LibreLogo - Handbook

Turtle Geometry

Andreas R. Formiconi

Versione 0.1

January 2018

Figure 1.1: Jan Fabre, "Searching for Utopia", 2003

Contents

1.1 Aknowldegments 4

I Basics of Logo and LibreLogo 5

1.2 Preface 7

1.3 Acknowledgments 10

II Basics of Logo and LibreLogo 11

1.4 Preface 13

2 LibreLogo 15

2.1 How to manage graphics in Writer 21

1.1 Aknowldegments

First af all I thank all the students who worked beyond my expectations, even by sending works that they were not supposed to do. Their explosion of creativity was extremely useful. Some of these contributes deserve specific mentions. First the remarkable "Creativity Exercises" of Marta Veloce, Primary Education student, that inspired chapter **??**. Then Alberto Averono, teacher of a secondary school, who in a training course about coding further developed Marta's exploration. Another Primary Education student, Eleonora Aiazzi, sent a moving text, "When a professor treats you as a child", who was extremely encouraging about the teaching method I was experimenting. A well experienced primary school teacher, Antonella Colombo, gave me back a wonderful documentation of Papert's syntonic learning she tried to trigger in her classroom. Her work opens the trip in chapter **??**, from the draw of a circle to the Halley's orbit. Thanks to Piero Salonia, who corrected the drafts of the first Italian version of this manual. Finally, thanks to the beautiful world of Linux and its sharp tools - scp, ssh, rsync, grep, find, nmap, latex, bibtex and so on - which gives true superpowers, unknown in the Graphical User Interface World - the Internet of true freedom.

Part I

Basics of Logo and LibreLogo

1.2 Preface

This is an English version derived from the “Piccolo Manuale di LibreLogo” that I wrote for the students of the Primary Education curriculum at the university of Florence. The manual was intended to aid the students in developing their coding activities within the ”Laboratorio di Tecnologie Didattiche” (Laboratory of Educational Technologies). The programming language is LibreLogo, which is an implementation of Seymour Papert’s Logo language within the Writer word processor of the LibreOffice suite. LibreLogo is a plugin available by default in Writer since versione 4.0 of LibreOffice. It has been written in Python by László Németh. The specific documentation can be found at http://librelogo.org. From there a description of all the LibreLogo commands can be downloaded in 33 different languages [1]. Apart from this, some extended documentations can be found only in Hungarian, as far as I know as of March 2018: there are a pretty technical handbook written by Lászlo Németh itself [5] and a more classroom oriented one by Lakó Viktória [7]. Some of the examples presented in this work have been inspired by those in Lakó Viktória’s book, but the perpespective is different here. First of all I have put a strong emphasis on pedagogical aspects, beyond the mere technical facts, following the line of thought of Seymour Papert, as reported, for instance, in *Mindstorms* [6]. Secondly, during the couple of years since the writing of the first version of the “Piccolo Manuale di LibreLogo”, a number of reflections, exercises and hands-on practices got back from students and other sources, have been added. The overall result is a rather broad discussion of possible uses of LibreLogo, both in a vertical dimension, from primary school examples to tertiary education level exercises, as well as in a transversal dimension, through a variety of disciplines. Finally, this is not a true translation of the “Piccolo Manuale di LibreLogo” but an English and somewhat more concise rewriting.

[1]The commands dictionaries can be downloaded from https://help.libreoffice.org/Writer/LibreLogo_Toolbar

Contents

1.3 Acknowledgments

First af all I thank all the students who worked beyond my expectations, even by sending works that they were not supposed to do. Their explosion of creativity was extremely useful. Some of these contributes deserve specific mentions. First the remarkable "Creativity Exercises" of Marta Veloce, Primary Education student, that inspired chapter **??**. Then Alberto Averono, teacher of a secondary school, who in a training course about coding further developped Marta's exploration. Another Primary Education student, Eleonora Aiazzi, send a moving text, "When a professor treats you as a child", who was extremely encouraging about the teaching method I was experimenting. A well experienced primary school teacher, Antonella Colombo, gave me back a wonderful documentation of Papert's syntonic learning she tried to trigger in her classroom. Her work opens the trip in chapter **??**, from the draw of a circle to the Halley's orbit. Thanks to Piero Salonia, who corrected the proofs of the first Italian version of this manual. Finally, thanks to the beautiful world of Linux and its sharp tools - scp, ssh, rsync, grep, find, nmap, latex, bibtex and so on - which gives true superpowers, unknown in the Graphical User Interface World - the Internet of true freedom.

