Pers Ubiquit Comput (2004) 8: 326–337
DOI 10.1007/s00779-004-0295-6

ORIGINAL ARTICLE

Timothy S. McNerney

From turtles to Tangible Programming Bricks: explorations in physical language design

Received: 1 November 2003 / Accepted: 27 April 2004 / Published online: 29 July 2004

Abstract This article provides a historical overview of educational computing research at MIT from the mid-1960s to the present day, focusing on physical interfaces. It discusses some of the results of this research: electronic toys that help children develop advanced modes of thinking through free-form play. In this historical context, the article then describes and discusses the author's own research into *tangible programming*, culminating in the development of the *Tangible Programming Bricks* system—a platform for creating *microworlds* for children to explore computation and scientific thinking.

Keywords Programming languages · Microworlds · Tangible user interfaces · Education · Children · History of computing · Construction toys · Hands-on learning

1 Introduction

Over 35 years have passed since Seymour Papert co-founded the MIT Artificial Intelligence Laboratory with Marvin Minsky. However, the results of Papert's research continue to provide inspiration for researchers in educational technology even today, at the dawn of the third millennium. Papert's groundbreaking insights into how computers can help children to learn actively and to create knowledge have firmly withstood the test of time.

This article provides a historical overview of educational computing research at MIT, from Papert's pioneering work in the mid-1960s to the present day, focusing on physical human–computer interfaces. It discusses some of the laboratory prototypes as well as commercial fruits of this research: electronic toys that help children develop advanced modes of thinking through free-form play. In the context of this historical perspective, the article then describes and discusses the author's own research into *tangible programming*. His work culminated in the development of the *Tangible Programming Bricks* system—a platform that was used to build the *Digital Construction Set*. This is a *physical microworld* that children can use to understand the logic of digital devices and explore scientific thinking. Issues of language design that make such *digital manipulatives* [1] easy to learn and use as powerful and concise digital building blocks are also discussed.

Particular attention is paid to the continuation of Papert-style educational computing research through the efforts of his students at the MIT Media Lab. Their work is bridging the gap between abstract computation and the learning abilities of children by bringing computer programming into the physical world as a creative activity. This research on *programmable bricks* and *tangible user interfaces* has led directly to innovative commercial products. Many of these products were manufactured and marketed by the LEGO Group—an ongoing supporter, partner, and beneficiary of MIT's educational computing research for more than two decades. During his tenure in the Learning and Epistemology group at the MIT Media Lab, Papert was appointed the LEGO Professor of Media Arts and Sciences. As of this writing, Papert's student, Mitchel Resnick, now heads the "Lifelong Kindergarten" group at the Media Lab, and currently holds the LEGO Chair.

2 Historical background

2.1 Logo: a programming language for children

In the 1970s, Papert and his students at the MIT AI Lab began researching methods of introducing children to the world of computer science and bringing programming into their physical world. This initial research led to the creation of the Logo programming language and

T. S. McNerney
ChipWrights, Inc., 230 Third Avenue—6th Floor,
Waltham, MA 02459, USA
E-mail: tim@tuva.us
Tel.: +1-781-8393210
Fax: +1-781-8902201

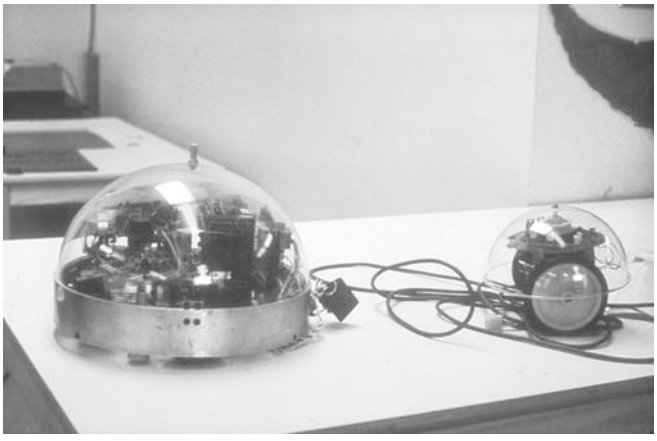


Fig. 1 Two Logo floor turtles (Photo © 2004 Robert W. Lawler, used with permission)

the Logo *turtle*—originally a basketball-sized, dome-shaped robot (Fig. 1) that moved across the floor in response to simple Logo commands like FORWARD, BACKWARD, LEFT, and RIGHT.

Logo is a variant of the LISP programming language, popular among artificial intelligence researchers. It is a powerful yet simple language for exploring linguistic and mathematical problem-solving. Logo is relatively free of the syntactic rules that make so many programming languages difficult for children to learn. At the same time, Logo shares much of LISP's power to create new, domain-specific languages. These properties of Logo led to the slogan: "low threshold and no ceiling."

The Logo *floor turtle*, so-called because it lived on the floor of the lab, could be fitted with a pen and be used to draw pictures on a large sheet of paper placed on the floor. Children learned to "teach" the turtle to draw basic shapes, such as triangles, squares, circles, and even letters. The turtle could also be instructed to draw complex Spirograph[1]-like shapes by repeatedly drawing simpler shapes, rotating the turtle slightly before each repetition.

2.2 The Logo turtle makes geometry physical

Papert espoused *turtle geometry* [2] as a new, more accessible way to teach geometric concepts to children. The core of Papert's approach was the idea that children learn abstract geometric concepts more easily if they can model geometric forms in physical space. Papert found that a child learned even more quickly if she was asked to rehearse geometric forms using his or her own body. This approach is sometimes called *body-centered geometry*. For example, rather than immediately asking a child to describe how she might instruct the turtle to draw a square shape, she is asked to explore the creation of the shape by moving her body; that is, actually "walking the square." Through trial and error, the child soon learns that walking a few steps, turning right 90°, and repeating this four times makes a square. In the process, she might be asked to notice whether she ended up about where she started. After this exercise, the child is better prepared to program the turtle to do the same thing. In Logo, this series of actions can be expressed as follows:

```
TO SQUARE   :STEPS
REPEAT 4    [FORWARD :STEPS
             RIGHT 90]
END
```

In the end, the child is rewarded by watching the turtle move around the floor in the same way as she acted out the procedure beforehand.

2.3 Programming by example

Over years of teaching Logo, it became clear to researchers that most children were not ready to start programming computers in the traditional manner (i.e., typing Logo code into a computer using a keyboard) until at least 10–14 years of age. Radia Perlman, a graduate student at the MIT Logo Lab in the mid-1970s, believed that the major impediments to children's access to computer programming were not only the language syntax, but also the user interface. Perlman [3] proceeded to design interfaces that would allow even preschoolers to learn to program a turtle. She came up with two novel input devices, informally called the *Button Box* and the *Slot Machine*.

