The Education Column

BY

Juraj Hromkovič

Department of Computer Science
ETH Zürich
Universitätstrasse 6, 8092 Zürich, Switzerland
`juraj.hromkovic@inf.ethz.ch`

Logo Environments in the Focus of Time

Jacqueline Staub
Department of Computer Science
ETH Zürich
jacqueline.staub@inf.ethz.ch

Abstract

Logo's history dates back more than half a century; time enough for several hundred environments to be developed which all rely on Papert's famous philosophy of learning. This article presents a selection of Logo environments that shaped the domain of novice programming. Quite contrary to the belief that Logo has gone extinct [19], this old language is currently experiencing a new upswing with a growing number of students who make their very first steps into the world of programming. At the end of our historic overview, we present a contemporary Logo environment that is used by tens of thousands of students every year and with which the old Logo legacy lives on even today.

1 Background

Before 1967, high-level programming languages such as FORTRAN, ALGOL, COBOL, and LISP were widely spread among professional programmers. These languages were popular choices among experts and allowed for various interesting applications once programmers got used to the syntactic and semantic details of the respective language. Many of these languages were, however, known to have an error-prone syntax that was difficult to get used to. Lisp, for example, was both popular for its expressiveness and notorious for its exuberant use of parentheses, which posed a risk for compile-time errors due to mismatched parentheses. In particular, novice programmers were prone to committing numerous errors which signified a rather tough start into the world of programming.

Researchers at Bolt, Beranek, and Newman (BBN) including well-known names such as Seymour Papert and Wally Feuerzeig came to the conclusion that the programming languages used at the time were poorly suited for the needs of beginners and that a solution to this problem could only lie in the development of a new programming language. They developed a special beginner-oriented programming language called Logo with which it should even be possible to introduce children to the world of programming, they claimed. This was a revolutionary assertion considering the fact that computers of the time were nothing like what they are today: a computer was a very large and expensive machine, and hardly any individual could afford one. The idea that computers could be an integral part of a classroom was by no means common.

Between the late sixties and today, more than 300 Logo implementations were implemented [11] and, together, they describe the story of how the world's first programming language for beginners came about and evolved over time. An article by Solomon, Harvey, Kahn, Lieberman, Miller, Minsky, Papert, and Silverman provides a broader view on the history of Logo [2]. The remainder of this article presents a selection of environments that mark specific milestones in the development of Logo. At the end of this article we present a Logo environment that is actively used and which acts as a gateway to the world of Logo for tens of thousands of students every year.

2 The Beginning

The first versions of Logo were all written in a time when personal computers had not yet been invented. Instead, time-sharing systems were used, which meant that each machine was intended to be used by multiple users simultaneously. John McCarthy invented the first time-sharing system around 1962 at BBN [1].

Several years before, McCarthy had also designed the first version of Lisp which had become one of the most-used languages at the time, especially for AI purposes. So, when Feuerzeig and his team at BBN collaborated with Papert, they thought of a language similar to Lisp (that is equally expressive), however, with a lighter syntax (especially in terms of parentheses). The goal was to design a language that is targeted at students of all ages, and even professionals.

One of the first Logo school projects took place at the Muzzey Junior High School [5] where students learned to create their own word games by programming. Listing 1 shows what such an activity might have looked like. Students were asked to generate random sentences using word lists containing a number of adjectives, verbs, and nouns that can be concatenated into sentences. It is important to notice that turtle graphics was only introduced in the next version of Logo.

```
TO NOUN
  OUTPUT PICK [CATS DOGS MICE]
END

TO VERB
  OUTPUT PICK [CHASE [PLAY WITH]]
END

TO ADJECTIVE
  OUTPUT PICK [YOUNG OLD]
END

PRINT (SENTENCE ADJECTIVE NOUN VERB ADJECTIVE NOUN)
```

Listing 1: This program creates English sentences such as YOUNG DOGS CHASE OLD CATS or YOUNG CATS PLAY WITH YOUNG MICE. The program uses three lists which declare words belonging to either of the following three word forms: (1) nouns, (2) verbs, (3) adjectives. The last line of code creates a sentence that picks and concatenates words from these lists in a way that fits the structure of English sentences. Note how this program uses the standard Logo syntax (e.g., for procedure declarations), yet it does not produce graphical output. The first version of Logo came without turtle graphics.

