

ToonTalk and Logo

Is ToonTalk a colleague, competitor, successor, sibling, or child of Logo?

by Ken Kahn

Abstract

The answer is all of the above. ToonTalk is a colleague because it shares with Logo so many goals and ways of thinking (so nicely described in Papert's book *Mindstorms* [Papert 80]). It is a competitor because teachers and learners have a limited amount of time to devote to such things. It can be argued that ToonTalk is a successor to Logo because it is built upon more advanced and modern ideas of computation and interfaces. ToonTalk is like Logo's little sister – looking up to her big brother while striving to out do him. And ToonTalk is a child of Logo in that it grew out of experiences of what worked well and what didn't in using Logo.

A Brief Introduction to ToonTalk

ToonTalk ([Kahn 96], [Kahn 01]) started with the idea that perhaps animation and computer game technology might make programming easier to learn and do (and more fun). Instead of typing textual programs into a computer, or even using a mouse to construct pictorial programs, ToonTalk allows real, advanced programming to be done from inside a virtual animated interactive world.

The ToonTalk world resembles a modern city. There are helicopters, trucks, houses, streets, bike pumps, toolboxes, hand-held vacuums, boxes, and robots. Wildlife is limited to birds and their nests. This is just one of many consistent themes that could underlie a programming system like ToonTalk. A space theme with shuttlecraft, teleporters, and so on, would work as well, as would a medieval magical theme or an Alice in Wonderland theme.

The user of ToonTalk is a character in an animated world. She starts off flying a helicopter over the city. (See Figure 1.) After landing she controls an on-screen persona. The persona is followed by a dog-like toolbox full of useful things. (See Figure 2.)

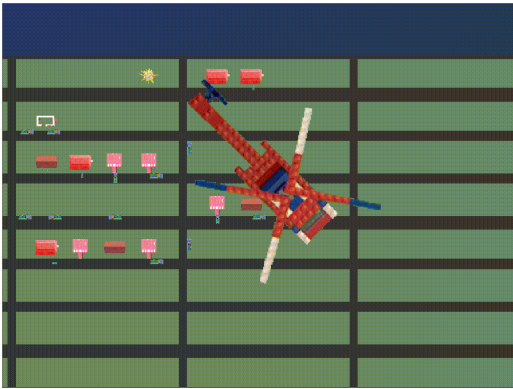


Figure 1 - Flying over the City

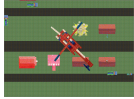
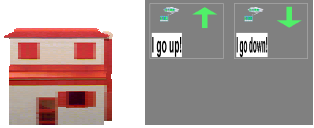







Figure 2 - Followed by the Toolbox

An entire ToonTalk computation is a city. Most of the action in ToonTalk takes place in houses. Homing pigeon-like birds provide communication between houses. Birds are given things, fly to their nest, leave them there, and fly back. Typically, houses contain robots that have been trained to accomplish some small task. A robot is trained by entering his "thought bubble" and showing him what to do. Robots remember actions in a manner that can easily be generalized so they can be applied in a wide variety of contexts. (See Figure 3.)



Figure 3 - Training a robot to double a number

Computational Abstraction	ToonTalk Concretization
computation	city 
actor process concurrent object	house or back of picture 
method clause	robot 
guard method preconditions	thought bubble 
method actions body	actions taught to a robot
message array vector	box 
comparison test	set of scales 
process spawning	loaded truck 
process termination	bomb





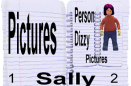
	
constants	number, text, picture 
channel transmit capability message sending	bird 
channel receive capability message receiving	nest 
persistent storage file	notebook 

Table 1 - Computer Science Terms and ToonTalk Equivalents

A robot behaves exactly as the programmer trained him. This training corresponds in computer science terms to defining the body of a method in an object-oriented programming language such as Java or Smalltalk. A robot can be trained to

- send a message by giving a box or pad to a bird;
- spawn a new process by dropping a box and a team of robots into a truck (which drives off to build a new house);
- perform simple primitive operations such as addition or multiplication by building a stack of numbers (which are combined by a small mouse with a big hammer);
- copy an item by using a magician's wand;
- change a data structure by taking items out of a box and dropping in new ones; or
- terminate a process by setting off a bomb.

The fundamental idea behind ToonTalk is to replace computational abstractions by concrete familiar objects. Even young children quickly learn the behavior of objects in ToonTalk. A truck, for example, can be loaded with a box and some robots. (See Figure 4.) The truck will then drive off, and the crew inside will build a house. The robots will be put in the new house and given the box to work on. This is how children understand trucks. Computer scientists understand trucks as a way of expressing the creation of computational processes or tasks.

