Tangible Computer Programming: Exploring the Use of Emerging Technology in Classrooms and Science Museums

A dissertation submitted by

Michael S. Horn

In partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Science

TUFTS UNIVERSITY

May 2009

Adviser: Robert J.K. Jacob

Abstract

In considering ways to improve the use of digital technology in educational settings, it is helpful to look beyond desktop computers and conventional modes of interaction and consider the flood of emerging technologies that already play a prominent role in the everyday lives of children. In this dissertation, I will present a research project that builds on tangible user interface (TUI) technology to support computer programming and robotics activities in education settings. In particular, I will describe the design and implementation of a novel tangible computer programming language called *Tern*. I will also describe an evaluation of Tern's use in both formal and informal educational settings—as part of an interactive exhibit on robotics and computer programming called *Robot Park* on display at the Boston Museum of Science; and as part of a curriculum unit piloted in several kindergarten classrooms in the greater Boston area. In both cases, Tern allows children to create simple computer programs to control a robot. However, rather than using a keyboard or mouse to write programs on a computer screen, children instead use Tern to construct physical algorithmic structures using a collection of interlocking wooden blocks. The goal of this work is not to propose that tangible programming languages are general purpose tools that should replace existing graphical programming languages; rather, I will present evidence to support the argument that tangible programming begins to make sense when one considers the contexts and constraints of specific educational settings. Moreover, in these settings tangible languages can compensate for some of the shortcomings of graphical and text-based systems that have limited their use.

Acknowledgements

This work is dedicated to my wife, Diana Reed, for her love and support throughout this journey, and to my daughter, Madeleine, for helping me keep everything in perspective.

I would first like to thank my adviser, Robert Jacob, for giving me both the freedom to explore and the support, encouragement, and advice necessary to succeed. I also would like to thank my collaborators and colleagues at the Tufts Human–Computer Interaction Laboratory, the Tufts Developmental Technologies Research Group, and the MIT Media Lab, including Marina Bers, Erin Solovey, Jordan Crouser, Rachael Fein, Emily Lin, Angela Chang, Audrey Girouard, Leanne Hirshfield, Natalie Rusk, Orit Shaer, and Jamie Zigelbaum. Work at the Boston Museum of Science was a result of a collaboration with Taleen Agulian, Dan Noren, Lucy Kirshner, Erin Solovey, Bill Rogers, and Tim Moore. I also received equipment, advice, and assistance from employees of the iRobot Corporation including Mark Schnittman, Zee Dubrovsky, and John Billington. Mary Murray from the Massachusetts College of Art and Design provided refreshing insights and designs for our tangible programming language, and Daniel Ozick (formerly of iRobot Corporation) contributed code and expertise to the design of the iRobot Create behavior–based interpreter and assembly language. Amelio Vázquez provided Spanish translations for all of the programming blocks in use at the Museum. I would also like to thank Dr. Wallis Raemer and the kindergarten teachers at the Dallin School in Arlington, Massachusetts for their willingness to participate in our pilot studies.

My early work on tangible programming languages was funded in part by grants from the Tufts University Center for Children (TUCC) and the University College of Citizenship and Public Service (UCCPS). Chris Rogers and the Tufts University Center for Engineering Education Outreach (CEEO) have been constant supporters of my work, lending equipment and advice over the years. I would also like to thank the National Science Foundation for its support through the NSF GK-12 Fellowship. The hard work of Brian Gravel, Chris Rogers, Diane Souvaine, Linda Beardsley, and Kevin Staszowski at Tufts, and Nakia Keizer and Ann Ruggiero at the Mission Hill School made the GK-12 program a wonderfully rewarding experience. Finally, I would like to thank the National Science Foundation for its support of this research through grants IIS-0414389 and DRL-0735657. Any opinions, findings, and conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the National Science Foundation.

Contents

Chapter 1

Introduction

Real world learning environments such as K-12 classrooms and science museums are complex and often chaotic places (Allen, 2004; Serrell, 1996; Cuban, 1984). Teachers in classrooms must learn how to balance the needs of anywhere from 20 to 30 students at a time with the demands of curriculum and the constraints of a regimented school day. In science museums, the challenge is different. Program developers and exhibit designers must work without the structure and guidance provided by teachers and curriculum. Instead, the goal in these *informal* settings is to devise activities and exhibits that engage a diverse audience and promote self-guided learning and discovery (Allen, 2004).

Given these challenges, it is not surprising that technology, of one form or another, has long been used to support teaching and learning in these environments (Cuban, 1984). The role of computer technology, in particular, has been the focus of close to four decades of research and innovation—not to mention heated debate (see Papert, 1980; Cuban, 2001; Oppenheimer, 2003). For educators, the decision to incorporate computer-based learning activities can be fraught with risk (AAUW, 2000; Cuban, 1984, 2001; Serrell, 1996). Not only are there concerns about what exactly students are doing on multi-media computers connected to the Internet, but teachers may also feel a sense of loss of control and self-doubt about their own proficiency with technology (AAUW, 2000). Furthermore, modern desktop computers, designed primarily as single-user productivity tools for businesses, can be inappropriate for many educational applications (Stewart et al., 1998; Scott et al., 2003). In many cases, students are required to leave their normal work space, crowd around a limited number of desktop computers, and share single user input devices. Likewise, in museums, although computer-based exhibits can be very engaging for individual visitors, they are often detrimental to the interactions of social groups as a whole (Heath et al., 2005; Hornecker & Buur, 2006; Serrell, 1996). As a result, many potentially beneficial computer-based activities are simply not included as educational activities (Serrell, 1996; Cuban, 2001).

In this dissertation, I will present a project that builds on emerging human-computer interaction techniques to facilitate the inclusion of computer-based activities in real world educational environments. In particular, I am interested in the potential of tangible user interface (TUI) technology (Ishii & Ullmer, 1997) to support computer programming activities. To this end, I will describe the design and implementation of a novel *tangible computer programming language* that I created called *Tern*. I will also describe the use of Tern in both formal and informal educational settings—as part of an exhibit on robotics and computer programming at the Boston Museum of Science; and as part of a curriculum unit on robotics piloted in five kindergarten classrooms with 93 children (ages 5–7) in the greater Boston area. Tern is similar to many other educational computer programming languages in that it features kid-friendly syntax that children can use to create computer programs consisting of sequences of com-