Part II

Basics of Logo and LibreLogo

1.4 Preface

This is an English version derived from the "Piccolo Manuale di LibreLogo" that I wrote for the students of the Primary Education curriculum at the university of Florence. The manual was intended to aid the students in developing their coding activities within the "Laboratorio di Tecnologie Didattiche" (Laboratory of Educational Technologies). The programming language is LibreLogo, which is an implementation of Seymour Papert's Logo language within the Writer word processor of the LibreOffice suite. LibreLogo is a plugin available by default in Writer since version 4.0 of LibreOffice. It has been written in Python by László Németh. The specific documentation can be found at http://librelogo.org. From there a description of all the LibreLogo commands can be downloaded in 33 different languages[2]. Apart from this, some extended documentations can be found only in Hungarian, as far as I know as of March 2018: there are a pretty technical handbook written by László Németh itself [5] and a more classroom oriented one by Lakó Viktória [7]. Some of the examples presented in this work have been inspired by those in Lakó Viktória's book, but the perspective is different here. First of all I have put a strong emphasis on pedagogical aspects, beyond the mere technical facts, following the line of thought of Seymour Papert, as reported, for instance, in *Mindstorms* [6]. Secondly, during the couple of years since the writing of the first version of the "Piccolo Manuale di LibreLogo", a number of reflections, exercises and hands-on practices got back from students and other sources, have been added. The overall result is a rather broad discussion of possible uses of LibreLogo, both in a vertical dimension, from primary school examples to tertiary education level exercises, and in a transverse dimension, through a variety of disciplines. Finally, this is not a true translation of the "Piccolo Manuale di LibreLogo" but an English rewriting, characterized by a somewhat more concise exposition of the same facts.

[2]The commands dictionaries can be downloaded from $https : //help.libreoffice.org/Writer/LibreLogo_Toolbar$

Chapter 2

LibreLogo

LibreLogo is the implementation of the famous Logo language within the word-processor Writer. Writer is the word processing application of the LibreOffice, analogously to word, which is part of MS Office suite. Logo was created by Seymour Papert in the seventies to improve the learning of math. Seymour Papert, born in South Africa in 1928, first studied math in Johannesburg and successively in Cambridge. Between 1958 and 1964 he got his PhD at the University of Geneva with Jean Piaget: interesting collaboration between a mathematician and a pedagogist. Since 1964 he was a researcher of the MIT Artificial Intelligence laboratory where, in 1967 he got the position of codirector with Marvin Minsky, a relevant scientist in the field of artificial intelligence. The laboratory is the same where Richard Stallman, father of free software, would have worked a few years later. Free software is a beautiful reality but, ironically, it is incredibly widespread and, at the same time, so few people know something about. And it is right in the school context that it should be much more diffused, because of its relevant ethic and educational features. Probably, the best known free software is the Linux operating system, but there are many more: LibreOffice, analogous of MS Office, Gimp for manipulating digital bitmap images, Inskape for vector images, Audacity for audio recorded files, OBS for streaming and screencasting, shotcut for video cutting and many others. Seymour Papert is famous for having invented Logo, a programming language for drawing by giving commands to a "Turtle". However, in the first version, conceived in the Seventies, the Turtle was a robot which was able to draw by means of a pencil.

When computers arrived in the homes in the 80s, Logo became a software and as such was described by Seymour Papert in *Mindstorms*. In order to understand the pedagogical value of Papert's thought, let's read this passage, taken from *Mindstorms* (pp. 7-8) [6]:

> I take from Jean Piaget a model of children as builders of their own intellectual structures. Children seem to be innately gifted learners, acquiring long before they go to school a vast quantity of knowledge by a process I call "Piagetian learning", or "learning without being taught". For example, children learn to speak, learn the intuitive geometry needed to get around in space, and learn enough of logic and rethorics to get around parents - all this without being taught. We must ask why some learning takes place so early and sponta-

