A Computer Game to Teach Programming

Ken Kahn
Kahn@csli.stanford.edu
Animated Programs
49 Fay Avenue, San Carlos, CA 94070
This paper will appear in the Proceedings
of the National Educational Computing Conference 1999

Keywords: learning computer programming, interactive tutorials, animated programming

ABSTRACT

ToonTalk™ is an animated interactive world inside of which one can construct a very large range of computer programs. These programs are not constructed by typing text or arranging icons but by taking actions in this world. Robots can be trained, birds can be given messages to deliver, and so on. ToonTalk has been described at NECC95 [Kah95] as well as [Kah96a and Kah96b].

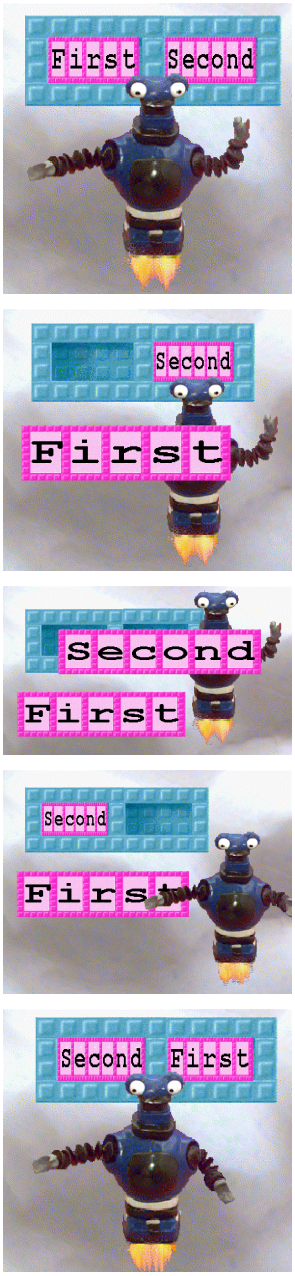
This paper describes the design and preliminary testing of an interactive puzzle game that functions as a ToonTalk tutorial. Children are presented with a series of interactive puzzles in a game-like narrative context. The puzzles gradually introduce programming constructs and techniques. Each puzzle presents the player with a very limited selection of ToonTalk objects. Even some very young children are able to solve the puzzles because the search space is so strongly constrained. And yet players do not behave as if the puzzles are too easy – the children are clearly challenged. The sequence of puzzles is carefully designed to gradually introduce new concepts one at a time. Testing has shown that both children and adults enjoy the puzzles and have learned some sophisticated programming skills.

Introduction to ToonTalk

ToonTalk™ is a programming "language" whose source code is animated. [Kah95] (ToonTalk is so named because one is "talking" in (car)toons.) This does not mean that it is a visual programming language where some static icons have been replaced by animated icons. It means that animation is the means of communicating the entire meaning of a program to both humans and computers. One programs in ToonTalk by directly manipulating objects in a virtual animated world. If, for example, if one needs to swap the values of two locations, what can be more natural and easy than grasping the contents of one, setting it down, grasping the contents of the other, placing it at the first location, and then moving the original item to the second location? (See Figures 2 to 6) This is something a very young child can understand and do, while only a programmer can write the equivalent code. (See Figure 1.)

```
temp = x;
x = y;
y = temp;
```

Figure 1 – Swapping two items in C



Figures 2 to 6
Snapshots of the creation of a ToonTalk program to swap two items.

Computer scientists strive to find good abstractions for computation. In ToonTalk a critical goal is to find good "concretizations" of those abstractions. The challenges are twofold: to provide high-level powerful constructs for expressing all programs and to provide concrete, intuitive, easy-to-learn, systematic game analogs to every construct provided.

The ToonTalk world resembles a twentieth-century city. There are helicopters, trucks, houses, streets, bike pumps, toolboxes, hand-held vacuums, birds, boxes, and robots. An entire ToonTalk computation happens in a city. Most of the action in ToonTalk takes places in houses. Homing pigeon-like birds provide communication between houses. Birds accept 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 into his "thought bubble" and showing him what to do. The robot remembers the actions in a manner that can easily be abstracted to apply in other contexts.

The behavior of a robot is exactly what he was trained to do by the programmer. This training corresponds in traditional terms to defining the body of a method or clause.