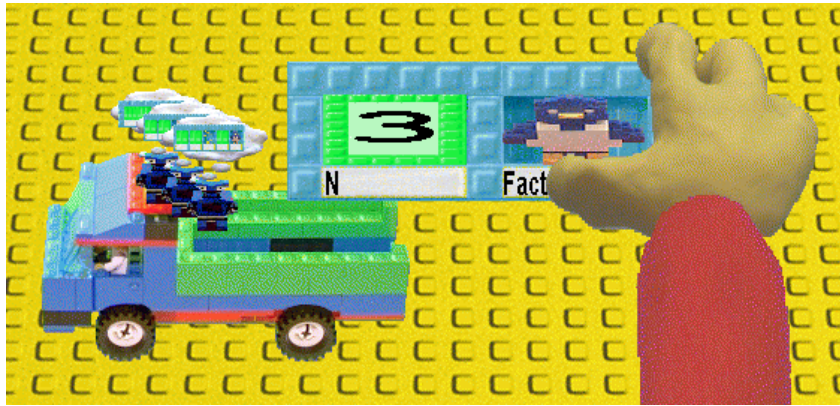


Figure 4 - A truck being loaded with robots and a box

Introduction to Logo

Seymour Papert once described Logo as taking the best ideas from computer science about programming languages and environments and “child-engineering” them [Papert 77]. When Logo was designed in the late 1960s the programming language Lisp was very innovative. Logo’s design was most heavily influenced by Lisp. Lisp programs consist of procedures that can compute with symbols, lists, and procedures, as well as the data types of conventional languages. Lisp programs can even construct and run other Lisp programs. Logo borrowed all of these powerful ways of expressing programs. Much effort was placed on making the syntax, names of primitives, and error messages child-friendly. Some of the more difficult aspects of Lisp, such as nested local variables and anonymous procedures, were dropped. (Recently local variables have returned to some dialects of Logo as primitives such as “localmake”.)

Soon after the birth of Logo it was extended to control floor turtles – robotic devices that can move and turn. Turtles also contain a pen that can be extended or retracted so they can draw on paper as they move around. Soon after floor turtles were introduced, display turtles were added that behave like floor turtles but are displayed on a computer screen.

Logo is both the name of a family of programming languages and the name of a broad set of ideas about learning where the Logo programming language plays an important role. The core idea is that computer programming can provide a particularly fertile field in which students can acquire and improve fundamental thinking skills. These include problem decomposition, representation, procedural thinking, debugging, reflection, and more. Programming can also be a powerful thinking tool for exploring and modeling other domains of knowledge from science, language, math, and art. The right programming language, the right way of teaching programming and problem solving, and the right context greatly

increase the odds that children will acquire these problem solving and thinking skills [Papert 80].

Colleagues

[*Warning: these first two paragraphs may only interest computer scientists.*] ToonTalk was designed as an attempt to once again child engineer the best ideas in computer science. Lisp was the best source of ideas in the 1960s, and concurrent constraint programming is the best source 30 years later [Saraswat 93]. Concurrent constraint programming is a synthesis of the ideas of concurrent logic programming and constraint logic programming. Rather than build upon the concept of procedures, concurrent constraint programming builds upon the notion of autonomous communicating agents. A procedural programming language like Logo is based upon the idea that you compose program fragments by passing arguments to program pieces called procedures. When a procedure has completed its task it returns, often passing back a value to its caller. The caller then proceeds. This is a sequential and restrictive view of computation.

Concurrent constraint programming replaces procedures by entities that some call "agents", others call "threads", and still others call "processes". We will use the term agents here. An agent is an autonomous activity that consists of some program fragments that define its behavior and a set of accessible variables that constitute its local state. The fundamental actions that an agent can perform are to construct new agents, add constraints to variables (called "telling"), and testing if some proposition is implied by the current set of constraints (called "asking"). Asking and telling constraints provide a very expressive way of describing the desired communication between agents and their synchronization. In the case of ToonTalk, only one limited kind of constraint can be expressed but it still provides a very rich communication mechanism. In ToonTalk the constraint that can be told is that something is an element of a multi-set or bag (i.e., a set that allows duplicates). This is expressed by giving something to a bird. It is then added to the items covering the bird's nest, which constitute a multi-set.