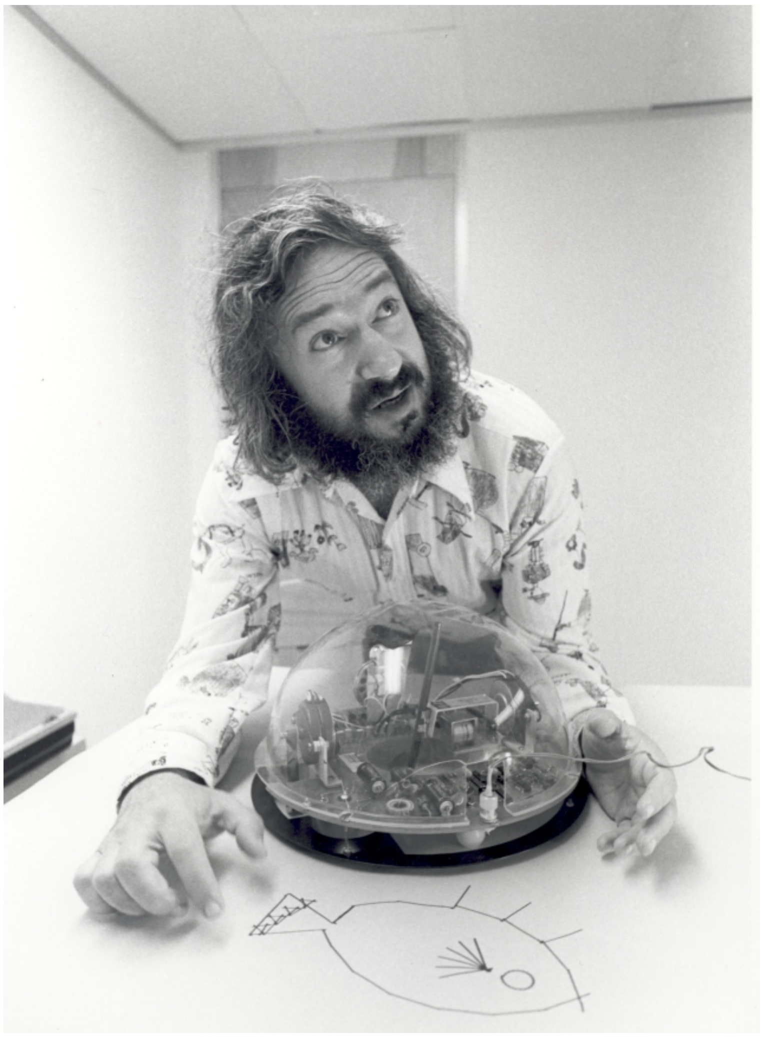
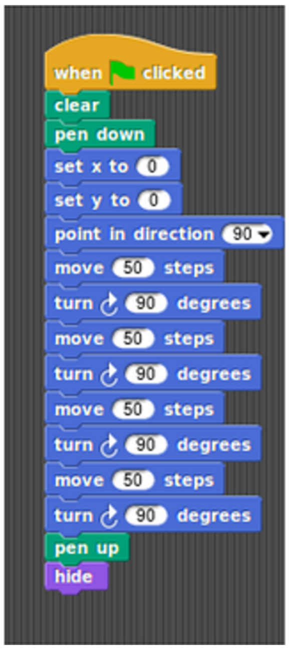


Figure 2.1: Seymour Papert shows one of the first versions of Logo, when it was kind of a robot for draving.

> neously while some is delayed many years or does not happen at all without deliberately posed formal instruction.
>
> If we really look at the "Child as a Builder" we are on our way to an answer. All builders need materials to build with. Where I am at variance with Piaget is in the role I attribute to the surrounding cultures as a source of these materials. In some cases the culture supplies them inabundance, thus facilitating constructive Piagetian learning. For example, the fact that so many important things (knives and forks, mothers and fathers, shoes and socks) come in pairs is a "material" for the construction of an intuitive sense of number. But in many cases where Piaget would explain the slower development of a particular concept by its greater complexity or formality, I see the critical factor as the relative poverty of the culture in those materials that would make the concept simple and concrete.