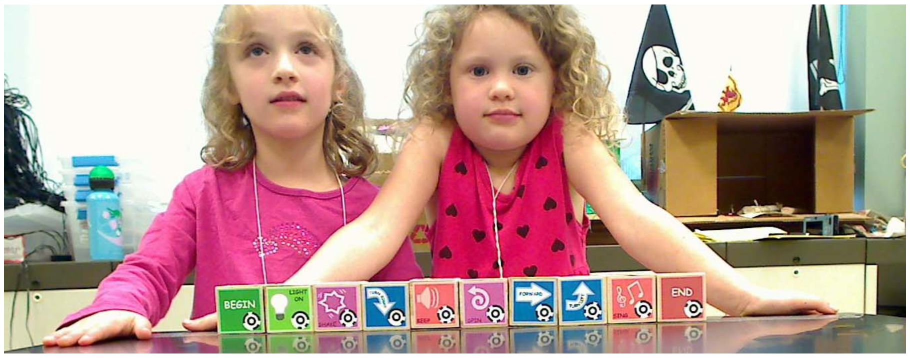


Figure 1.1: Tern is a tangible programming language that allows children to construct physical algorithmic structures using interlocking wooden blocks.

mands combined with basic flow-of-control structures. However, rather than program with a keyboard or mouse on a computer screen, Tern instead builds on tangible interface technology to allow children to construct physical algorithmic structures using a collection of interlocking wooden blocks (Figure 1.1). Tern uses reliable computer vision techniques to convert these physical programs into digital code that is then downloaded to an autonomous robot.

This idea of tangible programming poses many challenges, especially for use in educational environments such as classrooms and science museums. *How do students save their work? What if there aren't enough blocks to go around? Isn't this needlessly expensive? Is computer vision really reliable enough?* Despite these challenges, I will argue that one contribution of my work is a tangible interface for computer programming that is durable, inexpensive, and reliable. Furthermore, while far from perfect, I propose that if one considers the contexts and constraints of specific educational environments, then this type of system can address some of the shortcomings of graphical and text-based languages that I believe have restricted their use in education.

1.1 Overview

My work in tangible programming languages was originally inspired by informal observations of teachers in classrooms over a period of two years who were grappling with the challenge of incorporating computer-based learning activities into their curriculum. My work on early Tern prototypes led to discussions and an eventual partnership with program directors at the Boston Museum of Science who were interested in creating hands-on computer programming and robotics activities. The result of this partnership was the development of a permanent exhibit in the Cahners ComputerPlace Discovery Space at the Museum over a period of two years. The primary contribution of this dissertation involves a study comparing the use of a tangible and a graphical interface as part of this exhibit. For this study, we collected observations of 260 museum visitors and conducted interviews with 13 family groups. Our results show that the tangible and the graphical systems are roughly equally easy for visitors to understand. However, with the tangible interface, visitors were significantly more likely to try the exhibit and significantly more likely to actively participate in groups. In turn, we show that regardless of the condition, involving multiple active participants leads to significantly longer interaction times. Finally, we examine the roles of children and adults in each condition and present evidence that children were more actively involved in the tangible condition—an effect that seems to be particularly strong for girls.

In addition to the study conducted at the Museum, I will describe a pilot study involving the use of tangible programming in five kindergarten classrooms in the greater Boston area. This work is part of a project called *Tangible Kindergarten*, which has the goal of providing age-appropriate curriculum and technology for use in early elementary school classrooms. This project explores the idea that when given access to appropriate tools, young children can actively engage in computer programming and robotics activities in a way that is consistent with developmentally appropriate practice. Our vision is that these types of activities will not only provide children with positive technology and teamwork experiences, but will also provide valuable connections to other academic areas, including literacy, math, science, and engineering.

Despite the substantial differences between learning in science museums and classrooms, many of the design considerations for tangible programming in museum settings such as cost, durability, simplicity, apprehendability, and robustness, are also applicable in classrooms settings. Thus, I will describe the adaptation of Tern for use in kindergarten classrooms based on my experiences in the Museum. I will also describe results from our pilot study, which, in part, explored the differences between tangible and graphical programming in the classroom. These results suggest that both tangible and graphical systems may have advantages for use in the classroom depending on the situation. I will conclude with a discussion of implications for future work and a list of lessons learned from the design and deployment of Tern.

1.2 Educational Philosophy

My educational philosophy falls in the constructivist tradition and has been influenced by the work of Jean Piaget, Lev Vygotsky, and Seymour Papert in particular (Richardson, 1998; Vygotsky, 1978; Papert, 1980). In designing technology-based interventions for educational settings, my goal is to provide learning activities that are intrinsically motivated. In other words, as much as possible, my goal is for children to be the principle driving force behind their own learning experiences. Thus, rather than trying to transmit knowledge in a one-way stream from the teacher to the learner—or from the multi-media computer to the child—I seek to provide interactive experiences that allow children to construct knowledge through meaningful activity. In particular, Papert's notion of *constructionism* proposes that learning is most effective when children are engaged in creating meaningful projects in the real world (Papert, 1980). In addition, Vygotsky's theory of the *zone of proximal development* (ZPD) defines a critical for educators and designers. In particular, ZPD suggests that children learn best when provided with learning experiences that are at or slightly above their current level of development (Vygotsky, 1978).

1.3 Methodological Approach: Design-Based Research

My work in both classrooms and museums has been influenced by a methodological approach known as *design-based research.* Design-based research (DBR) is an approach to studying novel tools and techniques for education in the context of real-life learning settings (Dede et al., 2004). As a methodology, DBR acknowledges the substantial limitations of conducting research in chaotic environments like classrooms; however, in exchange researchers hope that the resulting designs will be effective and practical for future use. DBR is based on an iterative process of design, evaluation, and testing. By collecting both qualitative and quantitative data in non-laboratory settings, design-based research has the additional goal of developing theories of learning that will inform future research. This approach has roots in user-centered design principals (Norman, 1986), in that the focus of the iterative design and testing process is on the end users, be they students or teachers, museum visitors or staff. My approach, however, has typically not involved *cooperative* or *participatory* methods as end users have not been directly involved in the design process. Rather, I created designs based on assumptions of the needs and capabilities of end users. I then attempted to validate those assumptions through the testing of prototypes in real world settings with representatives from my target user population.

Chapter 2

Background

2.1 Technology in Education

This dissertation concerns *educational technology*, which I define broadly as any tool used in educational settings (both formal and informal) to support learning. This includes technologies that have been designed specifically for education as well as technologies that have been appropriated by teachers and students to support learning activities. To put this in context, Figure 2.1 shows a photograph that I took of the *Learning Tools* shelf in a 4th/5th grade (ages 9–11) classroom in an urban, Title I[1] school in Boston, Massachusetts where I worked for two years as a National Science Foundation GK-12 Fellow.[2] This shelf is full of excellent examples of educational technology, including both tools that have been appropriated from other contexts for use in the classroom—clipboards, rulers, markers, crayons, etc.—as well as tools

[1]In the 2008-2009 school year 44% of the students enrolled at this school qualified for free or reduced price lunch.

