Helping Children to Learn Hard Things:

Computer Programming with Familiar Objects and Actions

Ken Kahn, Animated Programs (KenKahn@ToonTalk.com)

Reprinted from the book *The Design of Children's Technology*, edited by Allison Druin.

Introduction.

Some children, when introduced to something new and complex, will jump in and explore because they enjoy exploration and are good at it. Others are much more timid and will explore only if coached or guided. Others ask for instructions and follow them meticulously. Some children will carefully watch a demonstration, while others are impatient to try things themselves.

It is possible that there is a style of learning that dominates others in effectiveness and appeal. The position taken here, however, is that children differ, and that all these learning styles have their place. Even an individual child may switch styles as circumstances and experiences change.

Given the wide variety of ways that children learn, how should we design software for children? This chapter attempts to answer this question by looking closely at our experiences with the design and testing of ToonTalk™.

ToonTalk (Kahn 1996, 1998) is an animated world where children build and run programs by performing actions upon concrete objects. The child builds real computer programs by doing things like giving messages to birds, training robots to work on boxes, loading up trucks, and using animated tools to copy, remove, and stretch items.

ToonTalk has been tested with 4th grade classes for the last 3 years. Initially, it supported only an exploratory learning style. Some children quickly began exploring and tried to figure out what each item does and how to combine them. Most, however, asked for guidance. This led to the development of enhancements of ToonTalk that cater to children with different learning styles. ToonTalk now includes a puzzle game that plays the role of a tutorial. This appeals most strongly to children who tend to like to solve problems but are less comfortable exploring on their own. An animated character named Marty was added to ToonTalk that acts like a coach or guide in ToonTalk. Some children like to hear suggestions from him and follow them. Others quickly send Marty away because they find him annoying. A set of narrated demos of ToonTalk was created for those who like to sit and passively be shown how to make things. Illustrated instructions on how to build some programs were produced. These too appeal to a subset of the children.

Children of different ages, experiences, and learning styles approach the same software in very different ways. The main lesson we can take away from this experience is that children differ in the degree to which they are motivated and effective at exploring (on their own or with an animated guide), or following instructions, or solving a puzzle sequence. Even the same child will prefer different styles of interaction depending upon her prior experience with the software. Ideally, a software program should be designed so that a wide variety of children might enjoy and benefit from it.

A brief introduction to ToonTalk.

Efforts to give children the opportunity to do real computer programming began over thirty years ago. The *Basic* programming language began as an ordinary

programming language with as much as possible removed (Kemeny, J. & Kurtz, T. 1985). (Over the years, procedures, recursion, data structures, and so on have been "put back" into it.) *Logo* and *SmallTalk,* in contrast, were designed to give children the best "state of the art" programming tools as possible (Papert, S. 1980 and Kay, A. et. al. 1981).

After thirty years of mixed results, many educators today question the value of teaching programming to children. It is hard and there are now so many other things children can do with computers. Proponents of programming argue that programming can provide a very fertile ground for discovering and mastering powerful ideas and thinking skills (Papert, S.1980). Furthermore, programming can be a very empowering and creative experience. Children who can program can turn computers into electronic games, simulators, art or music generators, databases, animations, robot controllers, and the literally millions of other things that professional programmers have turned computers into.

Why do we rarely see these wonderful results from teaching children to program computers? The answer seems to be that programming is hard — hard to learn and hard to do. ToonTalk started with the idea that maybe 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, the idea is that real, advanced programming can be done from inside a virtual animated interactive world.

The ToonTalk world resembles a twentieth-century 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. So 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. After landing she controls an on-screen persona. The persona is followed by a dog-like toolbox full of useful things.

An entire ToonTalk computation is a city. Most of the action in ToonTalk takes places 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 its "thought bubble" and showing it what to do. The robot remembers the actions in a manner that can easily be abstracted to apply in other contexts.

A robot behaves exactly as the programmer trained it. This training corresponds in computer science terms to defining the body of a method in an object-oriented programming language like 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

- terminate a process by setting off a bomb
- change a data structure by taking items out of a box and dropping in new ones

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 1.) 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.