In the 90's Logo circulated as a program installable from a floppy disk. Once launched, it produced a black screen on which instructions could be written in sequence, one after the other. The instructions represented the movements given to the turtle on the screen. Then, with a special command, you could "execute" the sequence of commands, and so the turtle would move leaving a mark on the screen. Logo had a great resonance as an experimental method for teaching math and a wide variety of versions have been derived, reaching to generalizations such as the current Scratch. However, it has not been widely distributed in schools and maybe it has been more successful among kids than among teachers. Probably it was too early. Using Logo means writing code, an activity that is not part of the preparation of most teachers, including those who teach science. Today it is perhaps different, we talk a lot about *coding*, even if perhaps not always with full knowledge of the facts. The situation has evolved so much that *coding* can mean so many different things. Moreover, from the 1980s to the present, the variety of programming languages has grown enormously. Nowadays, among the logo derivations, Scratch is the most renowned educational programming language. Mitchel Resnick, a former student of Papert, is the project leader of Scratch and now is still following it at the MIT Media Laboratory. Scratch goes far beyond the production of graphics and allows you to create animations and video games, thus also allowing one to experiment with rather sophisticated programming techniques. Another innovative aspect is that it is structured as a web service and this has allowed the creation of a large community of living dissemination and exchange of programs. From the operational point of view, Scratch differs from Logo in that it is a visual language. The commands are in fact made up of coloured blocks that can be interlocked. The program comes out from the execution of these sequences of commands connected together, as in a puzzle. It's an attractive system that's a bit like Lego, where the instructions you give are stuck together like bricks. The joints ensure that instructions are combined only in legitimate ways, protecting against the typical and frequent spelling and syntactical errors that anyone who is writing software in the conventional text mode would encounter. Many of these languages have emerged, in addition to Scratch. The most famous are Snap!, Alice, Blockly, Android App Inventor, just to name a few. The following figure shows the difference between a text code and a visual code. The code is used to draw a square. Left the LibreLogo version and right the Snap! In



```
when [flag] clicked
clear
pen down
set x to 0
set y to 0
point in direction 90
move 50 steps
turn 90 degrees
move 50 steps
turn 90 degrees
move 50 steps
turn 90 degrees
move 50 steps
turn 90 degrees
pen up
hide
```

```
CLEARSCREEN
PENDOWN
HOME
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
PENUP
HIDETURTLE
```

Scratch this simple code would be identical. I used Snap! since I have a certain preference for this language. Snap! represents an enhancement of Scratch, which makes it more similar to a generic language, while maintaining the visual form. Among these features there is the possibility of saving the code in a standard format (XML) which is readable and editable by any text editor. For those who are used to working with softwares, this is a very important element. The code is not "optimal", in any sense. It is only intended to compare instructions in two different environments.

A special feature of Scratch is that it has created a large software sharing community. This happened thanks to the fact that it was conceived as a web service, which allows the writing of programs and the possibility of running them but also the realization of a social environment for sharing and remixing the programs. Visual languages have also drawbacks. They are (apparently) easy, fun and colorful, their effectiveness would seem guaranteed but the scientific evidence is not so clear. There are in fact various studies that show that visual languages do not facilitate so much the learning of "real" languages [2]. It seems that they are advantageous to understand the simplest constructs of programming but studies where the deep understanding about what a given algorithm does do not show substantial differences between visual and textual languages [4]. The research of Colleen Lewis is of particular interest. She compared the results obtained with Logo and Scratch in a class of children between 10 and 12 years [1]. The results showed that the learning of some specific coding constructs was facilitated by Scratch but, on the other side, the kids showed a higher level of self-esteem when introduced to programming with Logo.

And even if in the initial phases kids prefer visual tools, later on, once they try conventional textual programming, they may perceive the limits of visual coding:

- as limiting their creativity because less powerful
- because they feel visual coding less real: "if you have to do something real, nobody will ever ask you to encode it with visual educational software" [3]

It is on the basis of such considerations that we decided to focus on the Logo language, as an introductory tool to programming. Pretty a number of Logo versions are available nowadays. We here focus on a version that is available by default in Writer, the word processing application included in the LibreOffice office suite[1], analogous of the widespread MS Office suite. The latter is a "proprietary product", i.e. the company that produces it sells it but without distributing the source code in clear, according to the conventional industrial model, with which the intellectual property is jealously kept secret. LibreOffice is a free software, and as such is ideal for use in any educational context. Firstly, because it carries an ethical message. In fact, free software is defined by four types of freedom[2]:

- The freedom to run the program as you wish, for any purpose (freedom 0).
- The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbour (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

