for tasks that call for formal representation and abstraction, such as the production of a novel intricate geometric design, or creative modeling of complex dynamical systems³³.

2.8 *Spirit*

And this new paradigm of the teachers would bring into the classroom the spirit of the subject matter (mathematics in our case) as a science - an open ended environment where problems with not known answers exist, other problems wait for the teachers and/or pupils to formulate them and together to act as a research team. Since the real flavour in mathematics does not lie in the body of facts or formulas that should be reproduced but in the eternal search for patterns, properties, harmony of shapes and in the style of thinking - to look for similarities, to understand what is unique and what could be generalized. And as it is often said it is not the final result that matters most but the variety of roads leading to it. And it is for the teacher to lead the pupils to the unknown. To

³¹ Noss, R. and Hoyles, C. (1996). *Windows on Mathematical Meanings*, Kluwer Academic Publishers, p 235

³² diSessa, A., Hoyles, C., Noss, R., Edwards, L. (1995). *Computers and Exploratory Learning: Setting the Scene*. In A. DiSessa, C. Hoyles, & R. Noss (Eds.), *Computers and Exploratory Learning*, Berlin: Springer-Verlag, p.8.

³³ in Eisenberg, M. (1995). *Microworlds as Representations*. In A. DiSessa, C. Hoyles, & R. Noss (Eds.), *Computers and Exploratory Learning*, Berlin: Springer-Verlag, p.180.

cultivate in them the spirit of adventure and have the confidence that although open the environment is not a jungle where he could be lost.³⁴

The real craft of the "maestro" is to enhance the scientist in the students, to demonstrate to them situations where one does not know the answer, even does not know if there is such an answer but is ready to take the risk of a possible failure in front of an audience. Before the introduction of the computer in class this was possible only with some selected students, now it becomes achievable with the whole class, in an ordinary school.

The teachers now are in the position to go beyond transmitting or receiving information, and can in addition generate ideas that can be either checked immediately or can, at least be used as a basis for formulating reasonable hypotheses. The implication is that students are given the chance to both observe and experience success and failure not as a reward and punishment but as information about what mathematics in fact is. Teachers and students in the role of researchers could treat the success as indication of being on the right track, failure - as being on the wrong one.

I have never seen anybody improve in the art and technique of inquiry by any means other than engaging in inquiry. We cannot teach the art of problem solving and problem formulating without engaging ourselves in inquiry - this is the message of the teacher³⁵.

It is *divine inspiration* (often a feedback to the audience) that can help in overcoming the boredom of lectures in which every little step is known in advance. But in order to be free to improvise it is crucial for the teacher to be supported by a computational environment that is built appropriately. *Appropriateness* however is rather subjective³⁶.

³⁴ Paul Goldenberg, *The art of problem posing*.

³⁵ Bruner, J. (1979) *On Knowing -- Essays for the left hand*, the Belknap Press of Harvard University Press, Cambridge, Massachusetts, London, England

³⁶ in E. Sendova, B. Sendov - *Harnessing the power of programming to support explorations in Euclidean geometry*, in International Journal of Continuing Engineering Education and Life-long Learning, vol. 9, Nos.2/3/4 1999 pp. 183-200

3. Pre-Elica History

The ultimate predecessor and inspirator for Elica are *Plane Geometry System* (PGS) and *Geomland*³⁷. They are Logo based geometry-oriented environments. The space and time between Elica and Geomland is occupied by several well-unknown systems. The first one was *TopLogo++*³⁸. It was a Logo-like language which ran out of the limitations of being a basis and now is a full featured compiler.

The next system was *TopLogo Geometry System* (TGS) written entirely on TopLogo++. The name was selected to resemble the former name of Geomland which was PGS at that time.

Later TGS was rewritten from TopLogo++ to Turbo Pascal and renamed to *Logo Geometry System* (LGS). When Borland released Turbo Pascal for Windows, LGS was upgraded to Windows too. The new system was called LGSW - *Logo Geometry System for Windows*.

Some time later a new idea appeared and that idea required a general revision of LGSW. The concepts were change to accommodate the some new requirements. The result is the *Relational Logo System*. Unfortunately this RLS had never had the chance to be programmed, because was too academic. After a reasonable revision the idea become simpler and gave birth to the next system. Its name again was RLS, but this time it meant *Research Logo System*. This last RLS is the direct parent of the current Elica system.

1985	PGS
	TopLogo++
1993	TGS
	Geomland
1994	LGS
1995	LGSW
1996	RLS
1999	Elica 3.0
2000	Elica 4.0
	Elica 5.0
2001	Elica 5.1
2002	Elica 5.2
2003	Elica 5.3
	Elica 5.4

³⁷ View it at <http://iea.fmi.uni-sofia.bg/PGS/INDEX.HTM> or download it from <http://iea.fmi.uni-sofia.bg/PGS/GEOMLAND.ZIP>

³⁸ Download it from <http://www.topteam.bg/exe/tl.zip>