Daniel Bobrow implemented the first draft of Logo on one of BBN's time-sharing machines using Lisp as the host language. For simple word games, many of Lisp's built-in commands like `FIRST` and `BUTFIRST` (that is `CAR` and `CDR`) proved useful and were directly offered to the user through the Logo interface. Indeed, while this first version of Logo was supposed to be a beginner-friendly programming language for kids, it could also be used as a full-fledged Lisp dialect which allowed everything a normal Lisp would allow as well. This was a necessary requirement in order to design a language that could be used by the full spectrum of users – from beginners to experts. Some later implementations stuck with the choice to integrate an entire Lisp into their grammar, despite being hosted in other languages. The discussion sometimes went so far as to suggest that all Logo implementations which do not contain a full Lisp implementation are "only Logo in name" [2]. Nowadays, such a statement is questionable since Lisp is certainly not the only option for beginning programmers to continue with. Python has proven to be a viable alternative to Lisp [3] and, with its turtle graphics extension, Python allows for an elegant transition from Logo as well.

In contrast to more recent implementations of Logo that can usually be run on many different machines and different operating systems, early versions were produced for one specific system only. Machines like SDS 940, PDP-1, and PDP-10 were available at BBN and were thus convenient platforms to be used in the first three implementations of Logo. All these systems relied on a time-sharing mechanism and could be accessed by children. For this, they used computer terminals that were located at schools and which were connected to one of these bigger machines at BBN.

3 Introducing the Turtle

In 1969, Solomon and Papert founded a new group at MIT (so-called *Turtle Group*) that specifically focused on the development and continuation of the Logo idea. They added the turtle as a new microworld to Logo. At the time, displays were still rare and very expensive, and thus Papert and his colleagues strived for a "tangible" experience. Inspired by William Grey Walter's robots Elmer and Elsie [4], the Turtle Group decided to build their own version of such a floor robot. Two floor robots were crafted at MIT in the year 1970, both of which could be steered using Logo.

Harold Abelson implemented the first *display turtles* as a virtual counterpart of floor robots. In later implementations the support for physical floor robots was completely omitted since most schools did not have any robots at hand. Only in 1987 was the potential of floor robots rediscovered thanks to a collaboration with Lego in LEGO TC Logo for the Apple II.

Between 1973 and 1976, Perlman proposed an approach to make even preschoolers learn to program [6]. Still, one commonality between all these different approaches (robots, screen turtles, and tangible programming interfaces) was that children interacted with a terminal that was linked to a central computing instance located at MIT. The machine at the university would then execute all given commands in a time-shared way.

4 A Shift Towards Microcomputers

With the advent of microcomputers around 1980, Logo experienced a rise in popularity and coverage. Over the next 40 years, several hundred different Logo implementations were invented. The first such implementation on a microcomputer was Pascal Logo in the year 1977. Among others, Texas Instruments had started to work on home computers which were considerably cheaper than the previously-used mainframe computers that could usually only be afforded by larger companies and universities. Cecil Howard Green, the founder of Texas Instruments, was planning to create a new generation of home computers (the TI 99/4) that was also supposed to be shipped to schools. For this purpose, he wanted Logo to be part of the system.

The TI 99/4 was a machine that came with its own individual sprite chip that would support the drawing functionality of turtle graphics. One significant constraint it had, however, was its limited memory. In its default configuration, the machine came with only 256 bytes of RAM which could be extended to at most 32kB using external expansion cards (which, however, led to a massive slowdown of the system). The only high-level programming language that was supported by the TI 99/4 was Pascal, which was therefore chosen as the host language for this new Logo implementation.

Pascal Logo was not the only Logo implementation of its era that struggled with memory constraints. The developers of Music Logo had to take drastic measures to offer a music extension and they decided not to offer turtle graphics instead. The topic of memory management was a topic in Logo long before other mainstream languages started to offer automatic garbage collection. Logo pioneers around the globe seemed to agree that `malloc()` and `free()` were not something young programmers should have to bother with – even more so on systems with tightly limited memory.

5 A Debate About Syntax

One of the most successful home computers around the eighties was the *Apple II*. It contributed to the movement away from large and expensive mainframe computers and towards smaller, more-affordable home computers that were much more wide-spread and accessible for the broad public. Seeing that the Apple II was used more and more widely, two companies decided to implement their own version of Logo for this computer: First, *Terrapin Logo* was implemented, later LCSI followed with *Apple Logo*. Most later Logo versions are descendants of either of these two implementations.

