

ToonTalk – Steps Towards Ideal Computer-Based Learning Environments

Ken Kahn

Published as a chapter in Mario Tokoro and Luc Steels, editors, *A Learning Zone of One's Own: Sharing Representations and Flow in Collaborative Learning Environments*, Ios Pr Inc, June 2004.

Abstract

Many of the attributes of good computer-based learning environments such as support for collaboration, creativity, and community are widely agreed upon. Interfaces should be seamless. Content should be challenging without being frustrating. Authoring support should be flexible and powerful.

But what about *universality* in learning environments? Should learning environments support the specification and execution of arbitrary computations? And if one does support universality, should it be for more than just expert users?

I argue here that universality for all is both desirable and attainable. Furthermore, by universality I mean more than a theoretic equivalence to a Turing Machine, but rather an elegant and cognitively appropriate model of general computations. The classical notion of universality only addresses the ability to compute the values of functions, ignoring the ability to effectively use the input and output devices typically associated with desktop computers. Ideally, learners should be able to describe computations, including those involving the display, mouse, keyboard, or sound card. And they should be able to describe those computations in a language that is well-suited to their cognitive abilities.

A computer becomes whatever software instructs it to be. A learning environment that does not restrict the range of software that can be created and run is able to exploit all that a computer can be. Most learners can find something that is personally compelling among the literally millions of different kinds of things a computer can be. Software is a fundamentally new kind of medium. It is a medium where one expresses ideas that are given life by machines called computers.

Ideally, educational software should be *transparent*. To be transparent, software needs to be composed of modules that can be understood and *changed* by students as well as authors.

Previous attempts to provide universal universality (i.e., fully general computational tools for everyone) have had limited success. Logo [Papert 1980] and Smalltalk [Kay 81] both strive to be general purpose programming languages for all. To master these languages or their successors (e.g. Boxer [Boxer 02], StarLogo [Resnick 1997], and Squeak[Squeak 02]) requires a set of skills that many students fail to acquire. To program in these

languages one needs to be fluent in a set of computational abstractions (e.g. procedures, variables, and conditionals). And programs are constructed in the realm of abstract variables and not concrete values.

Everyone, however, can master tools capable of constructing arbitrary computations. (By “everyone” I mean students who are capable of learning other school subjects.) ToonTalk ([Kahn 96], [Kahn 02]) is a programming environment that greatly lowers the difficulty of programming without sacrificing power or expressibility. It does this by replacing computational abstractions with tangible concrete analogs and by supporting the ability to program with examples and subsequently remove details to obtain generality.

A Brief Introduction to ToonTalk

ToonTalk started with the idea that perhaps animation and computer game technology might make programming easier to learn and do (and be 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.)

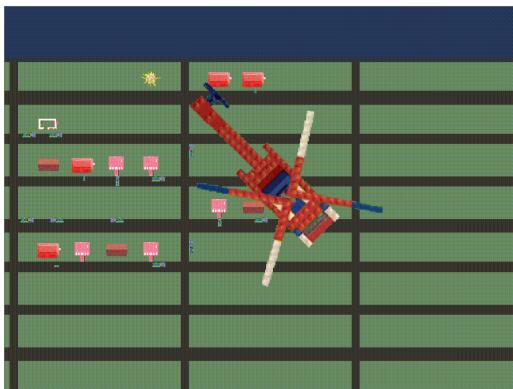


Figure 1 – Flying over the City



Figure 2 – Followed by the Toolbox

The ToonTalk city is where all computations take place. 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.)

