

Running Head: TOONTALK™ PROGRAMMING FOR KIDS

Problem Solving in a Digital Playground: ToonTalk™ Programming for Kids

William Warrick

George Mason University

With the advent of the first computers in schools in the 1980's came concerns over what to do with them. Never before had education seen such a case of putting the cart before the horse as was the case with the proliferation of computers without purpose. Generally, the public, school administrators, and teachers all somehow felt as though computers *should* be in the schools, yet didn't know why or how they should be used.

One of the first uses of the computers was to teach programming. Whether because of the dearth of available programs available, or due to the fact that the teachers charged with using the computers tended to be technically inclined, classes in BASIC programming proliferated. The argument/rationale for teaching a programming language was that it taught the students organization and logical thinking.

Once the education market was saturated with computers, software publishers geared up and flooded schools with ready-made software for students to use. The days of teaching programming appeared to be numbered.

ToonTalk represents a return to looking at programming as a means to provide students with tools for organizing and problem solving. The purpose of this paper is to investigate the program and, more importantly, it's application to the needs of today's students. Attention will be paid to situated uses of ToonTalk by researchers in a number of different areas.

The inventor of ToonTalk, Ken Kahn, states that his goal in creating his computer system was to design something that kids could use to build programs without being taught how to use it (1996). His interest in programming was sparked by the idea that programming for kids is an empowering activity but only accessible to those who can master programming languages - often arcane and esoteric. Learning a programming language was thus out of reach for most students. Kahn borrowed from precedents he discovered in the way that children learn and discover on their own with games and toys. LEGO systems seem to have provided the most concrete

example of how children, without prior instruction or learning, can design, construct, and use LEGO bricks. LEGO blocks, invented in 1958, are simple interlocking blocks which can be combined into complex structures in an infinite number of ways (Pesce, 2000).

Among the difficulties faced by Kahn in developing a programming language or system for students was the fact that programming languages were written at such a level as to be too complex and inaccessible to casual programmers or students without formalized training (Kahn, 1996). Toontalk departs from this paradigm of programming in that it is visual. In fact, ToonTalk goes beyond the visual metaphors and presents the programming tools in an animated fashion. Kahn states, "... the source code is animated and the programming environment is a video game. Every abstract computational aspect is mapped into a concrete metaphor" (1996, p. 197). The metaphor based language allows children to interact with the program in such a way as to relieve them of the odious task of learning programming code and commands. Commands are represented by familiar and intuitive metaphors. An animated vacuum cleaner, for example, is used to erase objects. Hammers are used by robots to 'join' or add values to one another. Norman speaks of the importance of representation in defining the ease with which we accomplish tasks. The form of representation makes a dramatic difference in the ease of the task, even though the choice [of representation] does not change the task. He goes on to say that where bad representations can turn problems into reflective challenges, good representations can often transform the same problems into easy experiential tasks (1994).

Kahn continues by saying that ToonTalk is based upon concurrent constraint languages such as Janus or Linear Janus in that it contains a consistent set of concretizations that convey every language construct. Concretizations in this sense indicates the desired ability to metaphorically describe actions and constructs within the program itself. Concretizations are mappings between programming language abstractions and tangible objects (Kahn, 1999).

Animation is the key element to ToonTalk and it is that which sets it apart from other languages. It has been described earlier in this paper that the greatest obstacle in making programming accessible to students is the fact that while they are elegant and expressive, they are difficult to learn and master (Kahn, 1996). The execution of ToonTalk programs is animated. The act of expressing a program in ToonTalk is based on a video-game metaphor (Cockburn and Bryant, 1997). ToonTalk goes beyond many ‘visual’ programming languages and animates both the source code and the program itself. Kahn takes the environment inherent in video games and adapts the ability to solve problems and create worlds to his program. Johnson (2001) writes that while complex concepts seem counterintuitive or abstract, programs can make them come to life with graphics suited to the Nintendo generation. ToonTalk succeeds in that the world in which the student works is familiar. It contains 20th century artifacts such as houses, cars, tools, helicopters, among other things. Essentially constructivist in nature, Kahn has provided students with an environment in which students can use familiar objects to create new tools for their tasks. Knowledge about the ToonTalk world and, by extension, about the world in which the child exists is created by the student.

The affordances of a visual, animated children’s programming tool are apparent. The world created in ToonTalk is accessible and motivating to students and allows for exploration easy understanding of the concepts. The question then becomes, “Why?”. With the glut of software on the market that is preprogrammed to solve problems, assist in the access and manipulation of data, and introduce and reinforce learning concepts, what purpose is served by developing such a programming language? Better yet, what need does this program address?