[2]http://www.nsfgk12.org/

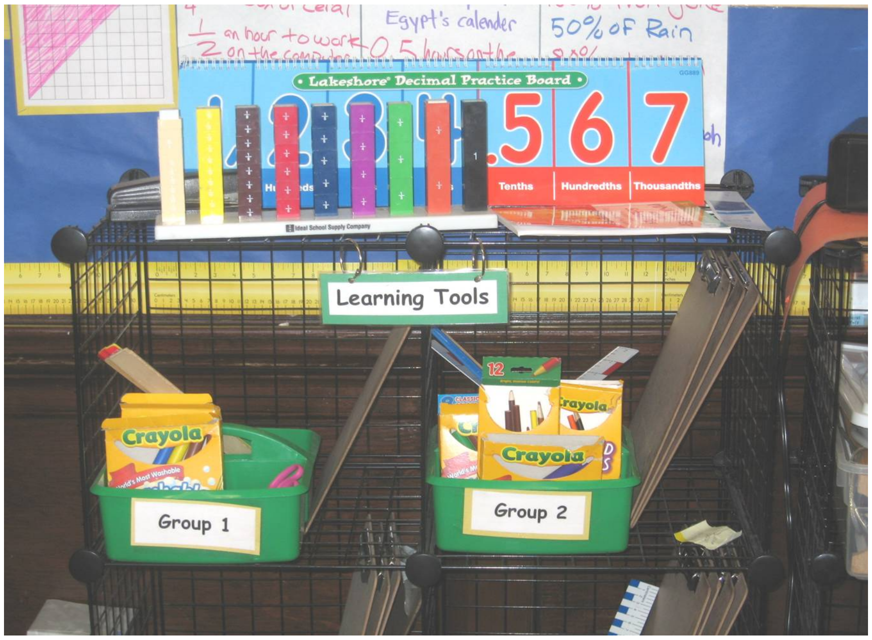


Figure 2.1: A picture of the *Learning Tools* shelf in a 4th/5th grade (ages 9–11)classroom at an urban, Title I school in Boston, Massachusetts (taken 2006).

that were specifically designed for educational use—the fraction stackers and the powers of ten flip-chart. Throughout the course of the school day, the teacher in this classroom would use this area of the room to help manage classroom dynamics. For example, he would often say things like, "Group 1, grab some clipboards and go out in the hallway and work on revising your essays; group 2, grab some map pencils and continue working on your observational drawings; Emily, grab a calculator and come over here to work on your math homework that you missed when you were out last week."[3] These tools fit comfortably within the routine of a classroom and are useful to students for learning and teachers for maintaining a positive and productive learning environment (Cuban, 1984).

The technologies available on the learning tools shelf share many common traits that make them advantageous for use in classrooms. Four traits of particular importance are cost-effectiveness, reliability, flexibility, and usefulness.

[3]These are not direct quotes from the teacher.

- **Cost-Effectiveness**: Cost-effectiveness implies that a technology is worth its investment. That is, the technology is either inexpensive (e.g. crayons), or it is useful enough that its cost can be amortized over a period of months or years (e.g. calculators).

- **Reliability**: Reliability implies that a particular technology either works well over a long period of time, requiring only minimal maintenance (e.g. clipboards and calculators), or it is easily repaired or replaced (e.g. pencils and markers).

- **Flexibility**: Related to reliability, flexibility implies that a tool can be used for a variety of tasks in a variety of contexts. Furthermore, it is readily adaptable to meet the needs of new or unexpected situations. Flexible technology can thus empower a teacher to organize the dynamics of the classroom, responding to the needs of the students and the demands of the environment in a more fluid manner. Flexibility, in many cases, also implies *portability*. That is, the tool can be easily carried from one place to another.

- **Usefulness**: Building on the previous traits, usefulness implies that a particular technology contributes to the role of the teacher or the student in a meaningful way, either as a tool to facilitate educational activities indirectly or as a tool to directly support learning. For example, overhead projectors and chalkboards indirectly support education by allowing teachers to present material to an entire class, while a math manipulative might directly support learning by allowing students to explore and reflect on a concept of interest.

Turning now to a different type of educational technology, Figure 2.2 shows a picture that I took in the same 4th/5th grade (ages 9–11) classroom in Boston. It shows a common configuration of desktop computers in an elementary school classroom—four desktop computers arranged around the periphery of the classroom to serve

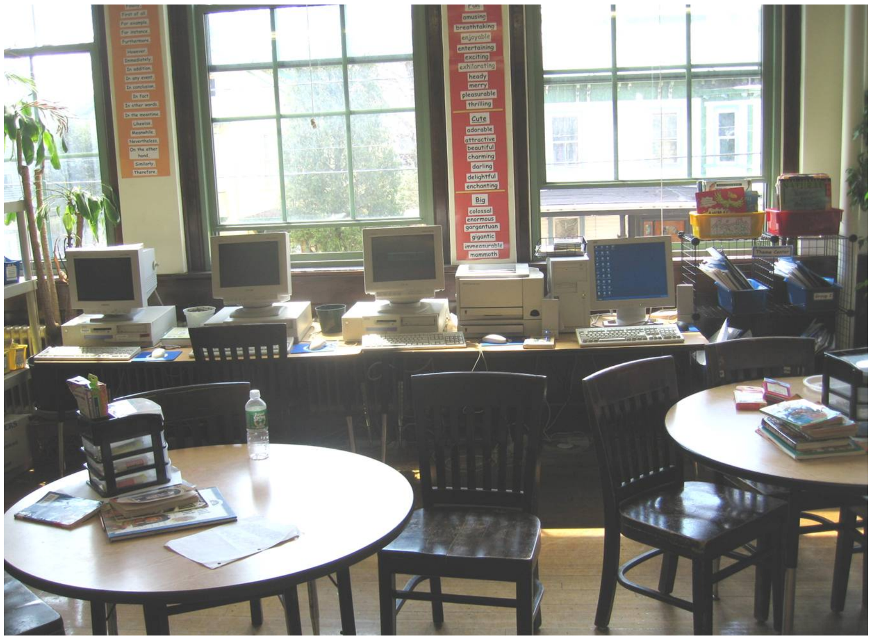


Figure 2.2: A picture showing the computer setup in the same 4th / 5th grade (ages 9–11) classroom in Boston.