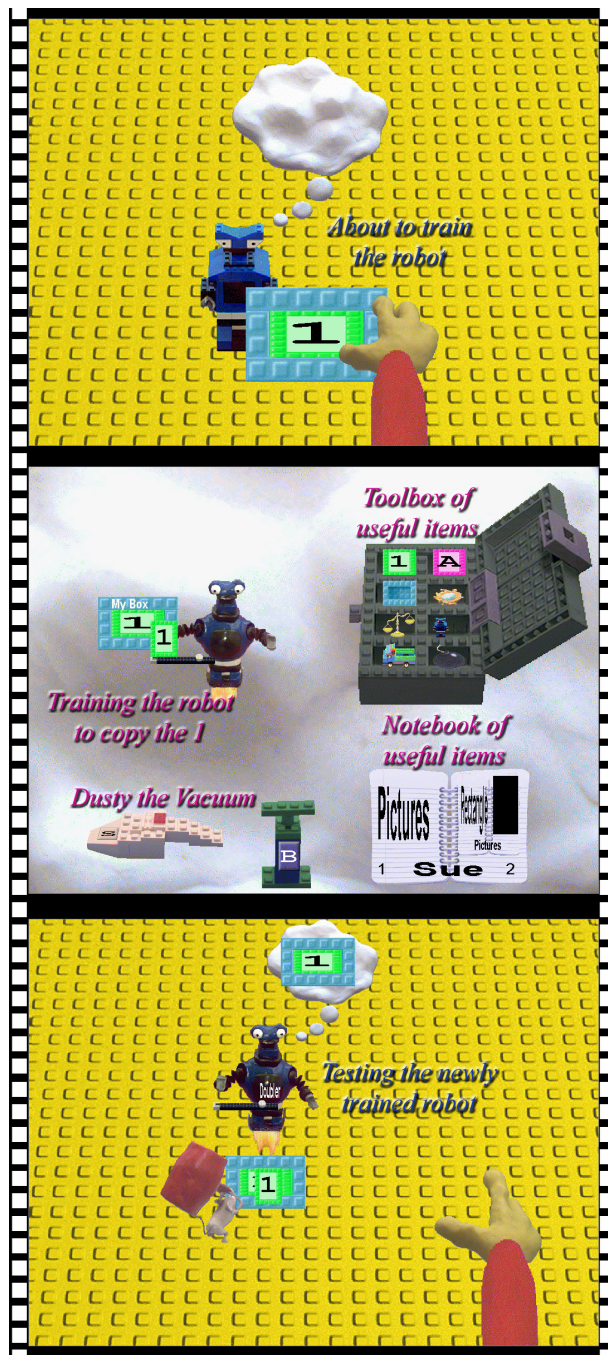
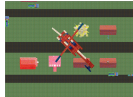





Figure 3 – Training a robot to double a number

Abstraction vs Concretization

A robot behaves exactly as the programmer trained him. In computer science terms, this training corresponds 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.

Computational Abstraction	ToonTalk Concretization
computation <i>a running program</i>	city 
actor, process, or concurrent object <i>an independent activity or behavior</i>	house or back of picture 
method or clause <i>the smallest coherent program fragment</i>	robot 
guard or method preconditions <i>conditions before running a program fragment</i>	thought bubble 
method actions or body <i>a sequence of actions</i>	actions taught to a robot
message or array or vector	box











<i>a container of items</i>	
comparison test <i>testing if something is more than something else</i>	set of scales 
process spawning <i>the creation of a new activity</i>	loaded truck 
process termination <i>the termination of an activity</i>	bomb 
constants <i>basic elements</i>	number, text, picture   
channel transmit capability or message sending <i>a way to send messages</i>	bird 
channel receive capability or message receiving <i>a way to receive messages</i>	nest 
persistent storage or file <i>a place to store things permanently</i>	notebook 

Table 1 - Computer Science Terms and ToonTalk Equivalents

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.

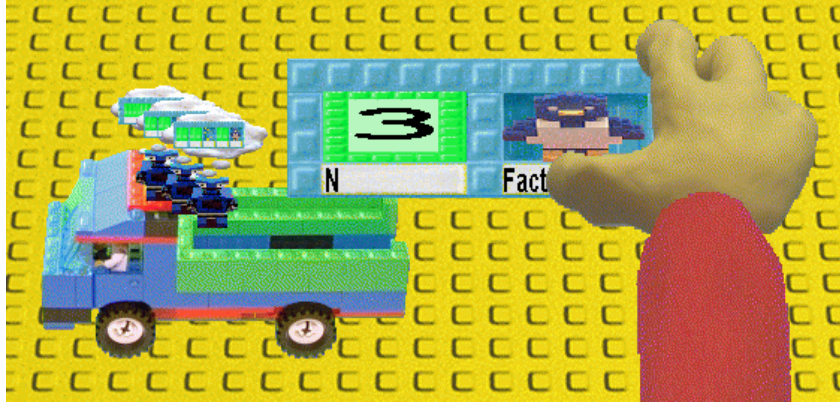


Figure 4 - A truck being loaded with robots and a box

ToonTalk as a Learning Environment

As Papert [Papert 1980] and others have observed, computer programming can be a fertile ground for learning general thinking skills. These include problem decomposition, component composition, explicit representation, abstraction, debugging, and thinking about thinking.