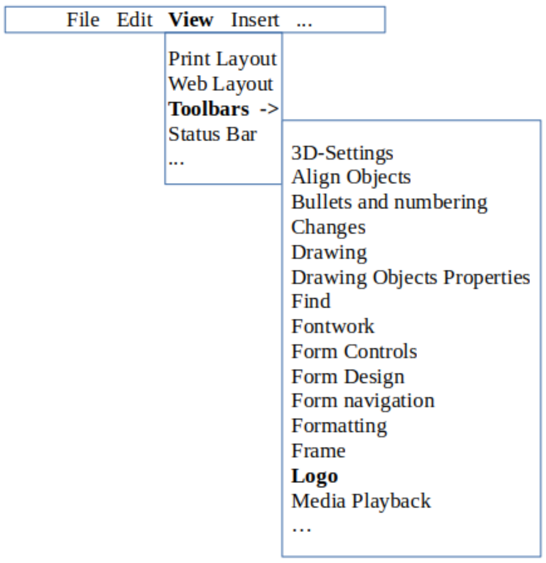
It should be noted - on this point many are confused - that *open source* software is different from (*free software*) since the ethical aspect is missing: open source software assumes that the source code is available in clear, but does not mention the four freedoms mentioned above and, in particular, the two specifications that characterize the ethical value of *free software*: "so as to help others" in the third freedom and "so that the whole community benefits" in the fourth freedom. Free software is developped by communities that may join together in non-profit societies. *Open source* is developped by private economic actors who adhere to the shared development paradigm because it fits well into their marketing strategies: there are companies that develop *open source* projects alongside traditional proprietary products because they find it convenient for their marketing strategies. LibreOffice's functionality can be enriched by means

[1]There is another similar project called OpenOffice. Many ask what are the differences with LibreOffice. A small history of the evolution of these two software, which have a common origin, can be found here (July 2016): http://www.navigaweb.net/2014/04/differenze-tra-openoffice-e-libreoffice.html. At present, LibreOffice is convenient because it incorporates more features and is updated more frequently.

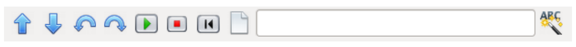
[2]Free software definition according to the free Software Foundation: https://www.gnu.org/philosophy/free-sw.en.html

of many *plugins*. LibreLogo is one of them and, from version 4.0 on, the LibreLogo plugin is included by *default* [3] in the program. But what does it mean to use Logo within a *word processor* like Writer, considered that this is a normal word processor while Logo is a kind of a drawing language? Simple: With the LibreLogo toolbar you can produce images that are integrated into the document as if they were imported. It's a brilliant idea, due to Németh László, who reproduced the features of Logo within LibreOffice. In reality, it has further increased them, taking advantage of the Python language, with which he wrote the plugin. Using LibreLogo is very simple: you open a document in Writer, you write some code in Logo language, as you would write any other text, and then you run it by pressing the appropriate button in the LibreLogo *toolbar*; if the code is correct, the turtle executes the code drawing a figure in the text, right in the middle of the page.

This design can then be managed like any other LibreOffice graphics. The first time you launch LibreOffice the Logo toolbar is not active. Therefore you need to activate it, with the appropriate menu command: **View** → **Toolbars** → **Logo**:



Once this is done, you must close the program and relaunch it to see, among the other toolbars also the LibreLogo one:



where the icons have the following meanings:

[3]Di *default* which means that this is the normal behavior. Those who use Linux (for Windows or Mac this problem does not exist) should take note of the following. Until the release of LibreOffice 4 excluded, install the LibreOffice extension from http://extensions.libreoffice.org/extension-center/librelogo. Instead, from version 4 on, install the Office-Library package directly, with the command *sudo apt-get install library-Library*. Then you need to restart LibreOffice, if it is already open. Successively, activate the toolbar in View-¿Toolbars-¿Logo. Close and relaunch

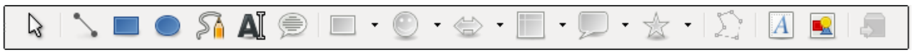
⬆	FORWARD 10	Forward by 10 points (we will see the meaning of point successively)
⬇	BACK 10	Back by 10 points
↷	LEFT 15	Left by -15 degree
↶	RIGHT 15	Right by 15 degree
▶		Executes the program. Starting with version 4.3, in a newly opened document it executes a standard sampled program.
■		Stops the running program
⏮	HOME	Brings the Turtle in its initial condition: at the center with nose up.
🗋	CLEARSCREEN	Cancels all the graphics - leaving the text.
▭		Allows to write a command and run it at once.
ABC		Adjust the whole text making it uppercase. At the same time, it translates the commands in the language of the document. Currently, the dictionary file for the Italian language is set up but the commands have to be written in English. I will fix this.

2.1 How to manage graphics in Writer

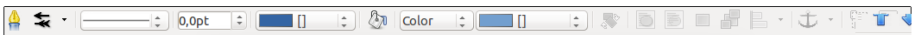
The interaction between LibreLogo and Writer is particular for graphics. It may seem cumbersome at first but you actually have to get used to it and learn two or three rules. The probably unique feature of LibreLogo is that by running [4] a script you get a graphic object in the same place where you have written the code, that's it in ODT document page. These objects are of "vectorial" type, that is, they are composed by a set of geometric objects.