The Button Box was her first creation. One prototype was a 50×50×10 cm transparent, plastic box with four groups of buttons:

1. The Number Box: 1–10
2. The Action Box: FORWARD, BACKWARD, RIGHT (turn clockwise), LEFT (turn counter-clockwise), PENUP, PENDOWN, plus "turn your light on," "turn your light off," and "toot your horn" (the clear favorite among the three- and four-year-old test subjects)
3. The Memory Box: "Start remembering," "Do it," and "Forget it"
4. The Four Procedure Box: YELLOW, RED, GREEN, and BLUE The Button Box, in its most basic form, had buttons for numbers and actions (i.e., turtle commands). Young children could use it as a kind of TV remote control to tell the turtle what to do without having to learn how to type commands on a keyboard. With the addition of the Memory Box, the system could record a transcript of the child's turtle commands, and later play them back.

The Memory Box innovation made the Button Box perhaps one of the earliest illustrations of programming by example [4]. Rather than creating a list of commands for a computer to perform, which requires a child to translate concrete actions into abstract directives, the

[1]Spirograph is a trademark of Hasbro, Inc.

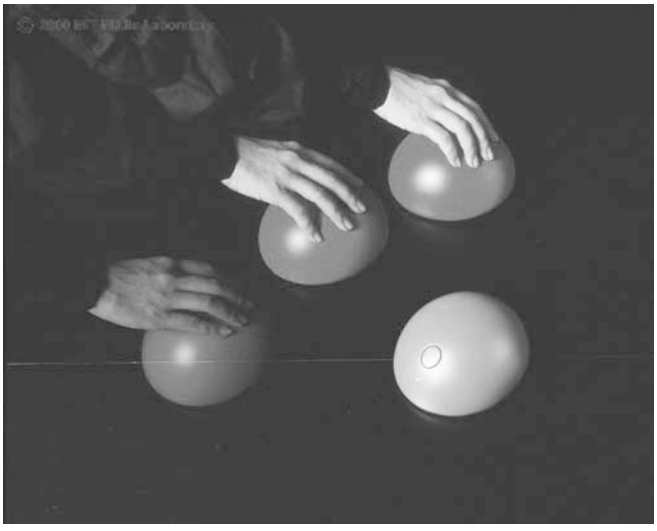


Fig. 2 Rehearsing Curlybot's motion (Photo © 2000 MIT Media Laboratory, used with permission)

child could physically show the computer what to do and have it memorize the sequence for later playback.

It is interesting to compare Perlman's early work with more recent research done in the 1990s by Phil Frei and Victor Su in Hiroshi Ishii's Tangible Media Group at the MIT Media Lab[5]. Affectionately called *Curlybot* (Fig. 2), Frei and Su's digital manipulative looks like a typical one-button computer mouse and fits comfortably in your hand. Its operation is wonderfully simple: press the button and guide Curlybot through a series of motions or gestures. Press the button again, and the motorized Curlybot faithfully repeats your motions. To instruct Curlybot to move in a circle, press the button, move it forward a bit, turn it a bit, and press the button again. Attach a pen, and Curlybot will draw a circle on the table, just like a turtle instructed using the Logo command sequence:

```
FOREVER [FORWARD 5 RIGHT 5]
```

Teaching the Curlybot to move about is not really programming in the traditional computer science sense, but a more direct form of communication that Brenda Laurel might call *programming by rehearsal* [6].

2.4 Playing the Slot Machine

Perlman soon discovered two major problems with her Button Box system. First, the concept of procedure calling, as implemented by the Button Box's "Four Procedures Box," was too abstract for younger children. Second, the system provided no way for a child to modify a program once it was recorded. If the child found a mistake, he had to re-record the entire sequence from the beginning. This last issue did, however, mean that children were not presented with the somewhat complex task of editing "buggy" programs.

Perlman's next system, the Slot Machine (Fig. 3), addressed both issues. Instead of controlling the turtle using

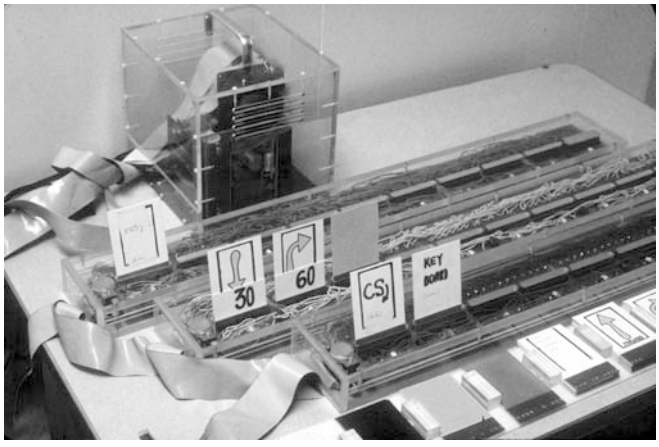
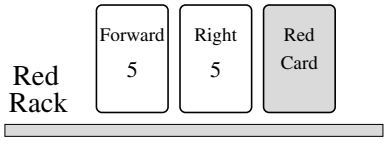


Fig. 3 Radia Perlman's Slot Machine (Photo © 2004 Robert W. Lawler, used with permission)

buttons, as with the Button Box, the Slot Machine employed plastic cards that could be inserted into one of three colored racks (red, yellow, and blue). On the left side of each rack was a "Do it" button. When the child pressed this button, the turtle performed the actions pictured on each card in the rack, in sequence from left to right. As an action was being executed, a lamp under the card illuminated.

The Slot Machine design offered a major advantage over the Button Box: the ability to manipulate a program directly, by adding, rearranging, and removing cards by hand. To address the procedure-calling issue, Perlman introduced special cards whose colors corresponded to the colored racks. For example, if a blue card was encountered, program execution continued at the beginning of the blue rack and returned to the original rack (i.e., the "caller") after the last card on the blue rack was completed. The colored cards could also be used to create FOREVER loops via *tail recursion*, by building a procedure that called itself. For example:



2.5 The turtle goes to school

The Logo programming language acquired unprecedented popularity in the early 1980s when personal computers (particularly the Apple II) became common in schools. However, few robotic turtles made it to school—although fine for pioneering laboratory research, the robots proved expensive, unreliable, and unsuitable for exploring complex geometric patterns. Educational computing researchers at MIT temporarily moved away from this approach, and instead developed the *screen turtle*, a representation of the floor turtle on the computer's video display. The screen turtle became the standard for Logo-style computational geometry. The notion of physically rehearsing geometric ideas was