In addition, educators can build levels of software upon ToonTalk to help students learn other subjects. The Playground Project [Playground 01] explored the use of ToonTalk to enable young children to build their own video games. In doing so the children acquired an understanding of the mechanics of a complex artifact (a computer game). They learned how to play creatively with rules to make their own games. The children became good at taking apart a computational entity and creatively reassembling it in new ways, sometimes combining parts from different games.

This work relied upon the fact that things built in ToonTalk are modular and transparent. A child, for example, can take a simple Ping Pong game, remove the bouncing ball, and flip it over to see the robots that give the ball its behavior. If structured well, the ball's behavior is composed of sub-behaviors for bouncing off of other objects, for bouncing off the edges of the game area, and for its initial movement. A child can then find a scorekeeping mechanism, watch how its robots work, and change it to reflect the way she wants the score to be kept in her game. The modified scorekeeping program can then be added to the back of the ball. She then may decide her game should have bouncing fish and place all of these behaviors as a component on the back of a picture of a fish. If she wants lots of fish in her game she can copy the fish and the behaviors on the back are copied as well. She then need only drop her fish on an appropriate background picture.

The Playground Project also explored collaborative game design. Children were able to exchange games with other children (even in other countries) simply by giving their games to ToonTalk birds whose nests are on other computers. The birds then deliver the games (and associated messages) to the other children. These children then play, analyze, modify and review the games and send them back in the same manner.

A new three-year European research project began September 2002 called WebLabs. Its goal is to repeat the success of the Playground Project with a focus on science and math instead of computer games. ToonTalk is providing the infrastructure for building “transparent modules” for learning about specific topics like gravity, randomness, robotics, infinity, or bouncing. Children are using these modules to create their own simulations and games to explore these scientific topics. The children then embed their ToonTalk creations into active essays [Resnick 96] that are web reports that include runnable programs. They then publish their reports to facilitate collaboration and peer review.

Learning ToonTalk

Among the subjects that can be learned while inside ToonTalk is ToonTalk itself. Receiving expert instruction from a teacher or a textbook is just one way to learn ToonTalk programming.

ToonTalk provides four fundamentally different ways for children to learn ToonTalk on their own:

1. *Free play.* An open-ended, unconstrained, rich environment to explore and create things.
2. *A puzzle game.* A sequence of puzzles that gradually introduces the elements of ToonTalk and techniques for building programs.
3. *Pictorial instructions.* Sequences of pictures that show how to build programs.
4. *Demos.* Narrated demos showing various elements of ToonTalk and construction techniques.

A Safe and Self-revealing environment

Proponents of constructivism [Papert 1993] argue well for the position that the best, deepest, longest-lasting learning happens when the learner discovers and constructs the knowledge herself. ToonTalk has a “free play” mode designed to accommodate this kind of learning.

Exploratory learning is best supported by an environment that is *safe* and *self-revealing*. An environment is *safe* to explore if novice actions will not cause any permanent damage. For example, in ToonTalk there is a character named Dusty that acts like a hand-

held vacuum. A beginner exploring ToonTalk might pick up Dusty and vacuum up something important. However, Dusty doesn't destroy things, and he can be used in reverse to spit out all the things he has ever vacuumed up.

A *self-revealing* environment is designed so that an inquisitive explorer can discover what objects exist and how they behave. ToonTalk, for example, contains boxes. Even very small children discover on their own how to move boxes, how to put things into them, and how to take things out of boxes.

Good animation and sound effects help greatly in making an environment self-revealing. If a user holds something over an empty compartment of a ToonTalk box, she sees that part of the box wiggle and change color in anticipation. If she then clicks the mouse button, she sees an animation of the item leaving her persona's hand and falling into the compartment and hears an appropriate sound effect. If a force-feedback joystick is connected to the computer, she even *feels* the weight and other properties of objects.

It is very difficult to make a completely self-revealing environment. For example, in ToonTalk, boxes can be joined together and broken apart by actions that are often not discovered by children on their own. To completely explore ToonTalk, children need help.