Building upon this one can do actor [Agha 87] or concurrent object programming [Kahn and Saraswat 90a]. The reader may wonder how a language based upon sophisticated notions such as constraints, implication, synchronization, and the like could possibly be appropriate for children. It would seem to be accessible only to those with advanced degrees in math, logic, and computer science.

This is where the idea of concretizing the underlying computational abstractions of ToonTalk shows its power. ToonTalk programmers need only think about birds, nests, trucks, and robots and the like. They understand these familiar objects only in terms of their straightforward behaviors – e.g. that birds take things to their nests. They don't need to learn about the underlying theory of concurrent constraint programming. This is

similar to the fact that Logo programmers need not know anything about Church's Lambda Calculus [Barendregt 84] to master Logo. But both Logo and ToonTalk benefit from having been built upon such strong theoretical foundations.

Besides sharing the idea of borrowing and then child engineering the best ideas in computer science, ToonTalk shares many goals with Logo. Both aspire to give children tools that empower them to be creative in new ways. Both wish to provide children with computational "thinking tools". Both hope to provide a fertile ground for children to explore and learn in new and particularly effective ways. Both hope to appeal to children so that they will use these systems without coercion from schools or parents. Sharing so much philosophy and sharing so many goals makes Logo and ToonTalk close colleagues.

Siblings

I think of Logo and ToonTalk as siblings as well. Maybe ToonTalk is Logo's kid sister who tries to copy or improve on what she sees her big brother has done. This historical connection is largely due to my early involvement in Logo and my subsequent attempts to improve on it. Soon after becoming a graduate student at MIT in 1973, my interest and involvement in the MIT Logo Project grew. Soon I was a member of the project, exploring ideas of how children might use Logo to construct programs that could process and generate English sentences and programs to produce animations (which was very challenging with the computers of 1975). I soon began to imagine better programming languages that were object-oriented and supported a limited kind of parallelism based upon "ticks" [Kahn and Hewitt 78]. Years later I explored how some of the ideas of logic programming might contribute to Logo [Kahn 83].

ToonTalk is not my first attempt to bring the ideas of concurrent constraint programming to children. While at Xerox PARC I build a system called Pictorial Janus [Kahn and Saraswat 90b]. Pictorial Janus programs are completely visual. Pictorial Janus animated the execution of programs. Pictorial Janus, while visual, was still formal and abstract and hence hard for children and most adults.

There is one great idea of Logo currently missing from ToonTalk. This is the idea of turtle geometry. I haven't added turtles to ToonTalk because I've focused instead on providing strong support for animation and giving behaviors to pictures. This support is based upon the idea of sensors and remote controls that can be manipulated by the programmer as well as by her robots. This framework has proven to be well suited for even young children who want to program animations or games [Playground 01]. A turtle package could be added to ToonTalk. It would be a valuable addition but it would not fit as well with the ToonTalk style of programming as turtle programming does with Logo.

Competitors

The reality of the world is that children, teachers, and parents have only so much time to be involved in Logo-like activities. Ideally, children should be exposed to several different computational tools and have the freedom to choose and switch between them. Ideally, children should be exposed to several different ways of thinking about computation since this could deepen and broaden their understanding and skills. After all, the best way to understand something is to understand it in multiple ways.

In attempting to obtain the unfortunately limited attention and resources of teachers, ToonTalk and Logo are competitors. Too many teachers and school administrators think that the time and effort of teaching children to use the most common applications (e.g. word processors, email programs, and web browsers) leaves no time for programming. Those enlightened teachers who see the value of Logo typically have too little time to devote to Logo activities as is. How can ToonTalk fit into this picture?

One answer is that ToonTalk is more appropriate for a wider age range than Logo. Unlike Logo, a ToonTalk user need not be able to read or type. And compared to single-key interfaces to Logo for young children, ToonTalk is much more expressive and flexible. Children as young as 5 or 6 can master the full set of ToonTalk elements. They understand boxes (data structures and variables), robot training (defining behaviors), robot's thought bubbles (conditionals), trucks (process spawning), birds (message passing), and more. While 5 year olds enjoy the basic elements of ToonTalk, university students can use ToonTalk to explore and visualize concurrent algorithms and distributed programming.

Another answer is that ToonTalk provides better support than Logo for a large range of programs. Due to ToonTalk's underlying concurrency and its extensive support for giving behaviors to pictures, it is a better tool than Logo for making most kinds of games and simulations. On the other hand, Logo is better suited for programs that draw or make heavy use of text. Game programming is very important, since games are what most children want to program.