Some authors speak of the necessity to learn to program. In a paper discussing their contribution to programming, Cockburn and Bryant (1997) speak of the need for school children to be introduced to programming at an early age simply because of the proliferation of computers

and technology.

From the beginnings of the technology infusion in schools, programming computers was at the top of the list of uses for the computer. In part because there were not many programs available to run on the computer, certainly not many that were appropriate for educational use, but also because computer programming was seen as a way to help students develop organizational, problem solving, and critical thinking skills. BASIC programming was, for the most part, the language of choice in many schools because it was higher level and provided the most syntactically familiar framework for students to master. No doubt a small part of the emphasis on programming for all students was to provide future programmers to an increasingly technical society.

For his part, Kahn talks about computers as tools that can be adapted to different tasks depending on the needs of the users (Morgado, Cruz, & Kahn, 2001). This view of technology's place in schools is shared by others as well. Seymour Papert was among the first to conceive of a programming language designed for use by children. His program, LOGO, however, was designed to be a tool, albeit a programable tool, with which children could learn mathematics (Terrapin Software, 1997) Research into the applications of LOGO has been done by many, with the focus on the program's ability to assist students with learning about learning. The students using Logo were learning about knowledge itself.

Extending Papert's work in LOGO, the creators of ToonTalk see the program filling a niche for a programming language that was visual and did not require the learning of BASIC commands in order to use it. The function of the program for students was to learn to create programs which could then be used to solve other tasks. Students might create a game, a calculator for use in solving mathematical problems, or some other tasks. The end result is that the students have access to deeper levels of the computer as a tool and are then able to

manipulate it to their desired ends (Kahn, 1996).

If the goal of the program is to teach programming - as opposed to its use as a tool to develop thinking skills - then the question becomes, "Why teach programming to students?"

The Playground Project, a collective devoted to developing tools and researching their use with students, answers this way,

There is a long pedigree to the idea that writing a computer program provides a broad canvas on which the learner can sketch half-understood ideas, and assemble on the screen a semi-concrete image of the structures he or she is building intellectually. Early research produced some robust indications that the expression of mathematical ideas in the form of computer programs suggested a promising line of enquiry. (2002)

The instructional benefits of ToonTalk, then, are twofold. Students are given the means to control a tool which has been, and will continue to be, an integral part of their lives. Through the metaphors and animations of their computer programs, students can manipulate the computer to achieve their desired ends. Additionally, ToonTalk can afford students a virtual world in which to experiment with ideas and concepts and use them to gain a deeper understanding of their own thought processes.

ToonTalk is available online in a beta format. The final product is available on CD for purchase. For the purposes of reviewing the program for this paper, the beta program was obtained and run on a PC laptop.

In keeping with the author's conceptions about this being a program that required no instruction at all in order to use it, very little in the way of tutorial or directions is given to the user of the program.

Users of the beta version are given a context in which to work. A town is in danger

of sinking and a friendly Martian, "Marty", lands to assist the citizens of the town. Before all of the citizens can be rescued, however, a mishap with Marty's spaceship leaves him stranded on the sinking island and in need of help. The authentic problem given to the user of the program, then, is to assist "Marty" in repairing his spaceship. To do this, you must complete tasks given to you by "Marty".

The tasks proceed from simple - going to the house next door to the spaceship and retrieving numbers to bring back to the spaceship - to complex, involving the manipulation of formulas and processes. All this is done, of course, in the context of Kahn's *concretization* using tools that are familiar and intuitive. For example, at one point the user is directed to return to the house to retrieve the number "4". In the house are found only two "2's" on the floor. Again, initially without instructions from "Marty", the user soon comes to realize that picking up one of the number "2's" and putting it on top of the other one causes a mouse to come from outside the field of view, hit the stacked numbers with a hammer and thus cause them to be added and morph into the number "4". Without realizing it, the student has graphically created a program that will find the sum of two numbers and discovered a construct for addition. Further exploration of the environment will allow the student to reuse those constructs in designing and creating more complex tasks. Just as programmers rely on subroutines that can be used and reused as parts of larger programs, the users of ToonTalk can use the animated addition routine (of placing one number on top of another) in other, more intricate programs. There are numerous puzzles to be solved in the Puzzle Game area of the program, as they progress from very simple tasks to the most complex. Each game introducing or reinforcing a programming concept. For example, in the first game, "Marty" needs a box with a "1" and a "2" inside. The user is introduced to the notion that boxes are for holding things. The textual equivalent to these

boxes in programming are data structures. The analogous task is to initialize elements of a vector. In the next puzzle, "Marty" asks for a box with an 8, 16, and 32 inside. The user learns to place boxes next to each other in order to make bigger boxes. Obviously, this animated version of the process is far easier to grasp for children than what the underlying process is called: concatenating vectors and operating elements of a data structure! And much more fun as well.