ToonTalk includes an animated talking guide or coach named Marty to help a child explore ToonTalk. (See Figure 5.) Marty keeps track of what actions a user has performed. He also is aware of what item a user is holding or pointing to and tries to suggest an appropriate action in the current context. For example, a child holding a box who has put things in and out of boxes but hasn't joined two boxes together will hear a suggestion from Marty about how to join boxes.

Some children send Marty away, preferring to explore without any help. Others can be seen trying Marty's suggestions one after another. Children react to Marty differently depending upon whether he communicates by talk balloons, as in comics, or uses a text-to-speech engine to actually speak. For some children, reading is a slow and burdensome task.

Ideally, a self-revealing environment should also be *incremental*. An incremental environment may feel open-ended and rich but is designed so that certain objects or actions can be discovered only after others have been mastered. This helps reduce confusion and frustration that often results from the initial explorations of a rich and complex environment. Popular video games such as Nintendo's *Super Mario Brothers*® are excellent examples of incremental self-revealing environments. When a player starts these games she finds herself controlling an on-screen character. Initially all she needs to do is move. Soon she sees some coins and by walking into them they are acquired. Soon after there are coins that are not reachable without jumping, and the player experiments with a small set of buttons on the controller to discover how to jump to get those coins. After hours of play, the player has discovered a wide variety of actions her persona can perform and the properties of many different objects in the environment. Some video

games have on-screen characters that reveal some of the harder-to-discover game elements. Such characters were the inspiration for Marty, ToonTalk's guide.

Puzzle sequences as tutorials

A carefully designed sequence of puzzles can be very effective pedagogically. Many computer and video games use puzzles as an effective and fun tutorial. *Lemmings*® and *The Incredible Machine*® are two good examples. The idea is to present a sequence of puzzles that introduces new elements or actions one at a time in a simplified or constrained environment.

A series of puzzles is more appealing to most children when it is embedded within a narrative adventure. The ToonTalk puzzle game starts with a brief “back story”. An island is sinking, and a friendly Martian named Marty happens to be flying by and rescues everyone. He is nearly finished rescuing them when he crashes and is hurt. The player volunteers to rescue Marty. Because he is hurt (you can see his arm in a sling and his bruises), Marty can't get out and build the things needed to fix his ship. So he asks the player to make things for him. The player goes nearby where the components she needs can be found. She has to figure out how to use and combine them. When stuck or confused, the player can come back to Marty, who provides hints or advice about how to proceed. If a player is really stuck on a particular problem, then Marty gives her detailed instructions so she can proceed to the next puzzle. Note that getting advice or hints from Marty fits the narrative structure since Marty knows what to do but is too badly injured to do it himself.

In order to fix Marty's ship, the first job is to fix the ship's computer. The computer needs numbers and letters to work. The goal of the first level is to generate the numbers needed. The culmination of the level is the construction of a program that computes powers of two (1, 2, 4, 8, and so on to 2 to the thirtieth power). The next level involves the construction of a program that computes the alphabet. The task after that is to fix the ship's clock. Solving these puzzles involves measuring time, mathematics, and some new programming techniques. At one point the player has constructed a number that shows how old she is in seconds. And the number changes every second!

It is instructive to look at some puzzles in detail. The first real program a player builds is in the ninth puzzle. Marty needs a number greater than one billion for the computer. The player needs to train a robot to repeatedly double a number. Several of the earlier puzzles prepare the player for this task:

1. The first three puzzles introduce numbers, addition, and boxes (data structures).
2. In the fourth puzzle Marty needs a number greater than 1,000 (see Figure 5). When the player goes next door on the floor is just the number 1 and a magic wand that copies things (see Figure 6). The trick to this puzzle is to repeatedly copy the number and add it to itself, thereby doubling it each time (see Figure 7). In addition, the magic wand has a counter that is initially set to 10. After ten copies it has run out

of magic and won't work any more. This helps constrain the search for a solution. The solution requires the player to repeat the same action 10 times.

3. In the eighth puzzle, the player is introduced to robots and builds her first program. This puzzle is very simple. Marty needs a box with two zeros in it. When the player goes next door she sees a robot with a magic wand and a box with one zero in it (see Figure 8). The wand is stuck to the robot and can't be used to copy the box. Most players discover that you can give the box to the robot (and those that don't, do so soon after getting hints from Marty). The player trains the robot to copy the box and drop the copy. Giving the robot the box activates him. He repeats what he was trained to do and copies the box.