One of the main differences between Terrapin Logo and LCSI Apple Logo lies in their different understanding of the conditional statement `if`. Terrapin decided for a verbose syntax of the form `if-then-else` (see Listing 2a) with the argument that this notation would favor beginners who can relate the syntax to spoken English. With this adaptation, however, conditionals become syntactic special cases. LCSI decided not to adopt Terrapin's syntax proposal and instead went for a notation that follows the *proceduralization approach* known from Lisp (that is, the approach tries to treat all linguistic elements like procedure calls). Listing 2b shows the notation LCSI chose for conditionals. Note that the keywords `then` and `else` have been replaced by two *instruction lists*, the first corresponding to the *then*-case, the second to the *else*-case.

Following Lisp's *code-as-data* concept, LCSI Apple Logo tried to describe the conditional statement `if` as a procedure with three arguments: The first argument evaluates to a Boolean, while the second and third arguments are interpreted as instruction lists. An instruction list is a dedicated data structure for representing

```
if true then fd 100 else bk 100
```

(a)

```
if true [fd 100] [bk 100]
```

(b)

Listing 2: Terrapin Logo syntax on the left (a) and LCSI Apple Logo syntax on the right (b).

code – in its raw form this data structure is not interpreted but just treated as a piece of information (like any other data type). Using the special command `run`, it is possible to parse the content of an instruction list and execute it. This corresponds to Lisp's *code-as-data* concept.

Using the *duality between code and data*, even control structures like `repeat` can be treated as simple procedure calls. Listing 3 demonstrates how Logo's looping construct `repeat` can be implemented in Logo using recursion. Once written, such a command can be used in exactly the same way as the traditional `repeat` statement (e.g., `myrepeat 4 [fd 100 rt 90]`).

```
to myrepeat :n :instructionlist
if :n>0 [
  run :instructionlist
  myrepeat :n-1 :instructionlist
]
end
```

Listing 3: Using a dedicated data respresentation for code, it is possible to treat even control structures like `repeat` as procedure calls. All it takes to implement `repeat` in Logo is the availability of instruction lists, the command `run`, and recursion.

Both Terrapin Logo and LCSI Apple Logo chose the same syntax for `repeat`, while their choices for `if` differ. Both sides had arguments for their respective decision: Terrapin argues that a simplified syntax would make the conditional statement more accessible for novices, while LCSI argued that their syntax was more uniform and thus especially useful for advanced programmers who wish to implement their own Logo interpreter. Both choices are sensible, although they aim at different user groups with different experience levels.

6 Object-Oriented Programming and Logo

As Logo grew older, new programming paradigms evolved around it, and so some Logo versions specialized in some of these new directions. One such example is *ObjectLogo*, a Logo implementation from the year 1986 which incorporated the ideas and concepts from object-oriented programming into Logo [7]. The implementation is designed to be an extension of LCSI Apple Logo and all previous teaching materials and common Logo notations could still be used. New features related to object-oriented programming were meant for advanced Logo programmers while beginners could still write the same programs as before, for instance the program in Listing 4 which draws a triangle without making use of object-oriented programming.

```
fd 100 rt 90 fd 100 rt 135 fd 141
```

Listing 4: A simple Logo program to draw a right-angled triangle. This program does not involve object-oriented features but it is still valid syntax in ObjectLogo.

The same triangle can also be drawn using object-oriented programming. ObjectLogo allows programmers to create multiple instances of the Turtle which all have their own internal state and behavior. New actors can be instantiated using the common `make` command and a special command `something` that is used to create a new object (see Listing 5, lines 1 and 2), and each object has its own local fields and methods which can be invoked using a special `tell` command. The `tell` command takes two arguments: first, the name of an object whose state or behavior shall be changed, and second an instruction list. The object reacts by executing all instructions that were provided in the instruction list.