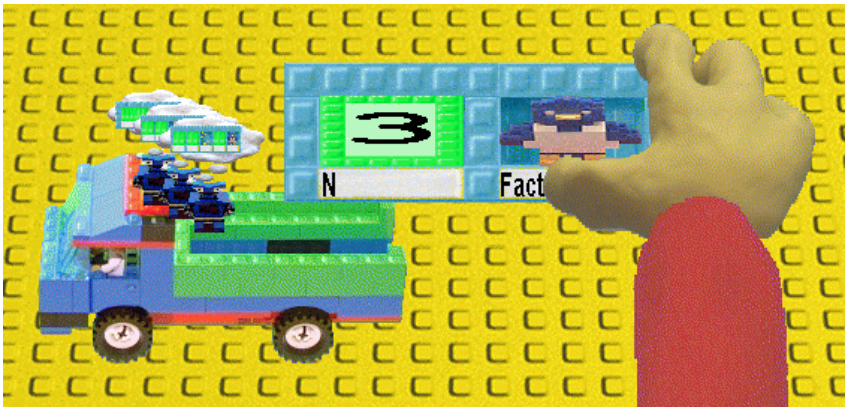

Figure 1 — A truck being loaded.

How children learn to master ToonTalk.

ToonTalk provides four fundamentally different ways for children to learn:

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.

Safe self-revealing environments.

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 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 2.) 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 *Mario Brothers*® or Sega's *Sonic the Hedgehog*® games 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 in this level 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 2). When the player goes next door on the floor is just the number 1 and a magic wand that copies things (see Figure 3). The trick to this puzzle is to repeatedly copy the number and add it to itself, thereby doubling it each time

(see Figure 4). 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 5). 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 2 – The injured alien introducing the fourth puzzle