Figure 5 – The injured alien introducing the fourth puzzle

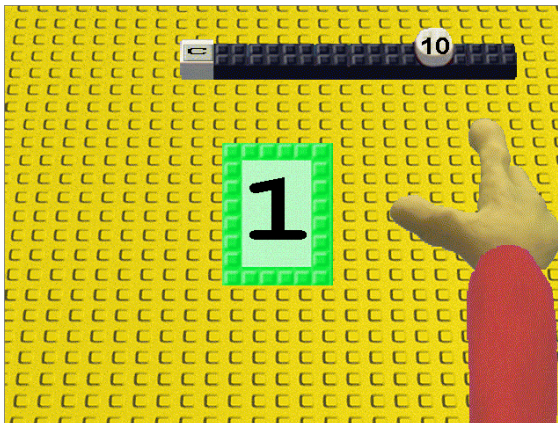


Figure 6 – The initial state of the fourth puzzle

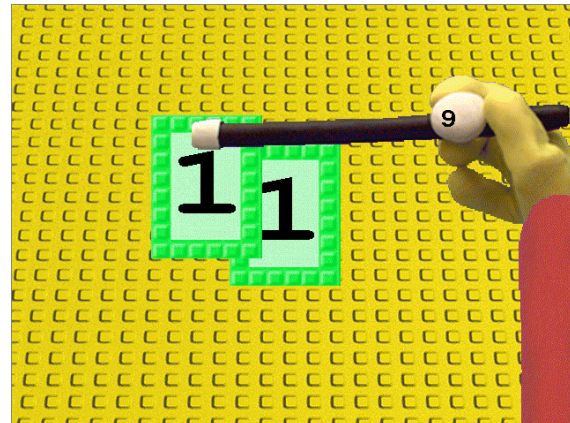


Figure 7 – Using the Magic Wand to copy a number during the fourth puzzle

These early puzzles are designed to simplify some programming tasks. For example, the player doesn't need to know how to terminate the training of a robot. When the limit on the number of steps the robot remembers is exceeded, his training is automatically terminated. Similarly, the counter on the magic wand ensures that the robot will stop after

the correct number of iterations. In later puzzles, arranging for robots to stop when a task is completed becomes the player's responsibility.

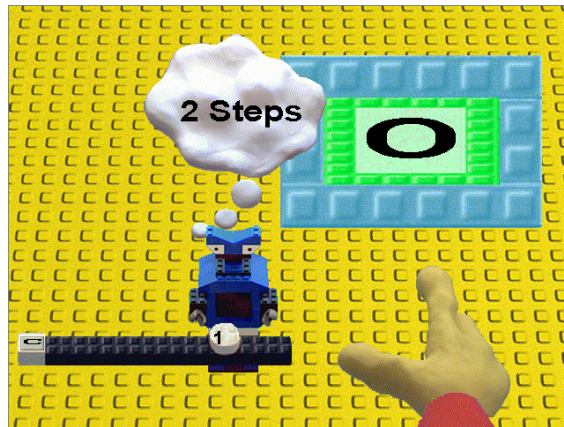


Figure 8 – The initial state of the eighth puzzle

By the time the player starts the ninth puzzle, she has performed the prerequisite actions and must combine them properly to train a robot to repeatedly double a number. The player is presented with a robot holding a wand good for 30 copies and a box with a 1 in it. Because the player has only the robot and the box to work with, and because of the limitations imposed on the robot, this otherwise overly ambitious early programming example can be solved by most players with few or no hints. And yet the constraints do not make the puzzle trivial: experimentation, thinking, and problem solving are necessary to solve the puzzle.

A good series of puzzles leads a player step by step where the puzzle designer wants to go. The players don't feel as if they are being led anywhere but have the illusion that they are in control. The puzzles constrain the set of objects that can be used and how they can be used so that the player has only a few choices. If designed well, the puzzle sequence can be challenging without being frustrating.