[4] In jargon, by "running a program" we intended to execute all its instructions. Today, with modern languages, programs are often called *script*. In general, a program is a complete software and maybe also very complex. A *script* tends to be a smaller, more specific fragment of code but these categories may overlap widely.

They are different from *raster* or *bitmap*, that consist of a matrix of pixel[5]. The graphic objects produced by LibreLogo are completely similar to those produced with the handwriting tools available in Writer, accessible through the special *toolbar*, under the menu item **View** → **Toolbars** → **Drawing**:



As such, drawings made with LibreLogo can be moved, copied, or saved like any other graphic object. One useful thing to understand is that such objects are often actually a composition of distinct objects. We will do many of them in this manual. To use them as a single object, use the grouping function, as follows: first, you delimit the region that includes the objects to group, by selecting *pointer* in the drawing bar and then by outlining the desired rectangular box with the mouse and holding down the left button. Please note that the mouse cursor must be in the shape of an arrow and not the typical you have when inserting text, in the shape of a capital I, because this is where you insert text and not graphics. The fact that the graphic (and not textual) cursor is active is also understood by the fact that, at the same time, another toolbar is activated for controlling the graphics:



When you select the region containing the graphic objects, icons are activated in this bar, including the icon for the grouping function: . Pressing this will group all graphic objects in the selected region into a single graphic object that can be copied elsewhere or saved.

Another useful trick is to properly "anchor" the graphics to the document, where we have to use them. The key to determine the anchorage in the usual graphic bar is this: . By clicking on the arrow on the right of the anchor, you can select four anchor types: 1) "on page", 2) "in paragraph", 3) "in character" and 4) "as character". In the first case the graphics are associated to the page and do not move from it, in the second to a paragraph, in the third to a character and in the fourth case it behaves as if it were a character. What is the most appropriate anchorage is something that you learn from experience. Most of the graphics in this manual have been anchored "to the paragraph", except for small images that are in line with the text, as in the previous one, these are anchored "as a character".

These concern the management of graphics in Writer in general. Using LibreLogo, the only difference is that the graphics are produced through the instructions we put in the code. LibreLogo places the graphics in the middle of the first page of the document, even if the code text extends on the following pages. It may happen that the graphics overlap the text of the code itself. At first glance the result may be confusing and one can believe something wrong is going on. None of this. The graphics are produced to be used somewhere else. It is simply a matter of selecting it, as we have just described, and taking it elsewhere, in a clean page simply to see it clearly, or in some other document where it must be integrated.

[5]A closer look at the distinction between bitmap and vector images can be found at http://https://iamarf.org/2014/02/23/elaborazione-di-immagini-tre-fatti-che-fanno-la-differenza-loptis/

Bibliography

[1] Lewis Colleen. How programming environment shapes perception, learning and goals: Logo vs. scratch. `http://ims.mii.lt/ims/konferenciju_medziaga/SIGCSE'10/docs/p346.pdf`, 2015. Accessed: 2017-08-13.

[2] Weintrop David. Comparing block-based, text-based, and hybrid blocks/-text programming environments in introductory computer science classes. `http://dweintrop.github.io/papers/Weintrop-diss-4pager.pdf`. Accessed: 2017-08-13.

[3] Weintrop David e Wilensky U. To block or not to block, that is the question: Students' perceptions of blocks-based programming. `http://dweintrop.github.io/papers/Weintrop_Wilensky_ICER_2015.pdf`, 2015. Accessed: 2017-08-13.

[4] Weintrop David e Wilensky U. Using commutative assessments to compare conceptual understanding in blocks based and text based programs. `http://dweintrop.github.io/papers/Weintrop_Wilensky_ICER_2015.pdf`, 2015. Accessed: 2017-08-13.

[5] László Németh. Esempi di uso librelogo (ungherese). `http://www.numbertext.org/logo/logofuzet.pdf`. Accessed: 2017-08-13.

[6] Seymour Papert. *Mindstorms, Children, Computers, and Powerful Ideas.* Basic books, New York, 2 edition, 1993.

[7] Lakó Viktória. Manuale di comandi di librelogo. `http://szabadszoftver.kormany.hu/wp-content/uploads/librelogo_oktatasi_segedanyag_v4.pdf`. Accessed: 2017-08-13.