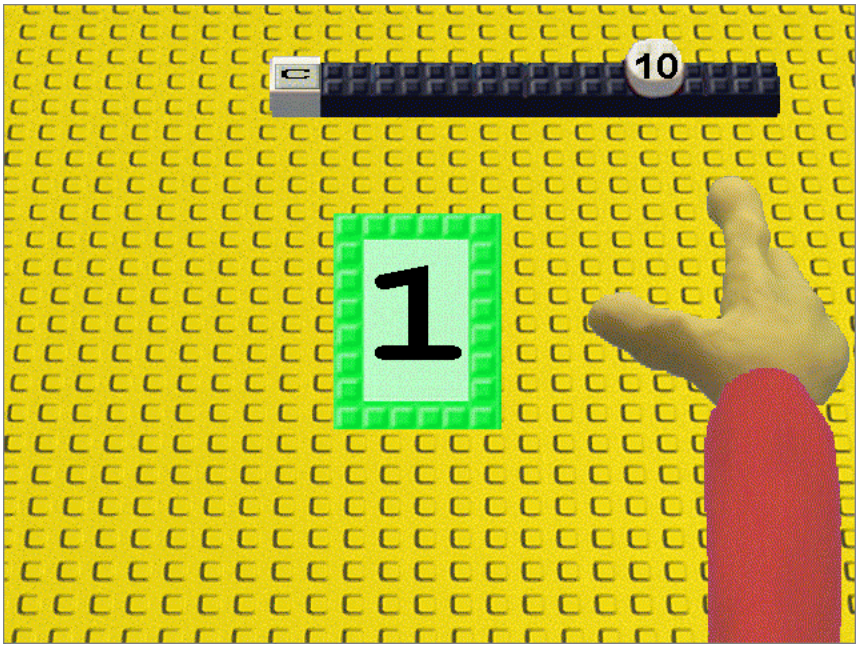

Figure 3 – The initial state of the fourth puzzle

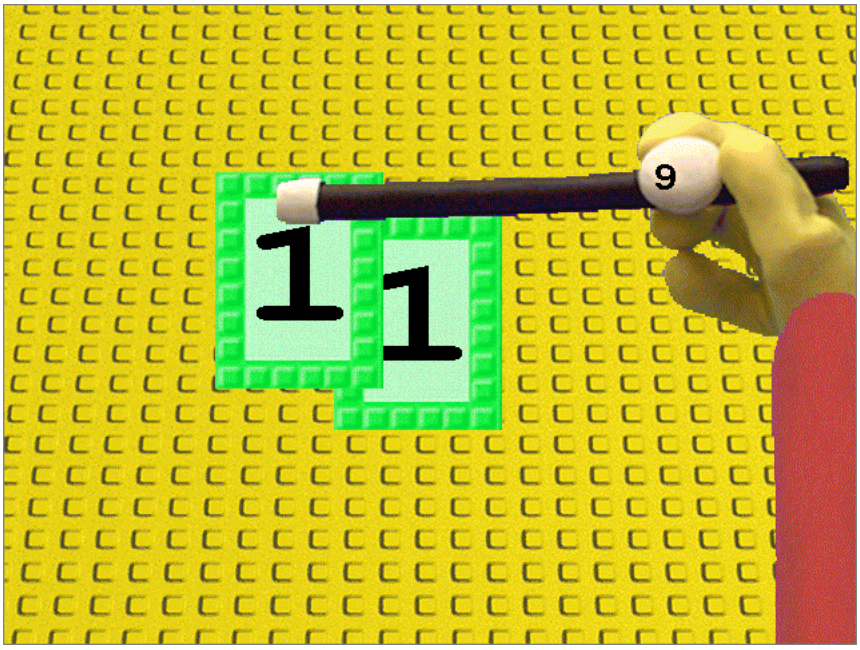

Figure 4 – Using the Magic Wand to copy a number during the fourth puzzle

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.

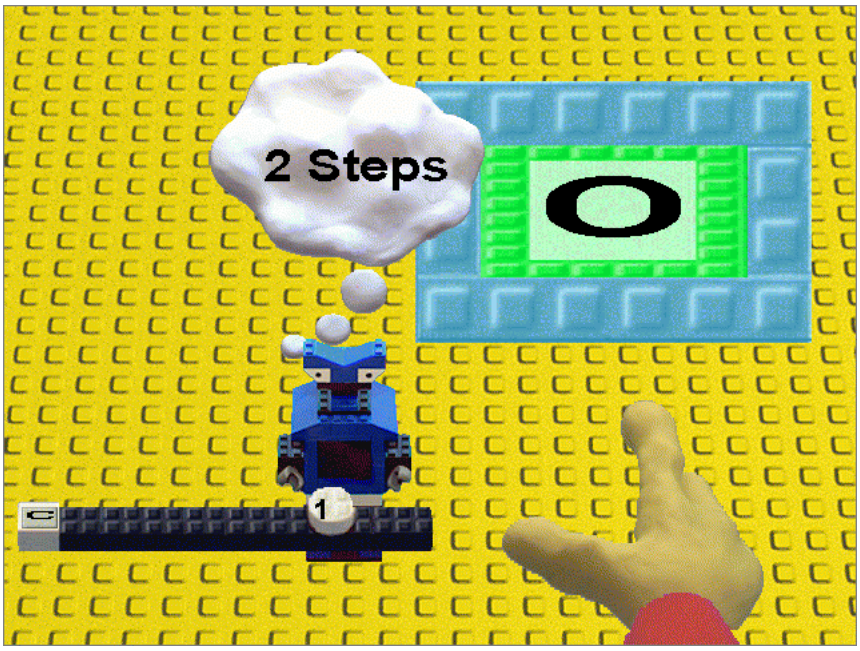

Figure 5 – The initial state of the eighth puzzle

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:

- 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.
- The step size should be just right. Too big or too small a transition between successive pictures confuses the children.
- 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.

We plan to generate and test new sets of instructions, taking into account the above lessons.

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.

Other ways to learn.

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, 1994). And computer programming is something interested children may wish to learn on their own. The 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, most 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.
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.

Dimensions of learning techniques.

One can analyze the techniques described here along various dimensions:

1. *Active-Passive.* The techniques differ along a spectrum from passively watching or listening to actively exploring. Figure 6 places the techniques discussed here on such a scale. Note that there is an analogous spectrum along the cognitive dimension. A child watching a demo may be thinking very hard about what she is watching. Or a child engaging in free exploration may be "randomly" clicking on things without much thinking. It is beyond the scope of this chapter to attempt to place activities on this cognitive active-passive spectrum.
2. *Creativity.* The techniques also differ in the extent to which the child is instructed versus creatively exploring or constructing. Figure 7 illustrates this. Notice that pictorial instructions require lots of activity but very little creativity.
3. *Planning-Tinkering.* (Turkle & Papert 1990) discuss these two learning styles in the computer culture. Some children prefer to understand and plan before attempting to build things while others like to intertwine planning, learning, building, testing, and revising.

There are other spectrums along which these techniques differ. Appeal and effectiveness are two dimensions that are specific to the individual child. Some children like watching demos, and some don't. Some seem to learn best by free exploration; others don't.