Even among those children for whom the puzzle game is well suited, there is variation from those that want to figure out everything themselves to those who very quickly want hints. In Toontalk, if you come to Marty empty handed or with the wrong thing, he will give you a hint. Each time you return during the same puzzle you get a more revealing hint until eventually you get detailed instructions from Marty on how the puzzle should be solved. This behavior accommodates a wide range of learning styles from independent problem solving to following directions.

A good puzzle sequence has a "self-testing" character. ToonTalk puzzle number 15, for example, is a difficult programming task for novices — generating a data structure containing 1, 2, 4, 8, and so on up to 1,073,741,824. The prerequisite knowledge for constructing such a program was acquired in solving puzzle 9 (constructing a program to compute 2 to the 30th power) and puzzle 13 (constructing a data structure filled with zeros). These puzzles in turn rely upon having learned in earlier puzzles how to double a number and how to train robots (i.e., construct programs). The fact that the children

succeeded in solving the puzzles indicates that the puzzles have succeeded and that the children are learning ToonTalk and computer programming.

This kind of tutorial puzzle sequence is strictly linear. A less linear game based upon the idea of a treasure hunt or an adventure game should also be considered. The player would explore and find puzzles to solve. The game designer could still maintain some control by making certain areas open only to those who have succeeded in solving some prerequisite puzzles.

Pictorial instructions

Children can often be seen building a toy or a kit by following instructions that consist of a series of pictures. Many children, for example, enjoy building LEGO® constructions by following pictorial instructions. They learn design and construction techniques in the process, as evidenced by their own subsequent creations. Might not this technique work for children's software as well?

To explore this question, a sequence of about 60 screen snapshots was generated for building an exploding object in ToonTalk. This involves using collision detectors, sound effects, and a change in an object's appearance. While children were able to follow the instructions, we learned that generating good instructions requires good graphic design, lots of testing and revision, and a good understanding of the required prerequisite knowledge and experience. In particular, we found the following:

1. Pictures should show what is necessary and nothing more. Screen snapshots are a poor substitute for a good drawing because there are too many irrelevant details in each snapshot that make it hard to find the important parts of a picture.
2. The step size should be just right. Too big or too small a transition between successive pictures confuses the children.
3. Instructions should be appropriate for the level of experience of the child. The exploding-object instructions were too hard for children with just an hour or so of experience with ToonTalk.

You might question the focus on *pictorial* instructions — what about textual instructions? Textual instructions for building things in ToonTalk tend to be awkward and hard to understand. The world of ToonTalk is so visual that text without accompanying illustrations or animations is not very effective. Consider how hard it is to explain to someone how to tie a knot over the phone. Nonetheless, a few children have been observed to repeatedly get hints from Marty to solve a puzzle until they receive full textual instructions from Marty, and only then, do they try to solve the puzzle.

Viewing of demos

Instructional films and educational TV are generally accepted as effective for some kinds of learning. Why not apply them to the task of learning to program inside of ToonTalk?

ToonTalk includes eight different automated demonstrations. They are simply a replay of someone using ToonTalk, accompanied by narration and subtitles. Most of the demos are not different from watching someone give a demo to an audience. They tend to highlight different features or techniques. As with TV, there is no opportunity for the student to ask questions.

Two of the demos are unusual. One is scripted like an introductory tour. The viewer imagines she is on a guided tour of the ToonTalk world. The tour guide welcomes the visitor and greets characters in the ToonTalk world and proceeds to show how the basic objects and tools in ToonTalk work. The other demo has a soundtrack of two children trying to build a Ping Pong game in ToonTalk. One of the children, Nicky, is a novice; the other, Sally, has a fair amount of experience but still finds building a Ping Pong game challenging. Nicky frequently asks questions, and consequently explanations of what is happening are given in a natural context. Most importantly, the children frequently make mistakes. This demo shows the *process* of building something in ToonTalk — including how to deal with bugs and mistakes. This demo illustrates the process of building small pieces, testing them, tracking down and fixing bugs, and then integrating the pieces.

It is important to pay attention to production values when making software demos. Children will, quite naturally, compare them to TV shows. If the narration, script, or voice acting is amateurish, for example, the demos will not be as appealing.

Another important thing that these demos attempt to communicate is good programming style in ToonTalk. By watching a demo of an expert building something, an observant student will notice not just the necessary actions, but all the other actions that constitute the style of the expert.