As with so many of the computer games available today, ToonTalk seeks to trick students into learning things where they might not ordinarily wish to do so. Immersed in the video game-like world of ToonTalk, students become part of another world. However, since they have the tools and can learn to manipulate them, the world becomes their own.

If the overt purpose of ToonTalk is to teach programming to students, it might be easy for teachers with limited funds for software purchases to pass it up in favor of more curriculum oriented software packages. Though jobs for programmers still fill the want ads, just as they did 20 years ago when students were being taught BASIC, the current emphasis on Standards of Learning focus teachers' attention elsewhere. Programming is left for elective courses in higher grades.

The covert purpose, or goal, of the program, however, is the development of thinking skills in students. This represents a return to one of the stated aims of programming in the early 80's. Students, by learning to program, acquire and develop such thinking skills as problem decomposition, component composition, explicit representation, abstraction, and, with a bow to Papert, metacognition, or thinking about how they think and solve problems (Animated Programs, 2001).

On either level, the program can succeed. However, Kahn's zeal in working towards a program that requires no intervention on the part of a teacher proves to be a hindrance rather than

a benefit. It is easy to sit back and understand how the student before you is learning to concatenate when ToonTalk hammers two boxes together, it is not at all clear to the student that what he has done constitutes programming. There is no transfer of learning from the game world of ToonTalk to the real world of programming. In fact, research has indicated that while children are able to articulate rules and behavior when ‘connected’ to an object, they may not be able to express it completely in spoken or written language, especially in a decontextualized form (Hoyles, Noss, Adamson, & Lowe, 2001). Perhaps coming to that realization, Kahn’s rhetoric about the benefits of ToonTalk as a programming language have been replaced lately with a greater emphasis on the development of thinking skills. Certainly in this area, Kahn's program does well. Independently, students do solve increasingly complex problems and begin to develop a greater sense of how they think.

Does ToonTalk fill a void in the classroom? Are students shortchanged if they do not have the program available? Probably not. If thinking skills are identified as target learning goals, and standards for students the paramount measuring stick, there are certainly other ways to merge the two and address each in a more economical way. ToonTalk addresses thinking skills, ironically, out of any real context. Kahn contextualizes the process, but the desired learning takes place independent of any relationship to curriculum.

## References

- Animated Programs, Inc. (2001). *Animated programs presents toontalk2*. Press release. Menlo Park, CA.
- Cockburn, A. and Bryant, A. (1997). Leogo: An equal opportunity user interface for programming. *Journal of Visual Languages and Computing*, 8, 601 - 619.
- Digiano, C., Kahn, K., Cypher, A., and Smith, D. C. (2001). Integrating learning supports into the design of visual programming systems. *Journal of Visual Languages and Computing*, 12, 501 - 524.
- Hoyles, C., Noss, R., Adamson, R., and Lowe, S. (2001). Programming rules: What do children understand? *The Playground Project*. [http://www.ioe.ac.uk/playground/frame\\_f.htm](http://www.ioe.ac.uk/playground/frame_f.htm).
- Johnson, S., (2001). *Emergence: The connected lives of ants, brains, cities, and software*. New York: Scribner
- Kahn, K. (1996). ToonTalk™ - An animated programming environment for children. *Journal of Visual Languages and Computing*. 7, 197 - 217
- Kahn, K. (1999). From Prolog and Zelda to ToonTalk. Proceedings of the International Conference on Logic Programming. MIT Press, Danny DeSchreye, Ed.
- Morgado, L., Cruz, M., and Kahn, K. (2001). Working in ToonTalk with 4- and 5-year olds. Paper presented at the Playground International Seminar, Porto, Portugal. April 3, 2001
- Norman, D. (1994). *Things that make us smart: Defending human attributes in the age of the machine*. Reading, MA: Addison-Wesley
- Pesce, M. (2000) *The playful world: How technology is transforming our imagination*. New York: Ballentine Publishing Group
- Terrapin Software, Inc. (1997) Why use logo? An overview of logo in education. <http://www.terrapiinlogo.com/>