the needs of around 25 students. This digital technology shares some of the traits of its counterparts from the learning tools shelf on the opposite side of the room. Namely, the computers are cost-effective, especially since older computer equipment is often donated to schools; they are reliable, although perhaps not to the degree of the learning tools from across the room; and they are flexible, in the sense that many different tasks can be accomplished with what Seymour Papert has termed the "protean machine" (Papert, 1980). But are they useful? Or, perhaps more to the point, what are the computers actually *used* for? In my experience, the answer is almost exclusively for word processing and Internet-based research, but little else.[4] And, for this small subset of tasks, the computers certainly are useful. But why aren't they used for more, especially since there is ample evidence pointing to benefits of the thoughtful inclusion of computer-based activities in classrooms (Clements, 1999b,c; Haugland, 1992; Haugland & Shade, 1994)? As the AAUW puts it, "Computers

[4]My observations seem congruent with other studies of computer use in classrooms (see Cuban, 2001).

can no longer be treated as a *set aside,* lab-based activity. Computation should be integrated across the curriculum, into such subject areas and disciplines as art, music, and literature, as well as engineering and science. This integration supports better learning for all, while it invites more girls into technology through a range of subjects that already interest them" (AAUW, 2000).

Larry Cuban argues that part of the answer to this question is that educators face what he calls *contextually constrained choices* that influence their decision to incorporate computer-based activities in the classroom (Cuban, 2001). In essence, individual teachers become gatekeepers for the kinds of technology that are present in their classroom and the ways in which those technologies are used (Cuban, 2001). Furthermore "...teachers will alter classroom behavior selectively to the degree that certain technologies help them solve problems they define as important and avoid eroding their classroom authority. They will either resist or be indifferent to changes that they see as irrelevant to their practice" (Cuban, 1984). In other words, in the real-world context of the classroom, shaped by history, culture, and day-to-day pragmatic decisions, teachers must ask themselves if the inclusion of a particular technology makes sense. Cuban therefore predicts that the incorporation of in-depth technology-based activities such as computer programming will be rare (Cuban, 1984). And, indeed, in his investigation into computer use in Silicon Valley schools he found that less than 5 percent of high school students had intense "tech-heavy" experiences, and that less than 5 percent of teachers integrated computer technology into their regular curriculum and instructional routines (Cuban, 2001). And, while my personal experience in classrooms is limited, this assessment seems in line with what I have observed in public schools in Bostons.

In particular, Cuban's concept of contextually constrained choice helps explain some of my observations working in schools as a GK–12 fellow. While I was working in the 4th/5th grade (ages 9–11) classrooms, the school used an elementary mathematics curriculum called TERC investigations. This curriculum includes a unit called *Picturing Polygons* which, in turn, includes a set of computer-based activities based on a variant of the Logo programming language (Papert, 1980). These activities were designed to allow students to explore concepts of geometry by writing short computer programs to draw polygons by moving the Logo *turtle* on a grid. This seemed to me to be an excellent example of a computer-based activity thoughtfully integrated into the larger classroom curriculum.

However, despite access to the TERC software, computers, and well-designed curriculum, the teacher chose not to implement the Logo activities. *Why not?* It's not that the teacher was opposed to using computers or even computer programming in the classroom—on the contrary, he is technologically savvy, and, in general, has a positive view of the role of technology in the classroom. Cuban's concept of contextually constrained choice perhaps offers a better explanation. He was a first year teacher who often confided that he was overwhelmed by the demands of the job; he had fifty minutes every day to teach mathematics to around 25 children; and, at the time, there were four working desktop computers available in the classroom. To make the Logo activity work, he would have had to have divided the children into small groups and rotated them between computer-based and non-computer-based work. The groups of students, in turn, would have had to crowd around the desktop computers and figure out how to share the single-user input devices (mouse and keyboard). And, while I believe this was within this teacher's capacity as a manager of classroom activities, the effort might have seemed more trouble than it was worth.

From a human-computer interaction perspective, part of the problem has to do with the desktop computers themselves. Primarily productivity tools for single users in business settings, modern desktop computers are in many ways poorly designed for classroom use (Cuban, 2001). For one thing, many young children lack the fine motor skills necessary to effectively use mouse-based interfaces (Hourcade et al., 2004), which is an important consideration when conducting programming activities in preschool and kindergarten classrooms (ages 4–6). Furthermore, group work is often the norm in both formal and informal learning settings. And, several research studies have demonstrated that overcoming the inherent single-user limitation of desktop computers can improve children's experiences with technology (Stewart et al., 1998; Scott et al., 2000; Inkpen et al., 1995b,a). For example, in a pilot study conducted with 72 elementary school children, Stewart, Raybourn, Bederson, & Druin (1998) observed the behavior of children collaborating on a computer-based activity with a single user input device (a computer mouse). They found that with a single mouse children would frequently fight for control of the input device and that, in general, their collaborative communication was poor. Furthermore, the passive collaborator (the child without the mouse) would often exhibit signs of frustration and lack of attention. Similarly, Scott, Mandryk, & Inkpen (2003) conducted a study with 40 elementary school children (ages 9–11) comparing children working in pairs on a puzzle game with both a single-mouse and a multiple-mouse system. They found that, "quantitative and qualitative analysis of computer log files, videotapes, and questionnaires revealed that providing children with technology that supports concurrent, multi-user interaction can positively impact their engagement, participation, and enjoyment of the activity."

2.2 Computer Programming and Education

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every childs analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

—Wing (2006)

Since the 1960s a large number of programming languages and systems targeted at novice users have been created (Kelleher & Pausch, 2005). Logo (Papert, 1980) is one of the earliest and perhaps the most influential educational programming language, and, the *constructionist* theories of its central proponent, Seymour Papert, have shaped the field of educational computer programming ever since. Notable recent languages include PicoBlocks, Scratch (Resnick, 2007), Alice (Conway et al., 1994), and ROBOLAB. Much of the effort to create these languages has been motivated by the belief that learning how to program is not only necessary for *technological fluency* (Bers, 2008; AAUW, 2000) in the digital age, but that it is also somehow beneficial as an academic endeavor in its own right (Papert, 1980). In other words, that *computational thinking* (Wing, 2006), as a more general abstraction of computer programming, is a powerful intellectual skill that can have a positive impact on other areas of children's intellectual growth. Indeed, some research has indicated that learning how to program computers can have a positive and measurable effect on children's achievement, not only in math and science, but also in language skills, creativity, and social-emotional interaction (Bers, 2008; Clements, 1999b; Haugland, 1992). Of course, the decades of research involving computer programming in schools is diverse, and much depends on the age of the students, the context in which the computer programming activities are introduced, and the ways in which the activities are integrated within the broader curriculum (Clements, 1999b; AAUW, 2000). As the American Association of University Women (AAUW) Educational Foundation puts it, "Fluency is best acquired when students do coherent, ongoing projects to achieve specific goals in subjects that are relevant and interesting to them" (AAUW, 2000). If children are merely *exposed* to programming there seems to be little measurable benefit (Clements, 1999b; Rader et al., 1999). Again, the AAUW emphasizes the need to integrate computer science throughout the curriculum, "have computer science go beyond programming to emphasize how computer science (including programming) is used to solve real-life problems" (AAUW, 2000).