Social learning

Absent from this discussion of ways of learning are the traditional ones like listening to lectures by teachers, asking questions, or doing homework assignments. These techniques can work quite well, especially when the teacher is knowledgeable. In such cases, the techniques described above can augment the activities of the teacher. Unfortunately, not all teachers are good at teaching complex subjects like computer programming [Yoder 94]. And computer programming is something interested children may wish to learn on their own. My hope is that a child can learn on her own with software that supports exploratory learning, problem solving, detailed instructions, and demonstrations.

Also absent from this discussion is learning in a *social* context. Children frequently play or study in pairs or teams and they help and teach each other in the process. How can we design software to facilitate this kind of group activity? The software should do the following:

1. *Work with a long viewing distance.* Most software is designed to work for a user who is 12 to 18 inches from the display. When 2 or 3 children work together, the distance usually increases. ToonTalk, like most video games, was designed to work in a typical living room, where the display may be 4 to 10 feet from the player: text and objects are large.
2. *Support multiple players.* Nearly all children's software is designed to work with a single child using the mouse, keyboard, and possibly a joystick. In contrast, many video games today support 2 to 4 simultaneous players, each with their own joystick or game pad. ToonTalk could be enhanced to support multiple users, each with their own controller and screen persona.
3. *Support networked collaboration.* For software to support children playing or learning together over a network, it must deal with many technical issues such as voice communication, latency, and reliability, as well as social issues such as privacy, inappropriate behavior, and trust. ToonTalk supports networked collaboration by the use of “long-distance” birds that can fly to nests on other computers running ToonTalk.
4. *Support an on-line community.* Web sites, email, chat, and discussion groups all can contribute to a support network to help children master complex subjects like computer programming.

Back to the Future

I have argued that computer-based learning environments should be universal in the broad sense of the word. Not only should they be capable of expressing any computation but the expression should be well-suited to the cognitive abilities of most students. Conventional programming languages fail here by being too abstract for most students. Furthermore, programs should be capable of using more than the purely computational capabilities of computers but their input and output devices as well. A Turing Machine cannot show nice graphics or play sound effects.

While I believe that ToonTalk meets these goals better than other environments, there remain many areas of improvement. While ToonTalk provides support for 2D graphics, sound effects, force feedback effects, multiple input devices, networked computers, and web page access, it provides no support for 3D graphics, audio input, and video input and display. Support is limited for generating music and intercommunicating with other programs. ToonTalk supports long-distance synchronous communication and collaboration via birds that can fly to nests on other computers, but has no support for asynchronous communication. There are plans to address these shortcomings in the near future (except for 3D graphics).

Like many other learning environments, ToonTalk shares the goals of supporting communities of learners, collaborative design and problem solving, and enabling learners to explore and create. Like many other environments, ToonTalk attempts to

accommodate different learning styles. It attempts to do all this in a playful manner that appeals to children.

Like few other learning environments, however, it also strives to make the full general power of computers available to not just the authors of learning materials but also to the learners themselves. The children can be not just consumers of educational software but producers as well. Unlike others that share this goal, ToonTalk is significantly easier for students to master. By making computation concrete, while keeping its generality, ToonTalk gives learners creative access to the full power of those magical devices we call computers.

References

[Boxer 02] www.soe.berkeley.edu/~boxer/bibliography.html

[Kahn 96] Ken Kahn, "ToonTalk - An Animated Programming Environment for Children", *Journal of Visual Languages and Computing*, June 1996.

[Kahn 02] Ken Kahn, ToonTalk Web Site, www.toontalk.com

[Kay 81] Alan Kay et. al., *Byte Magazine*, Vol. 6, No. 8, Smalltalk issue, August 1981.

[Papert 80] Seymour Papert, *Mindstorms: Children, Computers, and Powerful Ideas*, New York, Basic Books. 1980.

[Papert 93] Seymour Papert, *The Children's Machine: Rethinking School in the Age of the Computer*. New York. Basic Books. 1993.

[Playground 01] Playground Web Site, www.ioe.ac.uk/playground

[Resnick 96] Mitchel Resnick and Brian Silverman, "Active Essays"
<http://el.www.media.mit.edu/groups/el/projects/emergence/active-essay.html>

[Resnick 1997] Mitchel Resnick, *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*, MIT Press, Cambridge, MA, 1997.

[Squeak 02] Squeak Web Site, www.squeak.org

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