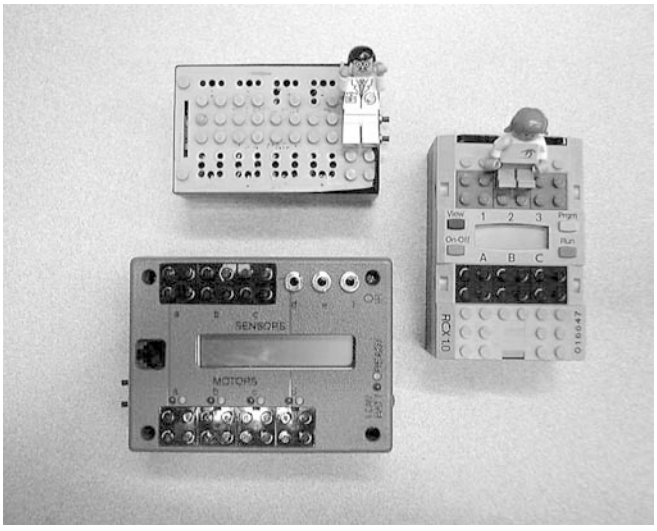


Fig. 4 The evolution of programmable bricks: early *Gray Brick* (top); later *Red Brick* (bottom); and commercial LEGO RCX (right) aka *Yellow Brick* (Photo by F. Martin)

certainly not abandoned, but the practice of using the floor turtle as a physical *thing to think with* lay dormant for a number of years.

Though the screen turtle was obviously more practical, it added a layer of abstraction that may have caused difficulty for some children, because the screen turtle was removed from the floor, and thus removed from the child's everyday environment. Also, the distinction between turning left and turning right was less intuitive, since the turtle was on a vertical display screen, whereas floors are horizontal. Nevertheless, the virtual nature of the screen turtle offered new opportunities for learning and exploration that were hard to resist.

2.6 Connecting with the physical world: LEGO and Logo

It was not until the 1990s that educational computing researchers at MIT returned in earnest to studying interaction with the physical world. The LEGO *TC Logo*[2] interface developed by Steve Ocko allowed young inventors to write Logo programs on a personal computer to control devices built out of LEGO building blocks. Even though these contraptions were tied by an "umbilical cord" to a computer, they could perform simple yet amazing real-world tasks, for example, sorting LEGO pieces by size.

Fred Martin, Randy Sargent[3], and Brian Silverman[4] helped Logo break free of the umbilical cord. Whereas Steve Ocko's interface required a separate external computer, Martin, Sargent, and Silverman built the computer directly into special programmable LEGO bricks (a.k.a. *P-Bricks*, shown in Fig. 4). Each P-Brick contained a battery-powered microcomputer running a Logo interpreter [7]. Like Ocko's tethered interface, the P-Brick could be used to sense its environment and control motors. LEGO eventually commercialized the P-Brick and sold it as the LEGO *Mindstorms Robotics Invention System*.

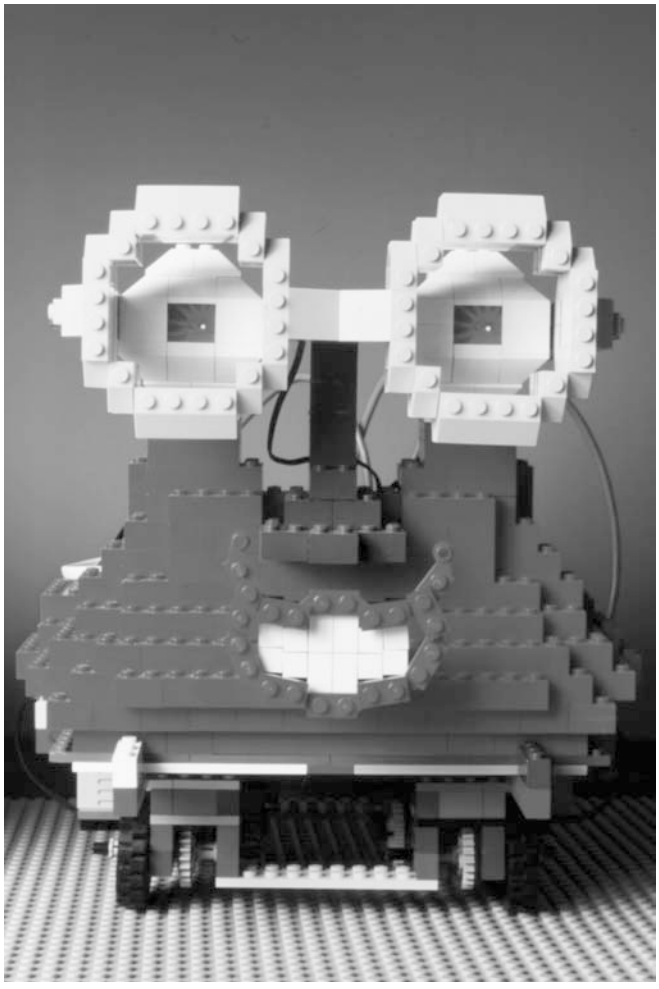


Fig. 5 Dr. Legohead

2.7 Toys with "personality"

The P-Brick was the inspiration and "brain" for a number of "magical machines" [8], toys with "behavior" [9], designed for children by children at heart. Rick Borovoy created *Dr. Legohead* (Fig. 5), an interactive character inspired by Mr. Potatohead[5]. Like Mr. Potatohead, its eyes, ears, nose, and mouth are removable, and in the spirit of "there is no wrong," body parts can be inserted in funny places. However, don't expect the

[2]LEGO is a trademark of the LEGO Group.

[3]Fred Martin co-created the now famous 6.270 MIT LEGO robot competitions with Randy Sargent, author of the "Interactive C" robot programming language.

[4]Brian Silverman was one of Papert's early students at MIT, who went on to become a founder of Logo Computer Systems, Inc. (LCSI).

[5]Mr. Potatohead is a trademark of Hasbro, Inc.

Fig. 6 LEGO MyBot

good Doctor to tolerate facial modifications with as much stoicism as his low-tech cousin. Unlike Mr. Potatohead, high-tech Dr. Legohead talked, and manipulating his removable body parts and accessories trigger humorous responses. For example, the Doctor would make silly comments if you inserted his eyes backwards, and he would act vain if you took a picture of him with a toy camera, making jokes about his nose being too big [10].

The LEGO *MyBot* (Fig. 6) is another toy inspired in part by research at MIT. This highly reconfigurable toy-building system was released in 2000. It consists of building blocks that can be used to make a racecar, an airplane, a towering mechanical monster, or any number of creative hybrids. Special *code bricks* give the child's creation additional personality. The toy emits different sound effects in response to a tilt sensor. Optional modifier code bricks can be used to configure the toy to exhibit supplemental behaviors, such as "car alarm," "laser tag," and "gas pump." For instance, with the gas pump code brick in place, the toy periodically "runs out of gas" and waits for the child to "fill'er up." When the child inserts a hose into the gas tank receptacle, the car, plane, or monster makes gratifying gurgling, "gas filling" sounds. Even young children quickly latch on to this sort of activity because it mimics familiar concepts from the world around them. In this case, that gasoline makes cars "go" and that one visits the gas station periodically to buy more.