The example in Listing 5 shows how two agents (`turtle1` and `turtle2`) collaborate to draw the same triangle as shown in Listing 4. In line 3, the first agent is told to draw the first two sides of a triangle. Then, `turtle2` finishes the drawing by connecting the last line. Note that `turtle2` is aware of the other agent's position. The command `setheading towards :turtle1` causes it to face `turtle1`.

```
1 make "turtle1 something
2 make "turtle2 something
3 tell :turtle1 [fd 100 rt 90 fd 100]
4 tell :turtle2 [setheading towards :turtle1 fd 141]
```

Listing 5: In ObjectLogo, multiple turtles can be instantiated and they can be addressed individually. In this program, two turtles are used to draw a right-angled triangle in collaboration. A first agent called `turtle1` is used to draw the first two sides of the triangle, another agent (named `turtle2`) is used to finish the work.

In contrast to non-object-oriented versions of Logo, ObjectLogo allows each actor to have its own local methods that can be used to overload pre-existing global commands. For instance, among three turtles each could have its own local version of the command `fd`. One turtle may draw a thick line, another one draws the line in red, while the last turtle draws a dotted line – all reacting to the same command `fd`. Listing 6 shows how such an implementation would look like. Three actors (`turtle1`, `turtle2`, and `turtle3`) have their own local versions of `fd`. The keyword `howto` indicates the beginning of a local procedure definition and corresponds to the global keyword `to`. In all three cases, a local command `fd` is defined which internally relies on the global implementation of the command `fd` (i.e., the keyword `usual` invokes the global version of `fd`). Each actor has their own version of `fd` and will react differently to the statement `tell :turtleX [fd 100]`.

```
tell :turtle1 [howto fd :distance
                setpw 5 usual.fd :distance
              ]

tell :turtle2 [howto fd :distance
                setpc red usual.fd :distance
              ]

tell :turtle3 [howto fd :distance
                repeat :distance/2 [
                  pd usual.fd 1
                  pu usual.fd 1
                ]
              ]
```

Listing 6: Local procedures can be used to overload global definitions. Different objects can be specialized on demand and each will use its own local version of procedure calls during execution.

ObjectLogo was an advanced implementation of Logo that aimed at teaching modular design by using the object-oriented programming paradigm. The environment allowed programmers to use several turtles which each have their own local environment with encapsulated variables and methods. Different objects could inherit features from one another and specialize them on demand. In addition to programming turtles, ObjectLogo also allowed most UI elements to be programmed such as menus, windows, files, and dialogues. All those elements were implemented as objects.

ObjectLogo was platform-dependent and could only be used on Macintosh machines. It was updated until the year 2000 but is no longer maintained and cannot be used on modern computers any longer.

7 Platform Independence and Standardization

By the late eighties, the computer world had stabilized around four common operating systems: DOS, Windows, Mac, and Unix. The majority of all machines ran one of these four systems and, yet, most Logo implementations were not designed to be portable between different operating systems. The consequence was a large variety of different Logo implementations, all crafted for different platforms and all with slight differences in terms of their internal implementation.

In 1988, Brian Harvey and his team at UC Berkley decided to develop a Logo version that was supposed to be portable across all common operating systems of the time. Their goal was to design a system that could be run among DOS, Windows, Unix, and Mac [2], and to standardize the graphical output by scaling it such that the graphics fit on all screens [8].

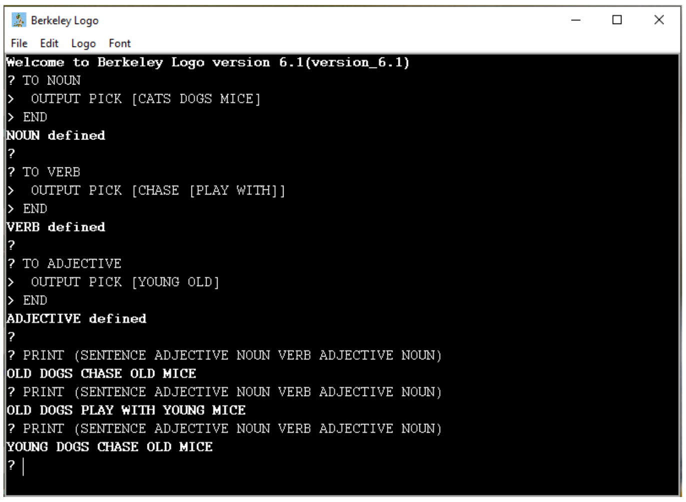
Berkley Logo was designed as a console application, a window with only one interface for the programmer to interact with (see Figure 1). This interface was a text-based interface similar to a typical command-line interpreter. Even though the application only contained one window, it still had more than one functionality this window could be used for. Specifically, the console window in Berkley Logo was used for three different purposes:

1. *Procedure invocations*: Firstly, the window can be used to execute built-in or user-defined procedures. The user is provided with an interactive programming experience in a so-called *REPL* (i.e., read–eval–print loop); an environment that allows programmers to piecewise execute code snippets. Each command may change the program state, which can be observed and inspected by the programmer. Unlike the LISP and Python REPL, however, Logo's REPL does not print results to arithmetic calculations such as `1+2`, but instead the outcome of such an arithmetic expression has to be printed explicitly using the `print` command.

2. *Procedure definitions*: New procedures can be defined using the common Logo notation `to ... end`. Once the keyword `to` has been detected, the environment enables multi-line input which can be closed again upon entering the `end` keyword. Lines which have already been executed cannot be revoked but an existing procedure can be edited and corrected. All versions of previously-executed procedures and procedure declarations remain visible by scrolling back through the history of previous activity.

3. *Output/history/error messages*: Finally, the window can be used as an output mechanism and as a history of previous activities. Besides the text related to procedure invocation, declaration, and their respective output, the window can also be used to issue error messages in case the input was not correct.

For turtle graphics applications, the window is split in two parts as shown in Figure 2. While the bottom part of the window is the same as before, the top part is used to illustrate the corresponding graphical output. The top part shows a canvas with a turtle. All movement and rotation commands leave a visible effect and can



```
Welcome to Berkeley Logo version 6.1(version_6.1)
? TO NOUN
>  OUTPUT PICK [CATS DOGS MICE]
> END
NOUN defined
?
? TO VERB
>  OUTPUT PICK [CHASE [PLAY WITH]]
> END
VERB defined
?
? TO ADJECTIVE
>  OUTPUT PICK [YOUNG OLD]
> END
ADJECTIVE defined
?
? PRINT (SENTENCE ADJECTIVE NOUN VERB ADJECTIVE NOUN)
OLD DOGS CHASE OLD MICE
? PRINT (SENTENCE ADJECTIVE NOUN VERB ADJECTIVE NOUN)
OLD DOGS PLAY WITH YOUNG MICE
? PRINT (SENTENCE ADJECTIVE NOUN VERB ADJECTIVE NOUN)
YOUNG DOGS CHASE OLD MICE
? |
```

Figure 1: Berkley Logo with a the same pure-text example as previously shown in Listing 1. For this use case, they provide a single-window interface similar to a typical command-line interpreter.

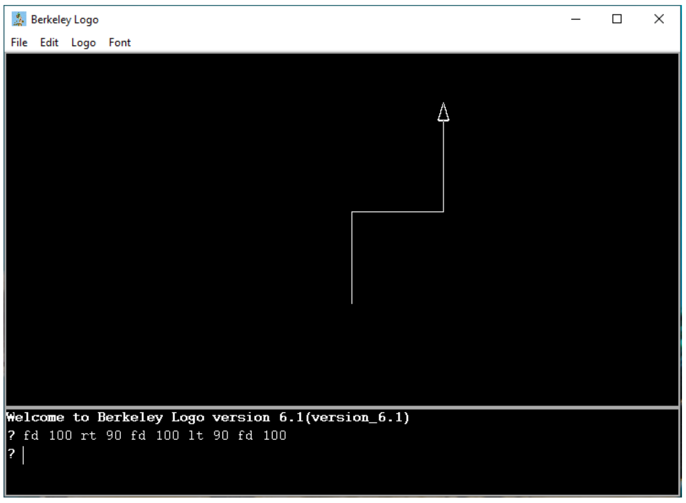
be observed on this screen, just as in all other implementations that make use of turtle graphics.

Berkley Logo is one of the oldest Logo versions that is still running today and works on modern computers. It has influenced multiple environments between the nineties and today. One main point that has been improved in more recent versions is the graphical user interface, as we will see in the next section.