A third argument is that ToonTalk supports remote collaboration and distributed programming in a manner that children can master. The ToonTalk model of concurrent computation generalizes to networked computations with the recent introduction of long-distance birds. A long-distance bird acts just like the ordinary birds of ToonTalk, except that it can fly to a nest on another computer. This enables children to collaborate by simply giving a copy of what they are building to a bird that takes it to a nest on the computer of another child. The second child can then run, pull apart, or modify what she received and send it back. Long-distance birds can also be the foundation for distributed programs such as networked multi-player games built by children.

A final argument in favor of ToonTalk is that children find it easier to learn. This is due to the self-revealing nature of the ToonTalk primitives and tools, the fact that there is no

syntax to master, the fact that ToonTalk programmers work out their programs with concrete examples and later generalize them, and that ToonTalk includes many learning tools [Kahn 98]. Some of these points are discussed in detail below.

Parent and Child

Despite the many wonderful things about Logo, it does have some shortcomings. Decades after the birth of Logo, I was able to address some of these problems while designing and building ToonTalk.

1. *Logo is sequential; the world and ToonTalk are concurrent.* Just look out of any window and you'll see lots of concurrent activity. And each thing you see typically has lots of internal concurrency – e.g. people are walking and chewing gum at the same time. Walking in turn has concurrent motions of components: arms and legs. Modern Logo implementations partially address this by providing threads, sometimes in the form of multiple simultaneous turtles. This is only a partial solution because there needs to be good ways of expressing communication and coordination between these threads. And unlike ToonTalk, conflicts such as race conditions and deadlock need to be dealt with. Some parallel versions of Logo such as StarLogo [Resnick 97] and NetLogo [NetLogo 01] do not support fully general concurrency and are too complex for young children or elementary school teachers.
2. *Logo rarely succeeds without exceptional teachers; ToonTalk is, to a large degree, self-teaching.* For Logo to succeed a student needs to be taught by someone who has a deep understanding of programming and who understands the broader ideas underlying Logo. In the 1980s Logo was widely used in schools in the US and elsewhere. And yet in most cases the children had an unsatisfactory experience [Yoder 94]. In an attempt to remedy this problem ToonTalk contains several learning tools including an interactive coach/guide character, a series of interactive tutorial puzzles, narrated demos, and more [Kahn 98]. One consequence of this is that many children have learned ToonTalk at home with little or no support from their parents. (While this antidotal evidence from parents is suggestive, a real study of this issue is really needed.) Of course, when available, a well-trained teacher is much more effective.
3. *When children first begin using Logo they typically aren't having fun; ToonTalk beginners do.* The mechanics of Logo programming involves typing syntactically correct commands, selecting items from menus, and responding to dialog boxes. The mechanics of ToonTalk programming involves using animated characters as tools, flying a helicopter, working with birds, trucks, robots, and bombs, and generally “living” in an animated game-like world. Not surprisingly children enjoy using ToonTalk even when not trying to achieve an explicit goal. Advanced users of Logo and ToonTalk probably derive equal pleasure from the creative and intellectually challenging programming tasks.

In a similar way to how C spawned C++, Basic spawned Visual Basic, or Logo spawned StarLogo, ToonTalk is a spawn of Logo that attempts to improve upon its parent.

Successor

For ToonTalk to be a clear successor to Logo it needs more than a better computation model. And it needs more than a new game-like user interface. And it needs to do more than enable its users to program with concrete examples and subsequently generalize. To be a successor, ToonTalk should be equal or better than Logo in all aspects. To achieve this ToonTalk would need to

1. provide turtle graphics
2. support better the building of text-based applications
3. support the building of GUI applications (i.e., applications that use windows, menus, and dialog boxes)
4. provide a way to see a program fragment in order to understand it (currently you can only activate a ToonTalk program fragment and watch it in action)
5. provide a way to edit a program fragment (currently you are limited to editing a robot's thought bubble, i.e. the conditional test, and re-training robots)

We have plans for dealing with the last two issues. The first three deficiencies are addressable within the ToonTalk framework but there are no current plans to do so. ToonTalk users have not asked for these abilities.

Besides the technical deficiencies listed above, ToonTalk lacks the worldwide active community of support that Logo enjoys. There are many Logo books, teacher training courses, the Logo Foundation, conferences such as EuroLogo and Logosium and the active on-line communities of the Usenet comp.lang.logo group and the LogoForum e-mail discussion list. The ToonTalk community is small but growing.

What other languages are related to Logo and ToonTalk?