- sending a message by giving a box or pad to a bird,
- spawning a new process by dropping a box and a team of robots into a truck (which drives off to build a new house),
- performing 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),
- copying an item by using a magician's wand,
- terminating an agent by setting off a bomb,
- changing a tuple by taking items out of compartments of a box and dropping in new ones.

Figure 7 – Possible actions of ToonTalk robots.

When the user controls the robot to perform these actions, she is acting upon concrete values. This has much in common with keyboard macro programming and programming by example [Smi75]. The hard problem for programming by example systems is how to abstract the example to introduce variables for generality. ToonTalk does no induction or learning. Instead the user explicitly abstracts a program fragment by removing detail from the thought bubble. The preconditions are thus relaxed. The actions in the body are general since they have been recorded with respect to which compartment of the box was acted upon, not what items happened to occupy the box.

The ToonTalk Puzzle Game

A carefully designed sequence of puzzles can be pedagogically very effective. 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 done when he crashes and is hurt. The player has volunteered to rescue Marty. Marty, because he is hurt (you can see his arm in a sling and his bruises), 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 he or she needs can be found and has to figure out how to 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 the player detailed instructions so he or 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 next task 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 he or she is in seconds. And the number changes every second!

A Detailed Look at some Programming Puzzles

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 bigger than 1,000,000,000 for the computer. When the player goes into the room all she finds is a box with the number 1 in it and a robot with a magic wand capable of copying things. The player needs to train the robot to repeatedly double a number. Several of the earlier puzzles prepare the player for this task. The first three puzzles introduce numbers, addition and boxes (data structures). In the fourth puzzle Marty needs a number greater than 1,000 (see Figure 8) and when the player goes next door on the floor is just the number 1 and the magic wand (see Figure 9). There are only two choices: do something with the number or do something with the wand. It turns out that in this context there is nothing that can be done with the number. Marty has already informed the player that pressing the space bar will turn on the wand. Even children as young as 6 years figure out how to pick up the wand, move its tip over the number, hit the space bar, and then drop the copy. If they drop the copy on the number on the floor then the two 1s will be added (in an entertaining way by a little mouse with a big hammer). The trick to this puzzle is to repeatedly copy the number and add it to itself, thereby doubling it each time (see Figure 10). 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. (This is not an easy puzzle for most 9 or 10 years olds. Many, for example, initially make the mistake of making many copies and then adding them.) The solution requires the player to repeat the same action 10 times. Avoiding such tedium is a motivation for programming robots to do the work for you.