8 Advanced Graphical User Interfaces

In the eighties, graphical user interfaces slowly started to evolve and during the nineties this progress continued. Step by step, the previously used command-line interfaces, as known in DOS, were replaced by graphical ones. Windows played an important role in this process. While MS-DOS was still widely used in the eighties, Windows started to move away from DOS with the introduction of Windows 3.0 – a system that came with an advanced graphical user interface which appealed the population. Windows 95 was the first Windows version that had MS-DOS fully integrated into the system (in contrast to the previous standalone DOS systems which could be booted without a graphical user interface). Microsoft swiftly moved away from DOS and towards a more graphical interface which made the computer more accessible to the wider public.

In 1993, a new Logo version called *MSWLogo* (Microsoft Windows Logo) was published specifically for Microsoft Windows. This Logo version used Berkley



```
Welcome to Berkeley Logo version 6.1(version_6.1)
? fd 100 rt 90 fd 100 lt 90 fd 100
? |
```

Figure 2: Berkley Logo with a simple turtle graphics example. For the purpose of displaying visual output, the screen is split in two parts.

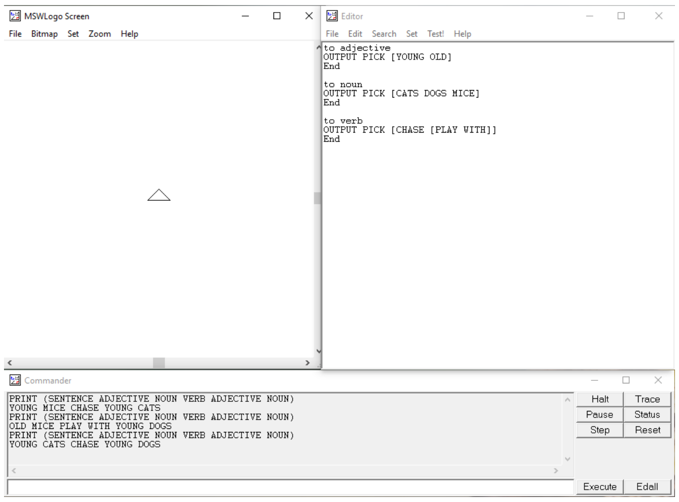
Logo as its core, but the designers enhanced it with a couple of additional features – first and foremost an improved user interface (see Figures 3 and 4). Instead of just one command-line window, MSWLogo provided three windows for three different functionalities: (i) a canvas (used for graphical output), (ii) an editor (used for displaying procedure declarations), and (iii), an input line with included history.

Debugging features were previously hidden features a programmer would only know of if she or he carefully read the user manual. In MSWLogo, debugging features such as `pause` and `trace` were added as visual buttons on the user interface. This simple adaptation made programmers more aware of debugging support which could help during troubleshooting.

MSWLogo was widely used among Windows users but it could not be used with any operating systems other than Windows. Two successors (*XLogo* and *XLogo4Schools*) were developed with the ambition to solve this problem. Both Logo environments were platform independent and both of them were widely used in programming courses across Switzerland some years ago.

9 A Contemporary Form of Logo Learning

Logo may be old but it still has not lost its political value and scientific appeal. In 2016, the development of *XLogoOnline* started as the main contribution of a master thesis at ETH Zurich [10] and by now it is actively used in Swiss schools by thousands of pupils every year. XLogoOnline is the last Logo version in the genealogy presented in this article and it builds on top of most Logo versions that



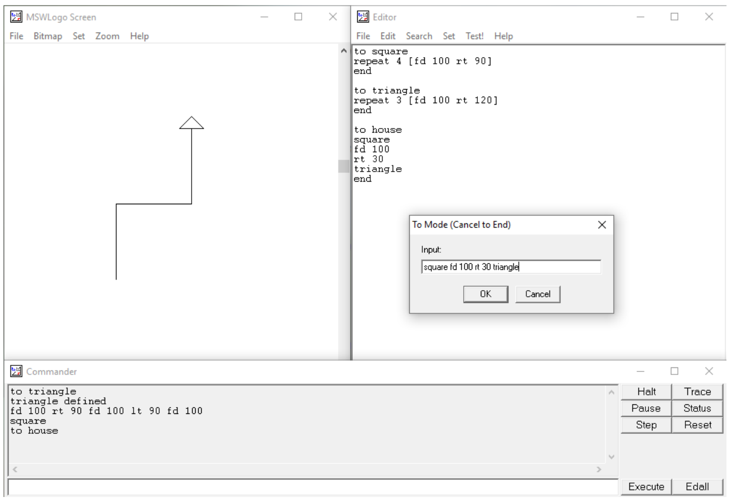
```
to adjective
OUTPUT PICK [YOUNG OLD]
End

to noun
OUTPUT PICK [CATS DOGS MICE]
End

to verb
OUTPUT PICK [CHASE [PLAY WITH]]
End
```

```
PRINT (SENTENCE ADJECTIVE NOUN VERB ADJECTIVE NOUN)
YOUNG MICE CHASE YOUNG CATS
PRINT (SENTENCE ADJECTIVE NOUN VERB ADJECTIVE NOUN)
OLD MICE PLAY WITH YOUNG DOGS
PRINT (SENTENCE ADJECTIVE NOUN VERB ADJECTIVE NOUN)
YOUNG CATS CHASE YOUNG DOGS
```