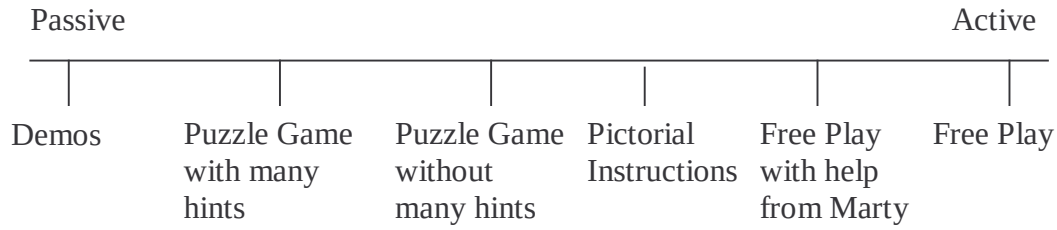

Figure 6 -- The Active-Passive Dimension

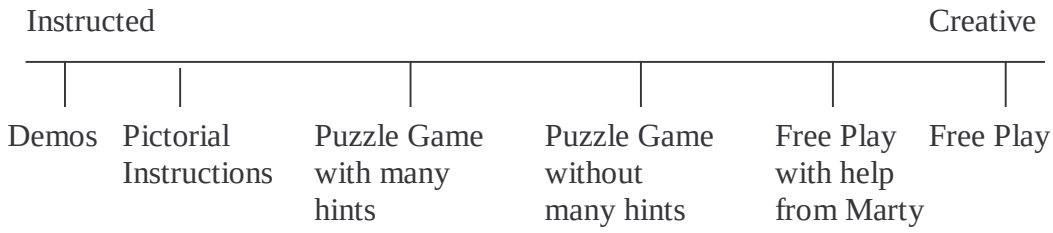

Figure 7 -- The Creative Dimension

Testing.

ToonTalk has been tested with fourth grade classes in Menlo Park, California from January 1995 to the present (June 1998). Pairs of children were observed using ToonTalk for about three 40-minute sessions apiece. For the first year and a half of testing, only free play was available, and most of the children continually sought

guidance from an adult. After the puzzle game was implemented, a new class of children tested it exclusively. No formal testing was performed but nearly all the pairs of children solved the first 25 puzzles without assistance. The children had no prior exposure to ToonTalk and only two had any prior exposure to computer programming.

Beginning in September 1997, the fourth graders were given the freedom to choose and switch between free play, the puzzle game, and demos. (The pictorial instructions were not ready for additional testing.) The informal observations of these 24 children confirmed the thesis of this chapter. The children tended to try the different modes and switch between them. Preferences varied as to which activities they would do.

ToonTalk has also been tested in hundreds of homes. The resulting anecdotal evidence is that most testers tried all three modes and learned different things from each. Preferences also varied.

Kids as Critics and Designers.

The design of ToonTalk and its learning aids was heavily influenced by over three years of use and feedback by children. Simply watching children use the software led to two major redesigns and hundreds of smaller improvements. For example, the initial design provided three kinds of magic wands — for copying, for changing sizes, and for removing things. Observing frequent confusion and frustration led to replacing the wand for changing sizes with a bicycle pump and to replacing the wand for removing things with a hand-held vacuum.

Children, unlike adults, have rarely provided criticism. They are, however, often eager to provide all sorts of creative suggestions. For example, in ToonTalk, you can only land your helicopter on the street. Many children have suggested that you should be able

to land on the roofs of houses. When asked what would happen then, responses varied, from the house blows up to you climb down the chimney. Others suggested that you should be able to walk into the water that surrounds the island. Some wanted to add boats, drowning, and sharks.

Very few suggestions from children for changes to ToonTalk have actually been followed. This is because every item or action in ToonTalk supports the fundamental purpose of the system — enabling its users to create programs. Much effort went into making them fun and appealing as well. ToonTalk has a magic wand because there is a need for copying things when programming. Many children find it fun to *play* with the wand, but it also serves an essential function. Blowing up houses by landing on them or drowning, not only do not serve useful functions, but they interfere with the task of programming.

Some suggestions are very good ones -- for example, that there should be more than three styles of houses or there should be more than one room per house. These would be both functional and add appeal. They haven't been implemented however, because of resource limitations and the large number of higher-priority changes.

My son, David, who was 7 years old when ToonTalk was started over 5 years ago, has been an invaluable source of ideas. He has also played a very important role as a *sounding board*. As I considered design changes and alternatives, I would frequently explain them to him first. In contrast to feedback from focus groups, David had years of experience with ToonTalk but still had a child's perspective.

Related Work.