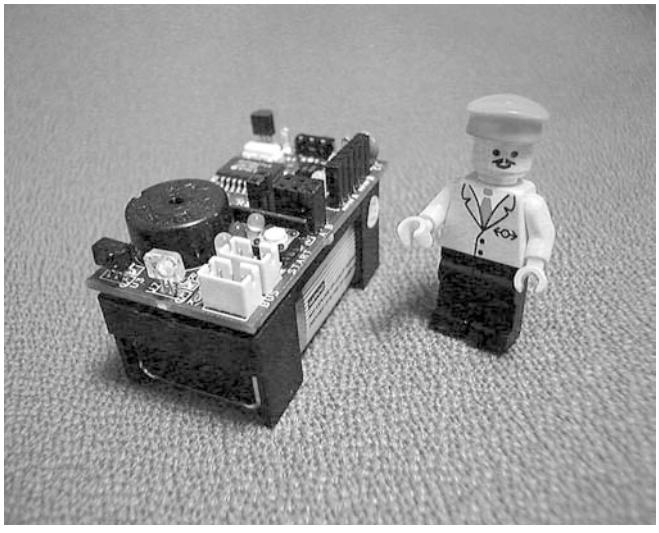


Fig. 7 MIT Cricket

While code bricks like "gas pump" definitely suggest specific interaction, the main "personality" bricks in the MyBot system do not impose any explicit scripting on the child's play. These bricks simply make non-imposing sound effects, variations on the timeless "vroom" (car) and "neeaw" (airplane) sounds.

2.8 P-Bricks "go mini": the cricket and its progeny

Continuing with the P-Brick concept, Martin, Silverman, and Robbie Berg of Wellesley College invented the *Cricket* [11]. The Cricket (Fig. 7) is a scaled-down version of the P-Brick based around the PIC microprocessor family[6]. The MIT "blue dot" Cricket is about the size of a 9-volt battery[7]. It can control two motors, read two analog sensors, and command peripherals via a serial bus. It has a buzzer to beep and play tones. It can communicate with other Crickets via infrared light. Logo programs written on a personal computer can be downloaded to a Cricket using the same infrared link. The user can also interactively send Logo commands to the Cricket by typing commands into a special window. This last feature is extremely useful for debugging and encourages incremental testing.

2.9 Tangible Mozart

Lackner, Dobson, Rodenstein, and Weisman used a team of Crickets to build their *Sensory Puzzles* [12] designed for tactile and auditory exploration of music. A commercial toy of similar design is Neurosmith's award-winning *Music Blocks* (Fig. 8), one of the best toys to

[6]PIC is a trademark of Microchip, Inc.

[7]Fred Martin's commercial version, called the *Handy Cricket*, is about the size of 4×AA batteries.

Fig. 8 Neurosmith Music Blocks

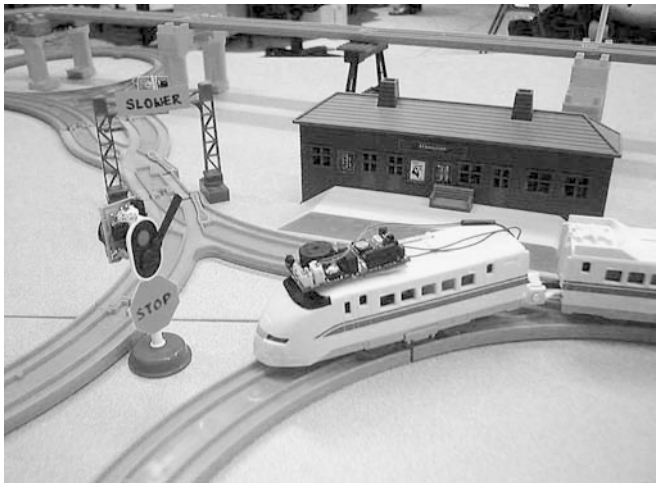


Fig. 9 Tomica train set augmented with a Cricket that heeds IR beacon signs (prototype). Tomica is a trademark of TOMY Company, Ltd

date to feature a truly tangible user interface[8]. The toy comes with five colored cubical blocks, each representing a musical phrase from the opening of Mozart's well-known *Eine Kleine Nachtmusik*. The child arranges the blocks on a music player. When a block is inserted, it plays its musical phrase in isolation. Pressing the "play" button on the player (the only button on the toy) plays the entire sequence in whatever order the child has chosen.

The musical phrases in the Music Blocks toy were carefully chosen so that the music sounds "right," regardless of the order in which the phrase blocks are arranged. The child can create new, entirely plausible variations, or try to re-create the Mozart original. Adding to the fun, the different faces of the cubical blocks play different arrangements of the music, from "doo wop" *a capella* to classical orchestral.

2.10 Tangible programming with trains

Genee Lyn Colobong and Martin's unpublished 1998 research *Tangible Programming with Trains* (Fig. 9) was commercialized by a number of toy companies. One such progeny, the LEGO *Intelli-train* comes with a battery-powered locomotive whose actions can be programmed by placing special, flat, code bricks on the track. For example, when the train rolls over the "toot" brick, it sounds its horn. When the train rolls over the "reverse" brick, it changes direction. When it encounters a "stop" brick, the train halts and waits for the child to press the green "go" button. By adding, removing, and moving code bricks, the child can modify the "program" even while the train is "running" it. Other code bricks include "stop to pick up passengers" and "stop to get fuel." Again, these are important because they relate to the child's real world and because they encourage storytelling.

In informal experiments, the author noticed that the concept of preparing train tracks with instructions to be executed later by the Intelli-train was initially not intuitive to four-year-olds. Shown how the code bricks affect the train's behavior, their first instinct was to place the code bricks directly in front of the approaching train to achieve immediate results. It is too bad that this approach mostly results in derailments, because providing a mechanism for "interactive" program execution would make a good transitional learning activity: immediacy first, delayed "programming" later.

2.11 The AlgoBlock system: a tangible video game

The term "tangible programming" was actually coined by Suzuki and Kato, developers of the *AlgoBlock* system (Fig. 10) [13] to study collaborative problem solving. The AlgoBlock system consisted of a collection of relatively large computational building blocks (approximately 15-cm cubes) that children used to direct a submarine through an underwater maze. Their language was very similar to Logo. Although the task of programming was physical, the effect of running a program was "virtual": guiding a submarine on the computer screen.

[8]Their paper was accepted for publication concurrently with the commercial release of the Music Blocks product, which was developed independently by Neurosmith.