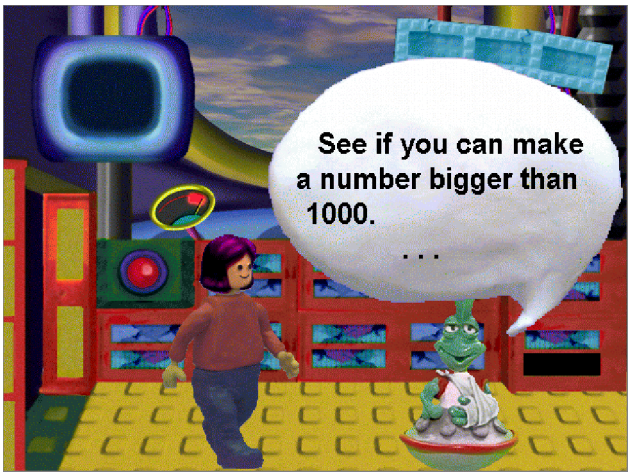


Figure 8 – The injured alien introducing the fourth puzzle

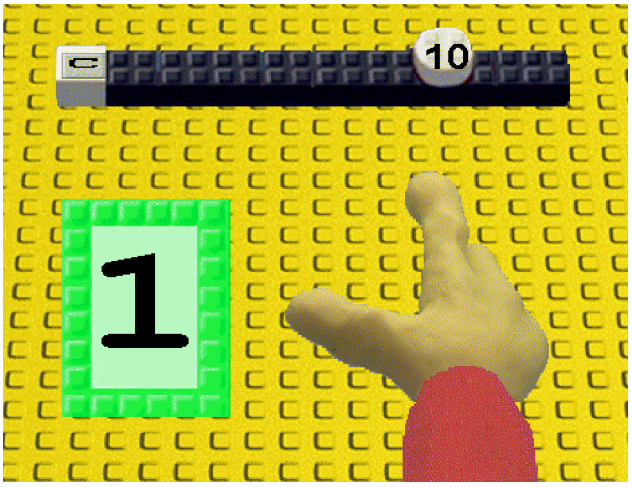


Figure 9 – The initial state of the fourth puzzle

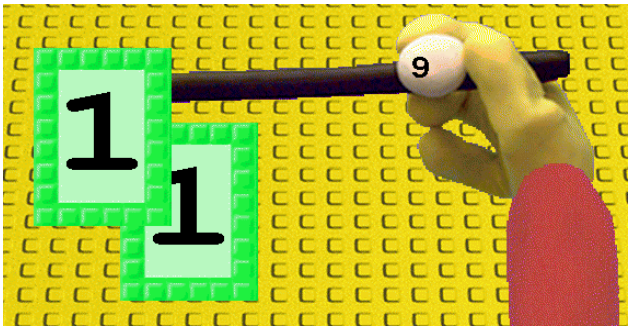


Figure 10 – Using the Magic Wand to copy a number during the fourth puzzle

Just before the ninth 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 11). 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). When they do, they see an animation of the thought bubble of the robot expanding and the player is now controlling the robot and the box with one zero is there in the robot's thought bubble. Nothing else is there so most players try to train the robot to copy the box. The robot started off with a very small memory – only enough to remember two steps. When the player trains the robot to copy the box and then join the copy with the original box, the robot's memory is full so his thought bubble shrinks back down and the player is back on the floor with the robot and the box. The robot now knows what to do when given a box so it repeats what it was trained to do and copies the box.

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 these tasks become the player's responsibility.

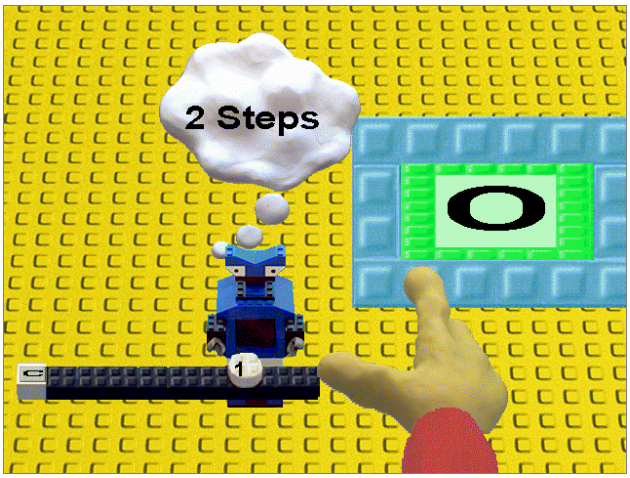


Figure 11 – The initial state of the eighth puzzle

By the time the player starts the ninth puzzle, he or 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. Again the robot is limited to remembering only two step programs. Because there is 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.

The Pedagogical Importance of the Puzzle Game

There are many ways in which children can learn to program [Kah98]. Typically the most effective is by many one-on-one sessions with a very knowledgeable and skilled teacher. Unfortunately, such teachers are in short supply. On the other hand, some children learn well by a self-directed exploration of ToonTalk in free-form mode. However, they seem to be the minority. Most children ask what they should do next rather than explore on their own. For them the puzzle game is ideal. They find it fun and challenging but don't feel lost. A good series of puzzles actually 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 so constrain the set of objects that can be used and how they can be used that the player has only a few choices.

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. The ToonTalk puzzle game deals with this by keeping Marty in another place. If you come to him 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. This behavior accommodates a wide range of learning styles.

In addition to free form exploration and the puzzle game, ToonTalk also includes demos. Here the child can observe a playback, with synchronized narration, of the demonstration of some programming construct or technique. This passive viewing has its place for many children but the learning is typically superficial unless followed by puzzle solving or free form construction.

An automated intelligent tutor [Sel94] is another approach to teaching programming. As the user performs actions the tutor maintains a user model and attempts to tutor the user. In ToonTalk's free-form mode, Marty can play the role of a tutor. He maintains a database of which actions the user has performed and suggests new ones. The suggestions are based upon what the user is currently doing and what he or she has already done. This kind of tutoring has worked well with a minority of the testers. It is a good complement to the other learning modes. A much more sophisticated automated tutor would presumably be much more effective but much more difficult to implement.