Smalltalk began as a programming language for children when it was first implemented in 1972. It also borrowed and pioneered state-of-the-art computer science ideas, especially, the idea of object-oriented programming. By 1980, however, Smalltalk had evolved into a programming system for professionals. In recent years interest in children using Smalltalk has revived with efforts of the Squeak Project [Squeak 01]. Smalltalk meets Logo's "no ceiling" goal very well, but does not have a "low threshold".

AgentSheets [AgentSheets 01] is particularly well suited for simulations. It provides a grid upon which programmable agents live. Stagecast Creator [Stagecast 01] is also

focused upon simulations and uses a grid. Unlike Logo, ToonTalk, and Smalltalk, these systems trade-off generality for ease-of-use and simplicity. Creator is very simple and yet is based upon sophisticated computer science but it is not well suited for a very wide range of projects. AgentSheets is more general but more complex. Both systems meet Logo's "low threshold" goal but have mixed success regarding the "no ceiling" goal.

Basic is a programming language designed for beginners and children. The design philosophy is quite different from Logo and ToonTalk. Basic borrowed from then current programming practice rather than advanced computer science. Many powerful programming concepts were removed from Basic in the interests of simplicity and smallness. Basic, however, has continued to evolve and progress. A modern implementation like Visual Basic has come much closer to Logo. Some have even added turtles to Visual Basic. Basic is like an impoverished and incompetent cousin of Logo that has grown up to be nearly a full family member.

Conclusion

The relationship between ToonTalk and Logo is complex. ToonTalk is simultaneously a colleague, competitor, successor, sibling, and child of Logo. And let's hope they remain good friends.

Acknowledgements

I wish to thank Mary Dalrymple, Richard Noss, and Mikael Kindborg for their comments on earlier drafts of this paper. This paper was also strongly influenced by many on-line discussions in the Logo Forum and comp.lang.logo. In particular the discussion held at the end of 1998 inspired much of this paper. The discussion is archived at www.toontalk.com/English/logo.htm.

References

- [Agha 87] G. Agha, *Actors: A Model for Concurrent Computation in Distributed Systems*. The MIT Press, 1987.
- [Barendregt 84] Barendregt, H.P., *The Lambda Calculus - Its Syntax and Semantics*, Second Edition, North Holland 1984
- [Kahn and Hewitt 78] Ken Kahn and Carl Hewitt, "Dynamic graphics using quasi-parallelism", Proceedings of the ACM/SI GGRAPH Conference, August 1978.
- [Kahn and Saraswat 90a] Kenneth Kahn and Vijay Saraswat, "Actors as a special case of concurrent constraint programming", Proceedings of the Joint Conference on Object-Oriented Programming: Systems, Languages, and Applications and the European Conference on Object-Oriented Programming, ACM Press, October 1990.
- [Kahn and Saraswat 90b] Kenneth Kahn and Vijay Saraswat, "Complete visualizations of concurrent programs and their executions", Proceedings of the IEEE Visual Language Workshop, October 1990.
- [Kahn 83] Ken Kahn, "A grammar kit in Prolog", In M. Yazdani, editor, *New Horizons in Educational Computing*, Ellis Horwood Ltd., 1984. Also In Instructional Science and Proceedings of the AISB Easter Conference on AI and Education, Exeter, England, April 1983.
- [Kahn 96] Ken Kahn, "ToonTalk - An Animated Programming Environment for Children", *Journal of Visual Languages and Computing*, June 1996.
- [Kahn 98] Ken Kahn, "Helping children to learn hard things: Computer programming with familiar objects and actions", *The Design of Children's Technology*, A. Druin, Ed., Morgan Kaufmann, 1998.
- [Kahn 01] Ken Kahn, ToonTalk Web Site, www.toontalk.com
- [NetLogo 01] NetLogo Web Site, www.ccl.sesp.northwestern.edu/netlogo/
- [Papert 77] MIT Logo Project meeting notes, 1977.
- [Papert 80] Seymour Papert, *Mindstorms: Children, Computers, and Powerful Ideas*, New York, Basic Books. 1980.
- [Playground 01] Playground Research Project Web Site, www.ioe.ac.uk/playground.
- [Resnick 1997] Mitchel Resnick, *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*, MIT Press, Cambridge, MA, 1997.
- [Saraswat 93] Vijay A. Saraswat, *Concurrent Constraint Programming*, MIT Press, Cambridge, MA, ACM Doctoral Dissertation Awards, 1993.

[Yoder 1994] Sharon Yoder, "Discouraged? ... Don't despair! [sic]", Logo Exchange, ISTE 1994.