One barrier to this vision is that computer programming is difficult for novices of any age. Kelleher & Pausch (2005) offer a taxonomy containing well over 50 novice programming systems, a great number of which aim to ease or eliminate the process of learning language syntax, perhaps the most often cited source of novice frustration. Beyond syntax, there are many specific conceptual hurdles faced by novice programmers as well as fundamental misconceptions about the nature of computers and computer programming (Ben-Ari, 1998). According to Norman (1986), the primary problem facing novice programmers is the gap between the representation the brain uses when thinking about a problem and the representation a computer will accept. Likewise, Levy and Mioduser present a study in which they detail children's struggles to construct technical explanations for robot behaviors, presenting evidence of Norman's gap from the opposite direction (Levy & Mioduser, 2008). In other words, the children were struggling to formulate technical language to describe the actions of a robot that the researchers had programmed ahead of time.

Part of this dissertation will describe the use of tangible programming languages by kindergarten-aged children (ages 5–6). Therefore, in addition to the above challenges faced by novice programmers, we must also consider the developmental needs and capabilities of young children. McKeithen, Reitman, Rueter, & Hirtle (1981) conducted a study that explored the differences in the ability of expert and novice computer programmers to recall details of computer programs. In their analysis, they theorize that because novice programmers lack adequate mental models for programming tasks, they rely on rich common language associations for these concepts. For example, computer words like LOOP, FOR, STRING, and CASE have very different common language meanings. Reflecting on this result, it seems reasonable to expect that young children will have a more difficult time building conceptual models for programming concepts because they have fewer mental schemas on which to build.

Likewise Rader, Brand, & Lewis (1999) conducted a study with the Apple's KidSim programming system in which 2nd/3rd graders (ages 7–9) and 4th/5th (ages 9–11) graders used the system for one year with minimal structured instruction. At the end of the year, the younger children had significantly more difficulty with programming concepts such as individual actions, rule order, and subroutine. However, the authors suggest that had the children received a structured introduction to programming concepts, they would have developed a much better understanding of the system.

Despite these concerns, previous research has shown that even children as young as four years old can understand the basic concepts of computer programming and can build and program simple robotics projects (Bers, 2008; Cejka et al., 2006; Bers et al., 2006). Furthermore, early studies with the text-based language, Logo, have shown that computer programming, when introduced in a structured way, can help young children with variety of cognitive skills, including basic number sense, language skills, and visual memory (Clements, 1999b).

For example, Clements & Gullo (1984) conducted a study comparing six-year-old children who spent twelve weeks working with either computer programming or computer-aided instruction. The computer programming group performed significantly better on post-tests measuring certain aspects of reflectivity, divergent thinking, and meta-cognitive abilities. The interesting aspect of this study was that six-year-olds, with adult help, could write text-based Logo programs and demonstrated at least short-term gains in some cognitive abilities. Clements and Gullo also proposed the hypothesis, originally attributed to Papert (1980), that "if computer programming can allow children to master ideas formerly though too abstract for their developmental level, it may accelerate cognitive development including operational competence." They tested this hypothesis using an operational competence instrument based on experiments of Piaget and Inhelder but found no significant difference between the groups.

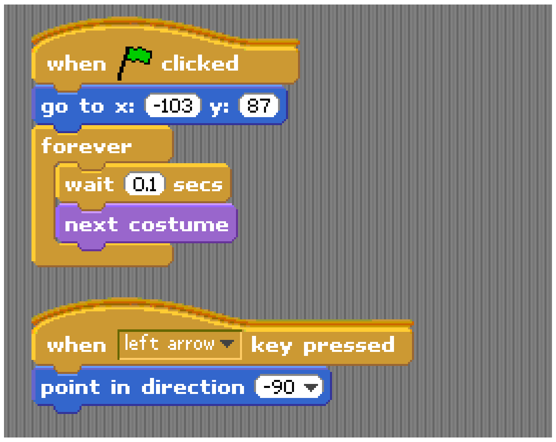
In a more recent example, Levy & Mioduser (2008) present a study in which six kindergarten children (ages 5–6) used a simple programming environment to complete a sequence of description and programming tasks with an autonomous robot. In describing the behavior of autonomous robots to researchers, the children employed two distinct modes of explanations—engineering (viewing the robot in terms of technical systems) and bridging (combining a technological and psychological perspective). The children used psychological perspective as a way to transition into a more technical understanding of the robot's behavior. An analysis of the children's dialog with researchers revealed surprisingly sophisticated understandings of the underlying concepts:

> On their own, the children were capable of deciphering the simpler robots behavior. However, with further support their abilities were augmented. They appropriated the offered tools, thinking in "concrete-abstractions" about the robot's behavior and relating larger functional information to the intricate causal technological underpinnings. Furthermore, they could use these tools to construct desired robot behaviors (Levy & Mioduser, 2008).

Levy and Mioduser conclude that:

> In light of the encouraging results of our exploratory studies, we believe that the interaction with knowledge-embedded artifacts in a supportive and playful classroom, represent clear opportunities for the children's intellectual growth and development.

However, regardless of children's intellectual capabilities with respect to computer programming, most current programming environments are poorly suited for young children. One problem is that the syntax of text-based computer languages, such as Logo, can be unintuitive and frustrating for novice programmers. This is exacerbated for young children who are still learning how to read and write. Modern visual programming languages such as Scratch or ROBOLAB allow children to program by dragging and connecting icons on the computer screen. Many recent languages have also adopted a puzzle piece metaphor, whereby programs are constructed by connecting interlocking visual elements. Tern follows this metaphor as well, although in our case the puzzle pieces are physical blocks rather than icons on a screen. And, while this approach simplifies language syntax, the interfaces require young children to use a mouse to navigate hierarchical menus, click on icons, and drag lines to very



```
when [flag] clicked
go to x: -103 y: 87
forever
    wait 0.1 secs
    next costume

when left arrow key pressed
point in direction -90
```

Figure 2.3: A screen shot from the Scratch programming language from the Lifelong Kindergarten group at the MIT Media Lab. Scratch is one of several recent educational programming languages to adopt a puzzle piece graphical metaphor.

small target areas on a computer screen. All of this requires fine motor skills that make it difficult for many young children to participate (Hourcade et al., 2004). As a result, adults often have to sit with young children and give click-by-click instructions to make programming possible, which poses challenges for children's learning (Beals & Bers, 2006). It also makes it difficult to implement computer programming in average schools, where there are often only one or two adults for twenty to thirty children. And, from a research perspective, it makes it difficult to understand what children can accomplish both with and without direct adult help. Attempts have been made to create simpler versions of these languages. However, the resulting interfaces can obscure some of the most important aspects of programming, such as the notion of creating a sequence of commands to form a program's flow-of-control.