Rocky's Boots and *Robot Odyssey,* two games from The Learning Company in the early 1980s, excited many computer scientists. In these games, you can build arbitrary logic circuits and use them to program robots in the context of an adventure game. The user persona in the game can explore a city with robot helpers. Frequently, in order to proceed, the user must build a logic circuit for the robots to solve the current problem. The design of ToonTalk and its puzzle game were inspired by *Robot Odyssey*. The most important difference is that ToonTalk is capable of supporting arbitrary user computations – not just the Boolean computations (AND, OR, and NOT) of *Robot Odyssey*.

Many computer and video games use puzzles as an effective and fun tutorial. *Lemmings*® and *The Incredible Machine*® are two good examples. Scott Kim (Kim, 1995) has written about puzzle design and the pedagogic role of puzzles.

Pictorial instructions are very commonly used in building a model airplane, a LEGO® set, or furniture. We are not aware of any attempts to use them for other tasks such as learning to program a computer. In the context of textual programming, it is unclear what the instructions could do other then tell the student what needs to be typed.

Safe, self-revealing, incremental exploratory environments are very common in computer and video games. They are not common in textual programming environments like Logo or Basic. Someone exploring Logo, for example, needs to be told that *FORWARD N* will move a turtle forward n units — she is unlikely to discover it by exploration.

In the last 15 years there have been several attempts to build intelligent tutors that can teach programming or math (e.g. Selker (1994) and Corbett & Anderson (1992)). They give explanations, propose problems, and, most importantly, can give intelligent feedback when students fail to solve problems. These tutors are much more sophisticated than Marty, the ToonTalk guide. A tutor based upon this research could be built to help children in free play, solving puzzles, or even following pictorial instructions. Such a tutor could analyze a student's behavior and give much more appropriate and intelligent advice than Marty is capable of.

Conclusion.

We have presented self-revealing exploratory environments, puzzle sequences, pictorial instructions, and demos as aids for students to learn hard things. Currently the evidence for their effectiveness and appeal is informal and anecdotal. We hope to find and collaborate with another group interested in studying the effectiveness of these techniques and their combinations in a more formal manner.

It seems likely that these techniques could be combined effectively to learn to use complex software like Windows 98® or Adobe PhotoShop®. Perhaps software can be made that uses combinations of these techniques to teach science, engineering, or math.

The lesson that I hope you take away from this is that video games and toys are a rich source of ideas and techniques for introducing rich and complex things to children in a fun and effective manner.

ToonTalk availability.

ToonTalk is currently available for beta testing. It has been published in The United Kingdom and Sweden and soon will be available in other countries. Visit www.toontalk.com for more details.

Acknowledgements.

I wish to thank Mary Dalrymple for her comments on early versions of this chapter. Special thanks go to Ruth Colton, who permitted me to test ToonTalk with her 4th grade students. And I am very grateful to David Kahn and all of the other ToonTalk beta testers.

References.

Corbett, A. T. & Anderson, J. R. (1992). The LISP intelligent tutoring system: Research in skill acquisition. In J. Larkin, R. Chabay, C. Scheftic (Eds.), *Computer Assisted Instruction and Intelligent Tutoring Systems: Establishing Communication and Collaboration.* Hillsdale, NJ: Erlbaum

Kahn, K. (1996, June). ToonTalk™ — An Animated Programming Environment for Children. *Journal of Visual Languages and Computing, Vol 7, pp 197-217.* (An abbreviated version appeared in *Proceedings of the National Educational Computing Conference*. Baltimore, Maryland, June 1995.)

Kahn, K. (1998). *ToonTalk Home Page*. [Online]. Available at http://www.toontalk.com.

Kay, A. et. al. (1981, August) Byte Magazine, Vol. 6, No. 8, Smalltalk issue.

Kemeny, J. & Kurtz, T. (1985) *Back to BASIC*; Addison-Wesley Publishing Company; Reading, MA

Kim, S. (1995, April). Puzzle Games and How to Design Them. *Proceedings of the Ninth Annual Computer Game Developers' Conference.* Santa Clara

Papert, S. (1980) *Mindstorms: Children, Computers, and Powerful Ideas*, New York, Basic Books.

Papert, S. (1993). *The Children's Machine: Rethinking School in the Age of the Computer*. New York. Basic Books.

Selker, T. (1994, July). COACH: A Teaching Agent that Learns. *Communications of the ACM* 37, 7, pp. 92-99.

Turkle, S. & Papert, S. (1990) Epistemological Pluralism: Styles and Voices within the Computer Culture. *Signs* 16, no. 1, pp 128-157

Yoder, S. (1994). Discouraged? ... Don't Dispair [sic]. *Logo Exchange* 12, 2.