Fig. 10 Children programming an on-screen submarine using the AlgoBlock system

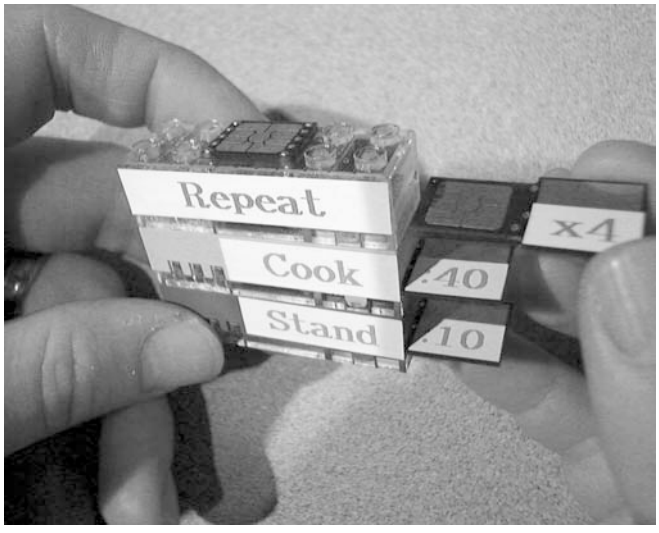


Fig. 11 Affordance: insert a card

2.12 Tangible programming bricks

The author's own research into *tangible user interfaces* (TUIs) started with the Cricket technology. In 1999, the author designed and built the *Tangible Programming Bricks* system [14] as a platform for exploring tangible programming languages. Once the hardware was designed and built, the challenge was to find a tangible programming language that met the following design criteria:

- Easy to understand
- Conducive to free-form play
- Small vocabulary–lexicon of building blocks
- Large number of legal programs
- Debuggable without external tools

In the interest of simplicity and accessibility, the author chose to build a 1-D system of stackable LEGO bricks that could be used for building programs out of simple, atomic token bricks. He quickly found that using such simple bricks was too constraining. In particular, there was no way to specify numeric parameters, like in Perlman's Slot Machine (the "30" and "60" in Fig. 3). To make the set more powerful, he redesigned the bricks, adding a card slot to the side of each brick. This feature was added initially to support parameters, but it soon became evident that the slot allowed a rich array of accessories to be added, such as digital thumb wheels and analog knobs, thus, providing additional manipulability. Using Don Norman's terminology in [15], the Tangible Programming Bricks now had two *affordances*: "insert card" (Fig. 11) and "stack bricks" (Fig. 12). The connector system used for stacking bricks relies on the plastic knobs and tubes of the LEGO SYSTEM for physical *clutching*, and employs a "smart card" (ISO 7816) connector for electrical contacts. The bricks can be stacked successfully in two orientations (0° and 180°). They are physically prevented from snapping together in other orientations (90° and 270°).

2.12.1 Implementation

The technology inside a Tangible Programming Brick is a close relative of the Cricket's. They both have a PIC microprocessor inside, each running a Logo interpreter, and have a programmable memory to store a Logo program. The program inside gives each brick its unique computational behavior. There are no batteries inside the bricks. They all receive power from a single battery brick at the base of the stack. Each brick has connectors on the top and bottom used for power and communication, three colored lights, and a peripheral card slot. In the final implementation, the card connector provided electrical circuits for:

- Power and ground
- I^2C serial bus
- Buzzer or capacitive touch sensor

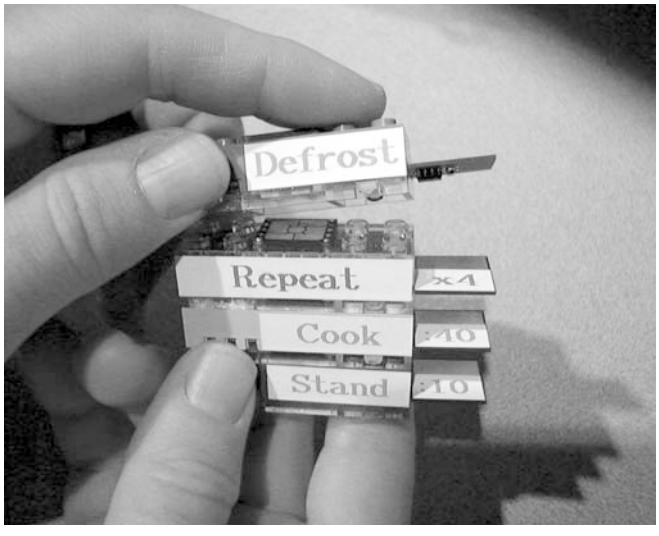


Fig. 12 Affordance: add/stack a brick

– IR transceiver
– Serial *Cricket bus*
– Analog sensor input.

A brick can communicate with its neighbors. It can send messages containing events or data to its neighboring brick above, and it can receive (and answer) messages from its neighbor below. This asymmetry leads to a natural upward data flow. Because each brick can only communicate with its immediate neighbors, passing data from the bottom brick to the top brick in a stack requires the cooperation of all the bricks in between to form a "bucket brigade."

2.12.2 Early experiments

Early experiments with the Tangible Programming Bricks focused on programming external devices, like toy cars and kitchen appliances, using a sequential, procedural language. The building blocks shown in Figs. 11 and 12 were from an early experimental language aimed at the *end-user programming* of microwave ovens.

2.13 Functional programming is well suited to physical interfaces

In the spring of 2000, a more successful tangible programming language emerged. Previously, the author had been working in much the same direction as his predecessors: researching physical media for *imperative programming*. The author began exploring the physical expression of another programming style, *functional programming*.

An *imperative program* describes an algorithm, or "how to" do something. When a child programs a Logo turtle or Intelli-train, he sets up a sequence of commands (a.k.a. *imperatives*) that the device executes one-at-a-time, in the order that it encounters them. A *functional program* denotes "what is" being computed. The most familiar example of a functional programming system (although unbeknownst to many of its users) is an electronic spreadsheet program. Each cell of a spreadsheet describes a piece of the computation. The order of evaluation is not explicitly specified; rather, it is derived from the relationships between the cells. For instance, if cell B2 depends on the value of A4, then A4 will be computed before B2.