2.3 Robotics, Programming, and Young Children

"Computer technology" is a broad term that can mean many things, especially in the context of a classroom. Robotics is one type of educational computer technology that has received much recent attention. Bers (2008) argues for the potential of robotics in early childhood classrooms as a path to technological fluency and as an integrator of curricular content areas. She argues that in early childhood classrooms, content areas tend not to be isolated, but integrated more broadly into classroom curriculum that encompasses different content and skills; learning can be project-driven and open-ended; and student work does not have to fit into an hour-long class period.

> While using these tools, children learn about sensors, motors, and the digital domain in a playful way by building their own cars that follow a light, elevators that work with a touch sensor, or puppets that play music. Young children can become engineers by playing with gears, levers, motors, sensors, and programming loops, and they can become storytellers by creating their own meaningful projects that move in response to a stimulus (either another robot or the environment... Robotic manipulatives are a gateway for helping children learn about mathematical concepts and the scientific method of inquiry. (Bers, 2008)

Robotics, however, is about more than just creating physical artifacts. In order to bring robots to "life" children must also create computer programs—digital artifacts that allow robots to move, blink, sing, and respond to their environment. One goal of the Tern project is to give these digital artifacts a physical presence in the classroom. Since the process of constructing programs can now be situated in the classroom at large—on children's desks or on the floor—children's programming work can be more open and visible and can become more a part of presentations and discussions of technology projects. Likewise physical programming elements can be incorporated into whole-class instruction activities without the need for a large computer or a large monitor that the entire classroom can see.

2.4 Tangible Interfaces for Education

Building on foundations of ubiquitous computing and augmented reality, Ishii and Ullmer describe their vision of tangible user interfaces (TUIs) as systems that "will augment the real physical world by coupling digital information to everyday physical objects and environments" (Ishii & Ullmer, 1997) And, over the past decade, research in tangible user interfaces (TUIs) has expanded our definition of what it means to interact with computers. Much of the research conducted with tangible user interfaces has focused on education (O'Malley & Fraser, 2004). Perhaps this is not surprising given that many of the benefits of moving interfaces into the physical world seem especially beneficial for children in the classroom.

In the *Reality Based Interaction* framework, we propose that tangible interfaces have potential advantages for users because they build directly on existing knowledge and experience from the real world such as an understanding of naive physics, body awareness and skills, and social awareness and skills (Jacob et al., 2008). This seems advantageous for education because it de-emphasizes the process of learning how to manipulate an interface and focuses instead on the concepts to be learned. As an example, a potential advantage of tangible programming languages is that they can encode syntax in the physical form of the objects that make up the language, thus decreasing syntax that children must learn before they can create computer programs on their own. Tern, for example, uses a jigsaw puzzle metaphor at the Robot Park exhibit. The blocks thus *afford* (Norman, 1986) chaining together in a linear sequence. It is important to note that graphical languages can do this as well through the use of visual metaphors, and our work in kindergarten classrooms (ages 5–6) suggests that the graphical metaphors may be just as easy for children to understand as physical metaphors. Indeed, Marshall points out a general lack of empirical evidence supporting value of tangible interfaces for educational use compared to standard computer interfaces (Marshall, 2007).

Marshall, Price, & Rogers (2003) propose two main classes of tangible systems to support learning: expressive and exploratory. Expressive tangible systems allow learners to create their own external representations of a concept or an activity, while exploratory tangible systems allow learners to explore a model presented by someone else. Marshall et al. go on to propose that productive learning results from a cycle of alternately attending to a task at hand and reflecting on the tangible object used to accomplish the task (i.e. ready-at-hand and present-at-hand) (Marshall et al., 2003). Marshall (2007) expands this framework with six perspectives on learning with TUIs. Although cautioning that little comparative work has been done, Marshall posits (among other things) that tangibles might engage children in playful learning and that novel links between physical and digital systems might increase engagement and reflection.

In this vein, Fails, Druin, Guha, Chipman, Simms, & Churaman (2005) compared the use of desktop and physical environments for preschool-aged children in the context of a "content-infused story" game designed to teach about environmental health hazards. The game used a collection of props (either physical or virtual depending on the condition) and audio clips. A qualitative evaluation with 16 children found that "the physical environment to have several advantages over the desktop environment. These were interest, engagement, and understanding [...] This suggests that embedding technology in the physical world, rather than simply presenting them with traditional desktop applications may be beneficial to young children."

Likewise, Rogers, Scaife, Gabrielli, Smith, & Harris (2002) explored the coupling of digital and physical inputs and outputs in an educational color-mixing game for children called *Chromarium*. Here the author's compared each of the four possible combinations of digital to physical couplings (i.e. digital $\rightarrow$ physical, digital $\rightarrow$ digital, physical $\rightarrow$ digital, and physical $\rightarrow$ physical). They discovered that the unfamiliar coupling of physical input to digital output was highly effective at producing reflection, collaboration, and exploration on the part of children. In a sense, Tern follows this physical $\rightarrow$ digital transformation. While, strictly speaking, the physical manipulation of blocks in Tern results in physical output (motion of the robot), the output is greatly mediated by a digital system. Moreover, this digital system is designed to be highly visible to users.

In designing novel couplings of physical and digital systems, the work of Mitchel Resnick and the *Technologies for Lifelong Kindergarten* group at the MIT Media Lab is notable for its focus on *digital manipulatives*, or educational manipulative augmented with digital technology. For example, BitBall (Resnick et al., 1998) allows children to explore concepts of motion and acceleration with a rubber ball with an embedded microcontroller and colored LEDs. Using a Logo-like language, children can program the ball to respond in different ways to motion. Zuckerman, Arida, & Resnick (2005) also created two digital manipulative systems called *FlowBlocks* and *SystemBlocks*, both of which model abstract system dynamics concepts. Testing these systems with children, ages 4–11, Zuckerman et al. conclude that, "our findings suggest that [these systems] are engaging for children, and successfully introduce specific concepts such as rate, accumulation, feedback, and probability to different age groups."