Figure 3: MSWLogo uses three windows: (i) canvas on the top left, (ii) editor on the top right, and (iii) input line and history



```
to square
repeat 4 [fd 100 rt 90]
end

to triangle
repeat 3 [fd 100 rt 120]
end

to house
square
fd 100
rt 30
triangle
end
```

```
square fd 100 rt 30 triangle
```

```
to triangle
triangle defined
fd 100 rt 90 fd 100 lt 90 fd 100
square
to house
```

Figure 4: Turtle graphics commands are drawn into the canvas, while new procedure declarations are written into the editor.

were presented before.

Four attributes characterize the environment's most important features:

- *Focus on K–6*: Originally, Logo was designed as a language that suits both the needs of beginners and experts. That is why many early versions included highly advanced features from the Lisp universe. But in the end, Logo was established mostly as a language for novices and it could not gain the same popularity among experts. In the development of XLogoOnline, a special focus was put on the age range from kindergarten to grade six (age 6 to 12 years). The programming environment is designed for teaching the basic concepts that are part of a spiral curriculum that is widely used in Swiss schools [13, 14, 15].

- *Platform independence*: Having software that is not bound to one platform but that can be used from many different devices and operating systems was already considered an important feature thirty years ago. The same feature is still relevant and can be considered especially important in the context of schools. IT administrators at Swiss schools invest an average of 2.5 hours per computer and school year in the installation and maintenance of software [12]. The easier it is to maintain software, the more time a teacher can spend on the preparation of lessons instead. For this reason, XLogoOnline was designed as a web application that uses JavaScript as its host language. In addition to portability between platforms, the choice of web technologies also allows for simple deployment without the requirement of installing or maintaining software on the client side.

- *Proactive error handling*: Learning to cope with errors is one of the most crucial aspects of a young programmer's experience. Bugs cannot be prevented, and every programmer eventually needs to be able to identify, locate, and fix programming mistakes by her- or himself. From the beginning, Logo had built-in error detection mechanisms [2]. These tools are however comparatively crude considering the great progress the professional community has made in terms of automated error diagnosis and debugging support throughout the last decades. XLogoOnline provides a variety of modern troubleshooting tools including compile-time checks for structural errors [16] and a reverse debugger [17].

- *Support for physical and virtual turtles*: Although the first versions of Logo usually came with integrated support for both physical floor turtles and virtual screen turtles, most environments in the eighties and nineties did not include support for floor robots any longer since schools "did not typically have any floor turtles they could connect with" [2]. Nowadays, physical robots have regained popularity and many Swiss schools bought physical floor robots for educational purposes. We have decided to provide potential connections to several floor robots (e.g., BlueBot and Root) such that each programmer has the option to choose between the screen turtle or a floor robot to accomplish their tasks.

XLogoOnline is a small part of a long and extensive history of how programming has found its way into general education. History has shown that the same idea can be realized in many different ways – during the past two decades, several alternatives to Logo programming were proposed and traditional turtle geometry has been overshadowed by new application domains such as digital storytelling using Scratch [20]. Contrary to the belief that Logo is dead [19], however, XLogoOnline proves that Papert's legacy still lives on and that Logo is finally experiencing a new upswing.