The new language created by the author, dubbed the *Digital Construction Set* (see Fig. 13 and corresponding legends in Table 1), was inspired by the functional programming concepts of *combinators* [16] and *streams* [17],

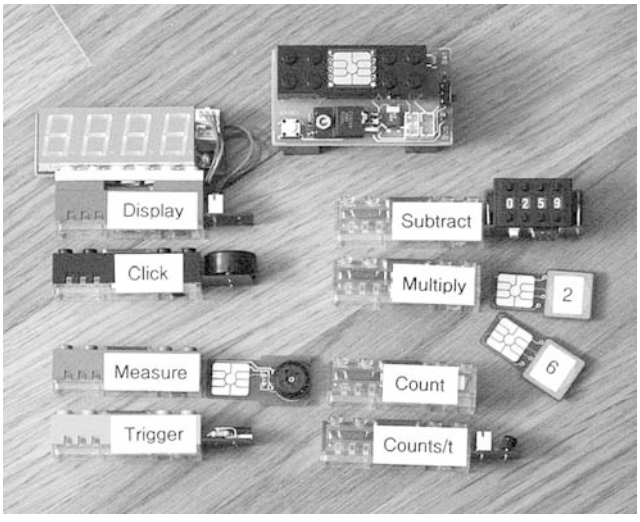


Fig. 13 Digital Construction Set: vocabulary

Table 1 Digital Construction Set for Bricks and Cards (refers to Fig. 13)

	Function
Brick	
BATTERY	This provides a computationally passive foundation for building a stack of bricks (the unlabeled brick in top right of figure)
MEASURE	Generates a continuous stream of samples from analog SENSOR, KNOB, THUMBWHEEL, or CONSTANT cards
TRIGGER	Generates a stream of discrete events triggered when the analog sensor data rises above a threshold (no event is generated when sensor data falls)
MULTIPLY	Multiplies input data stream by a value supplied by card
SUBTRACT	Subtracts from input data stream a value supplied by card
COUNT	Tallies events (ignores data associated with each event)
COUNTS/T	Counts the number of events in a specific period of time (e.g., 10 s)
DISPLAY	Displays, in succession, each element of input stream
CLICK	Clicks whenever it receives a new data element (a.k.a. event)
Card	
TOUCH/CONSTANT	Used either as a push-button, or as a symbolic or numeric parameter
KNOB	Used to enter a variable input parameter
BEEPER	Gives the CLICK brick its "voice," or can be used for other sound, music, or noise-making
IR/BUS	Allows the user to download Logo code into a brick to give it its own distinct behavior; can be used to communicate with Crickets, other stacks of bricks, or peripherals (e.g., display)
THUMBWHEEL	Used for entering numbers that are not available in the limited set of CONSTANT cards (seen showing "0259" in figure)
SENSOR	Used to sample analog data from sensors (e.g., to measure light, temperature, magnetism, etc.)

and introduces the concept of *sequencers*. In traditional functional programming parlance, a combinator is a function that creates a new function out of existing functions (e.g., compose: $(f \circ g)(x) = f(g(x))$). In this context, a *function* is constrained to be a mathematical transformation whose output is dependent solely on its input (e.g., MULTIPLY and SUBTRACT). In other words, for a given set of input value(s), a function always computes the same output. By contrast, a *sequencer* is allowed to remember previous data (i.e., *state*) and incorporate this history into the calculation of its output (e.g., COUNT and COUNT/T, described in Table 1). Both types of linguistic building blocks can be used in isolation or together to transform a stream of input data (e.g., sensor data) into an output data stream that can be used in turn as input by another brick. This is similar in spirit to how a Unix/Linux shell command can be *piped* to another command. A major advantage of the streams programming model is that control flow is implicit. Programs using streams are naturally concise, and the building blocks can easily be recombined to solve new problems.

The message-passing architecture of the Digital Construction Set uses a variation of the ''bucket brigade'' concept described above. Here, a brick is allowed to modify each element of the data stream as it passes it up the chain. For example, a MULTIPLY brick with a ''2'' card takes each number received from its lower neighbor, multiplies it by two and passes the result up to its upper neighbor. So, if the brick receives a 21 from below, a 42 is passed up the chain.

2.14 Building physical microworlds

The Digital Construction Set was designed for children to explore the logic of everyday electronic devices through hands-on learning. In his seminal book, *Mindstorms* [18], Papert introduces the concept of a *microworld*, a deliberately simplified computational environment that allows students to explore ideas that are not normally demonstrable in an average classroom setting. Papert describes how microworlds can be used to learn aspects of Newtonian mechanics that students never experience in their everyday lives, such as the motion of objects in a vacuum. Children, programming a *Dynaturtle* (dynamic turtle, a.k.a. *sprite*) in Logo, can design experiments not only to model traditional Newtonian mechanics, but also alternate laws of motion (e.g., Aristotelian), or even fictional universes whose natural laws are defined entirely by the student. Papert's theory is that, when a child creates a simulation by herself, she has ''made it her own,'' it becomes personal, and thus, more powerful.

2.14.1 ''Getting under the hood''

Fifty years ago, home appliances and scientific instruments were made from individual components and invited amateur disassembly, experimentation, and learning—but the microprocessor changed all that. Even if you did open the case of a modern gadget, there is almost nothing to see that can be used to understand its inner workings. The National Science Foundation funded the *Beyond Black Boxes* initiative [19] to explore the theory that children more readily learn scientific concepts by building their own scientific equipment.

2.15 Bricks at work: a bicycle trip computer

The Digital Construction Set was designed to be used as a tool for learning the relationships between physical concepts of time, distance, and velocity, as well as the relationship between Celsius and Fahrenheit temperature scales. Figure 14 shows the Digital Construction Set in action. This *physical microworld* was inspired by one of Brian Silverman's anecdotes. His daughter wanted to understand how a bicycle trip computer works inside. Silverman knew that nothing inside the device would help his explanation, so together, they built one from scratch using a Cricket. They mounted a magnet to the rim of her bicycle, attached a Hall-effect (magnetism) sensor onto the bicycle frame. Then, they programmed the Cricket using Logo to count wheel revolutions, to compute and finally display the bicycle's speed and distance traveled.

The Digital Construction Set provides all the building blocks required to make a simple bicycle trip computer. In the simplified example shown here, the Hall-effect sensor on the toy bicycle is wired to the TRIGGER brick. Every time the bicycle wheel goes around, the magnet on the wheel rim moves past the sensor. When TRIGGER detects this, it sends a message representing an event up to CLICK, which beeps once per message, and forwards the message to COUNTS/T. This brick tallies messages within a 10-s interval and passes the total count to DISPLAY. In the full example, MULTIPLY would be used to compute velocity (e.g., MPH or KPH).

A second DISPLAY (shown in the lower right of Fig. 14) is connected directly to COUNTS/T. It is used to view the state of its internal counter. This was added

Fig. 14 Bicycle trip computer microworld

because, in early versions of the Digital Construction Set, the counter's internal state was invisible during the 10 s between clock ticks. This made it difficult to explain the operation of COUNTS/T to children. The lesson here: "Avoid hidden states. Make everything visible."

2.16 Features of the Digital Construction Set

2.16.1 Self-debugging

A valuable aspect of the Digital Construction Set is that it naturally encourages the user to start simple and to add complexity on top of already working structures. As the system becomes more complex, users spend more and more time debugging—figuring out why things do not work as expected. For this, it is important to be able to see what is going on inside the computation [20].