Ananny and Cassell (2002) created TellTale, a tangible toy designed to support children's language development. Presented as a toy caterpillar with five body segments, TellTale allows children to record 20-second audio clips into each body segment and then to rearrange those segments in any order. By attaching the caterpillar's head to the body, children can hear their entire story played back to them. TellTale shares attributes of tangible programming systems in that children can combine components to create sequences of actions that are then executed in order. And, indeed, Ananny and Cassell make the hypothesis that, "by carefully designing technology-enhanced language toys that give children control over both the structure and content of their language, young children may be able to engage in literacy activities previously thought to be too advanced for their age."

Mazalek, Davenport, & Ishii (2002) take storytelling using tangible interfaces in a different direction with a system called Tangible Viewpoints. Tangible Viewpoints combines tokens (representing characters and points-of-view) with an augmented surface to create an *interactive narrative.* By placing a token on the surface, users can see the story segments associated with that character projected around it. Users can then manipulate a lens-like object to selection story content to view in more detail. Furthermore, when two tokens are touched together, users see only the story segments that involve both of the represented characters. Mazelek et al. tested this system over a period of 10 days in a informal *clubhouse* learning setting.

Raffle, Parkes, & Ishii (2004) produced Topobo, a construction kit that has been actuated to produce kinetic memory. Using Topobo, children can construct imaginative creatures composed of passive and active building components. The active components have the ability to record and play back physical motion, allowing children to learn about concepts such as animal movement and their own bodies in the process.

Chipman, Druin, Beer, Fails, Guha, & Simms (2006) created a system called *tangible flags* that allows children to create a share "digital knowledge artifacts" in physical environments–such as on field trips. The system allows children to flag objects of interest in the physical environment and then add digital notes about those objects using a tablet computer. Other children can then access and annotate these digital artifacts as they explore the physical space. The researchers evaluated this system with visitors at a national park and concluded, among other things, that the children in their study were naturally able to use tangible interaction to access digital information.

2.5 Tangible Programming Languages

The idea of tangible programming was first introduced in the mid-1970's by Radia Perlman, then a researcher at the MIT Logo Lab. Perlman believed that the syntax rules of text-based computer languages represented a serious barrier to learning for young children. To address this issue she developed an interface called Slot Machines (Perlman, 1976) that allowed young children to insert cards representing various Logo commands into three colored racks, which in turn represented subroutines. The idea of tangible programming was revived nearly two decades later with projects such as Madea and McGee's *solid programming* (Maeda & McGee, 1993) and Suzuki and Kato's *AlgoBlocks* (Suzuki & Kato, 1995). With solid programming, physical instantiations of conditional logic elements (AND, OR, and NOT gates) can be embedded with sensors and motors in a robotic agent to describe responses to various input conditions. With AlgoBlocks children create Logo-like programs using interlocking aluminum blocks with embedded electronic components to control a virtual environment on a computer screen. LEDs in the aluminum blocks light up as the execution of a program passes through each successive command. This allows physical programs to both demonstrate their flow-of-control and to provide for a rudimentary tangible debugger. More recently, a variety of tangible programming languages have

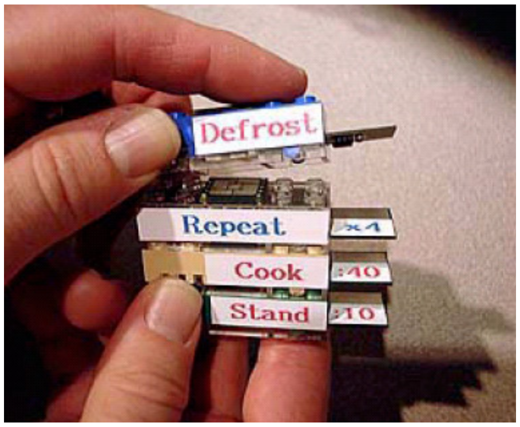


Figure 2.4: Tim McNerney's Tangible Computation Bricks (McNerney, 2000) features a LEGO-like construction kits with embedded microprocessors that allow children to experiment with programming concepts in real-time.

been created. Some involve program by example systems such as StoryKits (Montemayor et al., 2002), Topobo (Parkes et al., 2008), and Curlybot (Frei et al., 2000). Others fall more into Ullmer's *constructive assembly* classification for tangible interfaces (Ullmer, 2002). Frei, Su, Mikhak, & Ishii (2000) created *Curlybot*, an example of a program-by-demonstration system. With Curlybot children can *program* a robot by dragging it on the floor to demonstrate its motion. An LED on the robot indicates its mode: red for record, and green for playback. In playback mode, the robot repeats its recorded motion indefinitely. Tangible Computation Bricks (McNerney, 2000), on the other hand, feature LEGO bricks with embedded Cricket (Martin et al., 2000) microprocessors. McNerney described several types of tangible programming languages that could be expressed with the bricks. The bricks also accept a single parameter card which can interchangeably be a constant, a timer, a sensor, or some user-adjustable value. Likewise, Blackwell, Hague, & Greaves (2001) at the University of Cambridge developed *Media Cubes*, tangible programming elements for controlling networks of consumer electronic devices. Media Cubes are blocks with bidirectional, infra-red communication capabilities. Induction coils embedded in the cubes also allow for the detection of adjacency with other cubes.

Researchers have also begun to explore the exciting potentials of programming in and with the physical world. Some ideas that have been generated include the blending of physical space and digital programming (Fernaeus & Tholander, 2006; Montemayor et al., 2002), robots that are also embodied algorithmic structures (Schweikardt & Gross, 2008; Wyeth, 2008), the incorporation of found or crafted materials into algorithmic expressions (Smith, 2008), or the integration of physical activity and play with programming (Smith, 2007; Scharf et al., 2008). Scharf, Winkler, and Herczeg designed a programming system called Tangicons that builds on the TopCodes computer vision library to create a game in which teams of kindergarten children (ages 5–6) combine physical exercise with simple programming activities (Scharf et al., 2008). Work on tangible programming languages has often focused on young chil-

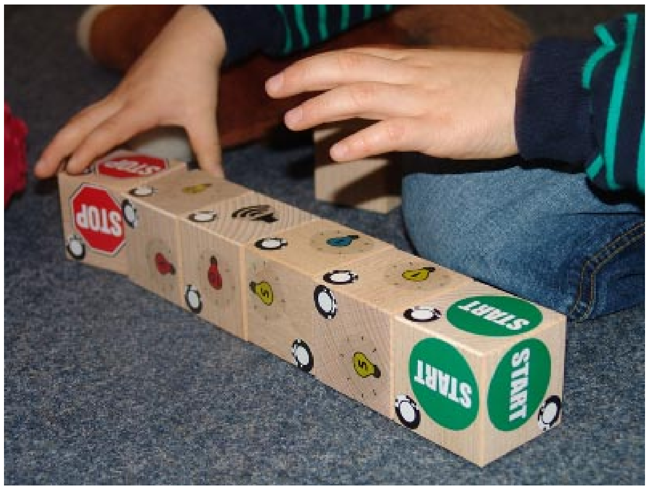


Figure 2.5: An early prototype of the Tangicons (Scharf et al., 2008) system from the University of Lubeck which uses my TopCode library to create a game for kindergarten children that combines physical exercise with simple programming activities.