10 Conclusion

Logo is not only a famous programming language but also a philosophy of education and, for its inventor, it represented a critique against the traditions of the US school system during the seventies and eighties [18]. The background of this story is that, at that time, technology consumption in US schools started ramping up and computer-aided instruction gained widespread acceptance. During this phase, computers were installed in many classrooms around the country. However, they were rarely used for constructive activities such as programming. Papert opposed this practise and proposed computers be used as a tool for creative problem solving instead of drill-and-practice routines. That is, rather than putting students merely into the role of technology consumers, Papert envisioned them to become creative inventors of technology. Seeing the path computer-aided instruction had taken, Papert considered Logo an "anti-schooling project," which in many ways contradicted the common trend that the school system seemed to be adopting at the time [18].

Within the last fifty years, education has changed and constructivist ideas finally found their way into general education. Across the globe, more and more countries decided to introduce computer science into their public school curricula and finally Logo has a chance to be re-introduced in a large scale after it had believed to be extinct [19]. While programming environments have evolved and new approaches have been brought up in the meantime, there is still an active Logo community and thanks to new programming environments, Papert's legacy lives on and finally reaches a new upswing in central-Europe.

References

[1] John McCarthy, S. Boilen, E. Fredkin, and J. C. R. Licklider. A time-sharing debugging system for a small computer. In *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*, pages 51–57, 1963.

[2] Cynthia Solomon, Brian Harvey, Ken Kahn, Henry Lieberman, Mark L. Miller, Margaret Minsky, Artemis Papert, and Brian Silverman. History of Logo. *Proceedings of the ACM on Programming Languages*, 4(HOPL):1–66, 2020.

[3] Juraj Hromkovič, Tobias Kohn, Dennis Komm, and Giovanni Serafini. Combining the power of Python with the simplicity of Logo for a sustainable computer science education. *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP 2006)*, LNCS 9973, pages 155–166, Springer 2016.

[4] Peter F. Bladin. W. Grey Walter, Pioneer in the electroencephalogram, robotics, cybernetics, artificial intelligence. *Journal of Clinical Neuroscience*, 13(2):170–177, 2006.

[5] Seymour Papert. Mindstorms: Computers, children, and powerful ideas. *NY: Basic Books*, 1980.

[6] Radia Perlman. Tortis (Toddler's own recursive turtle interpreter system). 1974.

[7] Gary L. Drescher. Object-oriented Logo. In *Artificial Intelligence and Education*, volume 1, pages 153–165, 1987.

[8] University of California. Berkley Logo user manual, 2020a. https://people.eecs.berkeley.edu/~bh/usermanual.

[9] Brian Harvey. Berkley Logo user manual. *University of California Berkley*, 1993.

[10] Jacqueline Staub. XLogoOnline – A web-based programming IDE for Logo. Master's thesis, ETH Zürich, 2016.

[11] Pavel Boytchev. Logo tree project, 2007.

[12] SFIB Schweizerische Fachstelle für Informationstechnologien im Bildungswesen. ICT und Bildung in der Schweiz, 2004. https://archiv.educa.ch/sites/default/files/20121003/ictund_bildung_2004.pdf.

[13] Juraj Hromkovič, Dennis Komm, Regula Lacher, and Jacqueline Staub. Teaching with Logo philosophy. In *Encyclopedia of Education and Information Technologies*. 2019.

[14] Juraj Hromkovič. *Einführung in die Programmierung mit LOGO*, volume 206. Springer, 2010.

[15] Juraj Hromkovič. *Einfach Informatik 5/6: Programmieren. Primarstufe*. Klett und Balmer, 2018.

[16] Martina Forster, Urs Hauser, Giovanni Serafini, and Jacqueline Staub. Autonomous recovery from programming errors made by primary school children. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP 2018)*, LNCS 11169, pages 17–29. Springer 2018.

[17] Renato Menta, Serena Pedrocchi, Jacqueline Staub, and Dominic Weibel. Implementing a reverse debugger for Logo. In *In International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP 2019)*, LNCS 11913, pages 107–119. Springer 2019.

[18] Angelos Agalianos, Geoff Whitty, and Richard Noss. The social shaping of Logo. *Social Studies of Science*, 36(2):241–267, 2006.

[19] Michael Tempel. The Turtle is dead: Rethinking Logo in the age of Kid Pix. 1995.

[20] Quinn Burke and Yasmin B. Kafai. The writers' workshop for youth programmers: Digital storytelling with scratch in middle school classrooms. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE 2021)*, 433–438, 2012