The Digital Construction Set offers the ability to debug itself. The CLICK and DISPLAY bricks can be used to examine intermediate results in a stack of bricks, as well as to understand the internal message-passing architecture. These special bricks do not have any computational effect on the values passed up the stack. For example, the CLICK brick can be used to learn the difference between TRIGGER and MEASURE. When CLICK is placed above MEASURE, CLICK beeps rapidly. Put it above TRIGGER, and it beeps only after high–low transitions of the input data. Before an external sensor is connected, TRIGGER can be tested in isolation by using the KNOB card. Each time the knob is turned clockwise, CLICK beeps once. When the knob is turned counter-clockwise, no beep is heard.

This self-debugging feature turned out to be very useful when the author was first programming the bricks that make up the Digital Construction Set. Later, when the KNOB and THUMBWHEEL cards were added, the CLICK brick was used to listen for bugs in the internal message-passing system. Children can debug their creations in the same manner.

2.16.2 Consistency and simplicity

A fundamental design tenet of the Digital Construction Set is simplicity through consistency. In particular, the CONSTANT, KNOB, SENSOR, and THUMBWHEEL cards can be used interchangeably. For example, if a child wants to understand how the THUMBWHEEL works, she can simply insert it in the MEASURE brick and attach a DISPLAY brick above it. It becomes a game to manipulate the digits of the THUMBWHEEL and watch the DISPLAY change. The design of the Digital Construction Set makes it easy for users to understand the behavior of individual building blocks and small assemblies before moving to bigger projects.

2.16.3 Programmability

Perhaps the most powerful feature of the Digital Construction Set system is its ability to allow researchers and even young programmers to "get under the hood" of the bricks themselves and reprogram the behavior of any brick. In Papert's spirit of technology that children can "make their own," children are offered the opportunity to design their very own physical microworlds by inventing new, domain-specific vocabulary—a new set of building blocks suited to a particular task.

2.17 Why not allow different topologies?

Many *constructive tangible user interfaces* (e.g., [21, 22]) use computational building blocks to specify a wide range of geometric and topological structures. However, the Tangible Programming Bricks only stack in one direction—up. One cannot build 2-D or branching structures using the Digital Construction Set. The author deliberately chose this approach to avoid adding layers of complexity. Two of the main goals of this research were ease of explanation and ease of learning. Although the linear stacking constraint does limit expression, the overall simplicity of the Digital Construction Set is a clear benefit for the novice. Simplicity is an important feature of a Papert-style microworld.

2.18 Evaluating the Digital Construction Set

The Tangible Programming Bricks were demonstrated to hundreds of adults in the course of testing the suitability of various types of physical programming languages. Compared to the prototype languages based on an imperative programming model, the author found the streams-based programming model of the Digital Construction Set to be dramatically easier to explain to adults with both technical and non-technical backgrounds.

A few users were confused by the physical similarity of the connector systems used for stacking bricks and inserting cards. This was because the same gold "smart card" pattern appeared on both the bricks and the cards. Also, there was nothing to prevent cards from being inserted upside down. This would be easy to fix in a commercial implementation.

Only limited testing was conducted with children. To evaluate the Digital Construction Set, the author arranged informal user sessions with four children ranging from 6 to 13 years of age. A session consisted of a 10-min demonstration of the Digital Construction Set, followed by 10–20 min of free-form play and experimentation. An 11-year-old girl and a 13-year-old boy were successful at solving two problems posed to them with little prompting. The girl also launched into an impromptu activity using the bricks and the bicycle to measure distance along a yardstick. The two younger children did not really "take" to the activity. The author speculates that a less scientifically advanced microworld would have been more appropriate for younger children.

In fact, subsequent research studies by Wyeth and Purchase into the usability of tangible programming

systems found that preschoolers can create simple programs and benefit from the physical immediacy of stackable, electronic blocks [23]. The blocks of Wyeth and Purchase were much simpler than the Digital Construction Set. They employed on/off sensors, lights, and Boolean logic functions similar to the commercial toy, *LogiBlocs*.

2.19 Other related research

The research described in this paper is within an area situated squarely between *visual programming languages* [3, 24, 25], *direct manipulation* [26], *tangible user interfaces* [14, 27–31], and *end-user programming* [32]. It focuses specifically on children as programmers. In addition to the programming systems described in this document, a number of other systems have been designed for children: derivatives of Logo (e.g., *StarLogo* [33]), *ToonTalk* [34], *Cocoa* (a.k.a. *KidSim*) [23], and *Agentsheets* [35, 36], just to name a few. Martin's *Braitenberg Bricks* [37] demonstrated a physical embodiment of Braitenberg's behavior language described in [38].

3 Conclusion

The monumental work of Seymour Papert and his students both at MIT and elsewhere have provided a rich body of research into the field of educational programming and tangible user interfaces. The author has used this work as a foundation for the development of the *Digital Construction Set*, exploring the feasibility of expressing *functional programming* concepts in the physical world. The Digital Construction Set allows teachers and students alike to build *physical microworlds* that children can use to model the behavior of digital devices and to conduct scientific experiments involving sensing and computation directly in the physical world. Key properties of successful physical and language design make this set of *digital manipulatives* easy to learn and use. The author's research suggests that, compared to screen-based user interfaces, tangible user interfaces make computation immediate and, thus, more accessible, and that they are appropriate for children learning about computation and scientific exploration. Compared to languages based on an *imperative* programming model, the *streams* programming model is significantly easier to explain to both children and adults.

There is now a growing body of research in tangible user interfaces that focuses on traditional usability, as well as a mature body of research in elegant programming language design. The author hopes to inspire a new breed of interdisciplinary researchers to combine tangible user interfaces with language design, so that today's tangible toys lead to tangible tools for tomorrow's professionals.

Acknowledgements The author's research was funded by the MIT Media Lab's Things That Think consortium and the LEGO Company (which had no editorial influence over this article). Many thanks go to my thesis committee, Fred Martin, Mitchel Resnick, Hiroshi Ishii, and Hal Abelson for their guidance and encouragement; to Bakhtiar Mikhak and Rick Borovoy for their enthusiastic collaboration; and to Bonnie Friedman, Colin Ferguson, and Shari Goldin for their careful editing and help in preparing this article for publication.