Testing

The puzzle game of ToonTalk has been tested with two classes of 24 children in a fourth grade class in Menlo Park, California. Each pair of children was observed using ToonTalk for three 40-minute sessions. 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.

The ToonTalk puzzle sequence has a "self-testing" character. 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 (construct programs). The fact that the children succeeded in solving the puzzles indicates that the puzzles have succeeded and the children are learning computer programming.

There is also a beta version available on the Internet (see http://www.toontalk.com) and testers around the world have worked with the puzzles. The resulting anecdotal evidence is that a wide range of people can work through the puzzle sequences. For example, a 6-year old boy in Colorado solved the first 25 puzzles. He was not able to read, however, and required an adult to read the text. ToonTalk has since been augmented with a text-to-speech capability so even that kind of help is no longer necessary. Another example is a 25-year woman with no programming experience who solved all 62 puzzles on her own.

Related Work

Rocky's Boots and *Robot Odyssey* were two games from The Learning Company in the early 1980s that excited many computer scientists. In these games, one can build arbitrary logic circuits and use them to program robots. This is all done in the context of a video 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 that inspired the work reported here. Scott Kim [Kim95] has written about puzzle design and the pedagogic role of puzzles.

In the last 15 years there have been several attempts to build intelligent tutors that can teach programming (e.g. [Sel94]). This approach has its place in the pool of pedagogic tools but requires a much stronger external motivation for learning programming. The ToonTalk puzzle game is fun – children play it for pleasure, in some case without even knowing that they are learning computer-programming skills.

Discussion

The design and informal testing of a sequence of interactive puzzles designed to teach programming concepts and techniques has been presented. This work raises many questions. Is the technique limited to animated or visual programming languages? Could a similar puzzle sequence be designed for LOGO or Java? How really does a puzzle sequence differ from a more traditional problem set?

Let us try to imagine a puzzle sequence for LOGO. The equivalent of the ToonTalk puzzle constraints would be to present the player with a "TO", 3 "+"s, a "1", 2 variables, etc. and ask them to compute a number greater than 1000. Such puzzles might be as intellectually challenging as the ToonTalk puzzles, but probably would be harder and less fun than the ToonTalk puzzles. They would be harder because of a lack of direct manipulation and instant feedback. They would be less fun because it is hard to add the element of narrative and characters that is central to the ToonTalk puzzles. As a result such puzzles would probably feel more like problem sets. The puzzles in ToonTalk don't just *test* a student's skills and knowledge but *teach*. This is because the student is placed in an environment where they can safely explore a large, but not too large,

space of possibilities. And the exploration is immediate — every intermediate step they take has direct visible and audible effects.

Future Work

The testing to date has not been formal and precise. We hope to find and collaborate with another group interested in studying the effectiveness of these puzzle sequences in a more formal manner.

Other puzzle sequences can be imagined that would teach different computer programming topics like concurrency, algorithms, user-interface design, and distributed systems. Puzzle sequences can be designed for specialized domains like computer music, or game programming, or database programming. The dream underlying this research is that learning hard topics like computer programming can be made more effective and more fun by borrowing ideas from video and computer games.

References

[Kah95] Ken Kahn, ToonTalk™ -- An Animated Programming Environment for Children, *Proceedings of the National Educational Computing Conference*, Baltimore, Maryland, June 1995. Also an extended version appears in the *Journal of Visual Languages and Computing*, June 1996.

[Kah96a] Ken Kahn, Drawings on Napkins, Video Game Animation, and other ways of Programming Computers, *Communications of the ACM*, August 1996

[Kah96b] Ken Kahn, Seeing Systolic Computations in a Video Game World, *Proceedings of the IEEE Conference on Visual Languages*, Bolder, Colorado, September 1996.

[Kah98] Ken Kahn, Helping Children to Learn Hard Things: Computer Programming with Familiar Objects and Actions, *The Design of Children's Technology,* ed. Allison Druin, Morgan Kaufmann, 1998.

[Kim95] Scott Kim. Puzzle Games and How to Design Them. In *Proceedings of the Ninth Annual Computer Game Developers' Conference*, Santa Clara, April 1995.

[Sel94] Ted Selker. COACH: A Teaching Agent that Learns. In *Communications of the ACM* 37, 7. July 1994.

[Smi75] David Smith, *Pygmalion: A Creative Programming Environment*, Stanford University Computer Science Technical Report No. STAN-CS-75-499, June 1975.