References

1. Resnick M, Martin F, Berg R, Borovoy R, Colella V, Kramer K, Silverman B (1998) Digital manipulatives: new toys to think with. In: Proceedings of the CHI'98 conference on human factors in computing systems, Los Angeles, California, April 1998. ACM Press, New York, pp 281–287. DOI 10.1145/274644.274684
2. Abelson H, diSessa A (1981) Turtle geometry: the computer as a medium for exploring mathematics. MIT Press, Cambridge, Massachusetts
3. Perlman R (1976) Using computer technology to provide a creative learning environment for preschool children. Logo memo no 24, MIT Artificial Intelligence Laboratory Publications 260, Cambridge, Massachusetts
4. Halbert DC (1984) Programming by example. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California
5. Frei P, Su V, Mikhak B, Ishii H (2000) Curlybot: designing a new class of computational toys. In: Proceedings of the CHI 2000 conference on human factors in computing systems, The Hague, The Netherlands, April 2000. ACM Press, New York, pp 129–136. DOI 10.1145/332040.332416
6. Laurel B (1993) Computers as theater. Addison-Wesley, Reading
7. Martin F, Resnick M (1993) LEGO/Logo and electronic bricks: creating a scienceland for children. In: Ferguson D (ed) Advanced educational technologies for mathematics and science. Springer, Berlin Heidelberg New York
8. Martin F, Mikhak B, Resnick M, Silverman B, Berg R (2000) To mindstorms and beyond: evolution of a construction kit for magical machines. In: Robots for kids: exploring new technologies for learning. Morgan Kaufmann, San Francisco
9. Resnick M (1993) Behavior construction kits. Commun ACM 36(7):64–71. DOI 10.1145/159544.159593
10. Borovoy R (1996) Genuine object oriented programming. Masters thesis, MIT Media Laboratory, Cambridge, Massachusetts
11. Martin F, Mikhak B, Silverman B (2000) MetaCricket: a designers' kit for making computational devices. IBM Syst J 39(34):795–815
12. Lackner TM, Dobson K, Rodenstein R, Weisman L (1999) Sensory puzzles. In: Extended abstracts from the proceedings of the CHI'99 conference on human factors in computing system, Pittsburgh, Pennsylvania, May 1999. ACM Press, New York, pp 270–271. DOI 10.1145/632716.632882
13. Suzuki H, Kato H (1993) AlgoBlock: a tangible programming language, a tool for collaborative learning. In: Proceedings of the 4th European Logo conference (Eurologo'93), Athens, Greece, August 1993, pp 297–303
14. McNerney T (2000) Tangible Programming Bricks: an approach to making programming accessible to everyone. Masters thesis, MIT, Cambridge, Massachusetts. Available at http://www.media.mit.edu/people/mc/tangible-programming. html
15. Norman DA (2002) The design of everyday things. Basic Books (Perseus), New York
16. Henderson P (2002) Functional geometry. Higher-Order Symb Comp 15(4):349–365
17. Abelson H, Sussman GJ, Sussman J (1996) Structure and interpretation of computer programs, 2nd edn. MIT Press, Cambridge, Massachusetts
18. Papert S (1999) Mindstorms: children, computers, and powerful ideas, 2nd edn. Basic Books, New York

19. Resnick M, Berg R, Eisenberg M (2000) Beyond black boxes: bringing transparency and aesthetics back to scientific instruments. J Learn Sci 9(1):7–30
20. Ungar D, Lieberman H, Fry C (1997) Debugging and the experience of immediacy. Commun ACM 20(4):38–43
21. Anagnostou G, Dewey D, Patera A (1989) Geometry-defining processors for engineering design and analysis. Vis Comput 5(5):304–315
22. Anderson D, Frankel JL, Marks J, Leigh D, Sullivan E, Yedidia JS, Ryall K (1999) Building virtual structures with physical blocks. In: Proceedings of the 12th annual ACM symposium on user interface software and technology (UIST'99), Asheville, North Carolina, November 1999. CHI Letters 1(1):71–72
23. Wyeth P, Purchase H (2002) Tangible Programming elements for young children. In: Extended abstracts from the proceedings of the CHI 2002 conference on human factors in computing systems, Minneapolis, Minnesota, April 2002. ACM Press, New York, pp 774–775. DOI 10.1145/506443.506591
24. Begel A (1996) LogoBlocks: a graphical programming language for interacting with the world. SB thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, Massachusetts. Available at http://www.media.mit.edu/people/abegel/begelaup.pdf
25. Smith DC, Cypher A, Spohrer J (1994) KidSim: programming agents without a programming language. Commun ACM 37(7):54–67. DOI 10.1145/176789.176795
26. Repenning A, Ambach J (1996) Tactile programming: a unified manipulation paradigm supporting program comprehension, composition, and sharing. In: Proceedings of the IEEE symposium on visual languages, Boulder, Colorado, September 1996 pp 102–109
27. Fitzmaurice G, Ishii H, Buxton W (1995) Bricks: laying the foundation for graspable user interfaces. In: Proceedings of the CHI'95 conference on human factors in computing systems, Denver, Colorado, May 1995. ACM Press, New York, pp 442–449
28. Gorbet M, Orth M, Ishii H (1998) Triangles: tangible interface for manipulation and exploration of digital information topography. In: Proceedings of the CHI'98 conference on human factors in computing systems, Los Angeles, California, April 1998. ACM Press, New York, pp 49–56. DOI 10.1145/274644.274652
29. Ishii H, Ullmer B (1997) Tangible bits: toward seamless interfaces between people, bits and atoms. In: Proceedings of the 8th international conference on intelligent user interfaces, Orlando, Florida, January 1997. ACM Press, New York, pp 234–241. DOI 10.1145/604045.604048
30. Ullmer B, Ishii H, Glas D (1998) MediaBlocks: physical containers, transports, and controls for online media. In: Proceedings of the 25th annual conference on computer graphics (SIGGRAPH'98), Orlando, Florida, July 1998. ACM Press, New York, pp 379–386. DOI 10.1145/280814.280940
31. Underkoffler J, Ishii H (1998) Illuminating light: an optical design tool with a luminous-tangible interface. In: Proceedings of the CHI'98 conference on human factors in computing systems, Los Angeles, California, April 1998.. ACM Press, New York, pp 542–549. DOI 10.1145/274644.274717
32. Soloway E, Spohrer J (1989) Studying the novice programmer. Lawrence Erlbaum, Hillsdale, New Jersey
33. Resnick M (1994) Turtles, termites, and traffic jams: explorations in massively parallel microworlds. MIT Press, Cambridge, Massachusetts
34. Kahn K (1996) Drawings on napkins, video-game animation, and other ways to program computers. Commun ACM 39(8):49–59. DOI 10.1145/232014.232028
35. Gindling J, Ioannidou A, Loh J, Lokkebo O, Repenning A (1995) LEGOsheets: a rule-based programming, simulation and manipulation environment for the LEGO programmable brick. In: Proceedings of the 11th international IEEE symposium on visual languages, Darmstadt, Germany, September 1995. IEEE Computer Society Press, pp. 172–179
36. Repenning A, Sumner T (1995) Agentsheets: a medium for creating domain-oriented visual languages. IEEE Comput 28(3):17–25
37. Martin F, Resnick M, Silverman B (1990) Braitenberg bricks: A LEGO-based creature-construction kit. In: Proceedings of the workshop on artificial life conference (ALIFE'90), Center for Nonlinear Studies, Santa Fe, New Mexico, February 1990
38. Braitenberg V (1984) Vehicles: experiments in synthetic psychology. MIT Press, Cambridge, Massachusetts