dren. For example, Montemayor et al. created a prototype system called StoryKits that allowed children aged 4–6 to explore and program an interactive environment called (*StoryRoom*) (Montemayor et al., 2002). In this work, Montemayor et al., in collaboration with *children as design partners* (Druin, 1999), rejected the idea of using a visual programming language to control a physical environment. They found that children had difficulty conceptually connecting what was on the screen with what they used in the physical room. In this process, they developed a set of physical icons to program the room by demonstration including items such as a magic wand to enter authoring mode, a hand to make an object touch sensitive, and a light to make an object light up (see Figure 2.6). In evaluating their prototype, they found that children could easily comprehend and participate in stories set in physically interactive environments. However, Montemayor et al. also concluded that while most children could understand relationships between physical icons and the interactive props around the room, they had some trouble understanding the difference between programming and participation in an existing story. Another example is the Elec-

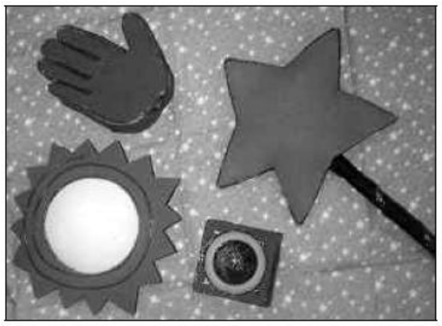


Figure 2.6: Montemayor et al. developed a set of physical icons called StoryKits to program an interactive physical room for storytelling.

tronic Blocks language created by Wyeth & Purchase (2002). This language consists of sensor blocks for detecting light, sound, and touch; logic blocks containing *and*, *negate*, *delay*, and *toggle* features; and action blocks for generating light, sound, and motion. The physical affordances of the blocks enforce simple syntax rules such as stacking blocks to chain input and output elements. When children create stacks of these blocks, they can experiment with sensors, actuators, and logic components in a free form, interactive way. Wyeth conducted a recent study to determine how twelve 7- and 8-year-old children were able to learn different aspect of programming through interaction with the blocks while trying to complete 20 programming tasks of varying complexity (Wyeth, 2008). Wyeth concludes that, "children developed a fundamental understanding of programming—that inputs affect outputs and that output behavior is reliant on specific combinations of input instructions. The input-output relationships of the blocks are not directly visible; these conceptual relationships were discovered by study participants through exploration and observation. *Tangible activity enabled the successful learning of simple abstract programming concepts.*" (italics added).

While all of this work on tangible programming is pioneering, there is notable lack of evidence that tangible systems offer any benefits compared to onscreen counterparts. Wyeth's conclusion that the tangibility of her language enabled successful learning seems overly optimistic given the scope of her study. And, the question of whether children would have had similar success with a comparable graphical language is never addressed. In general, while some of these systems provide unique programming experiences that have no reasonable graphical comparison, many could be compared to onscreen systems through controlled experiments. Even though the results of such experiments are always difficult to interpret when it comes to the use of technology in real life educational settings, knowledge gained from such research might offer useful directions for tangible programming as well as the field of tangible interaction as a whole.

2.6 Passive Tangible Interfaces

Tangible interfaces have been cited as being well-suited for use in educational environments such as classrooms and science museums (Hornecker & Stifter, 2006), but there are potential problems involving cost and reliability that come in to play when incorporating cutting-edge technology outside of laboratory settings. This is not to say that tangible interfaces cannot be reliable and inexpensive; however, designers must make careful tradeoffs between the type of interaction desired and the reliability and expense of the technology needed to support that interaction. One such tradeoff involves the closeness of the coupling between digital and physical aspects of a tangible system. In the framework for tangible interaction advanced by Hornecker & Buur (2006), this property is called *Isomorph Effect*—how closely coupled in space and time physical actions are to digital responses. Some examples of systems

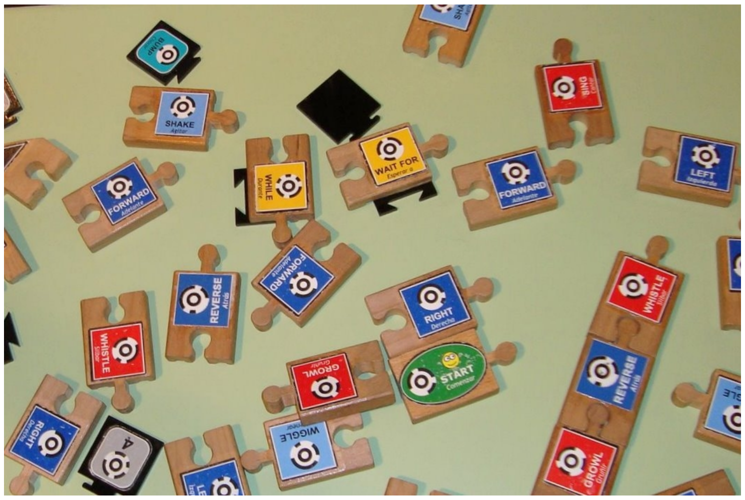


Figure 2.7: Our tangible programming interface, Tern, consists of a collection of wooden blocks shaped like jigsaw puzzle pieces.

with close coupling are FlowBlocks (Zuckerman et al., 2006), Tangible Programming Bricks (McNerney, 2004), and Electronic Blocks (Wyeth, 2008). In these systems, electronic components are embedded in the physical elements of the interface. As blocks are assembled physically, the digital response is immediate, both in time and space—lights blink and motors move. Ideally, this supports playful and exploratory interaction, well-suited for informal experimentation and learning. The downside is that each element of the system requires embedded electronics and a connection to a power supply, potentially adding cost and decreasing reliability.

On the opposite end of this tradeoff are systems with weak isomorph effects. For such systems, the digital response to physical manipulation is remote in time and/or space, but there are substantial potential advantages in terms of robustness, durability, and cost. I have proposed the term *passive tangible interface* (Horn et al., 2008) to describe these types of systems. Passive tangible interfaces consist of a collection of unpowered physical components with, in many cases, a non-continuous link to a digital system. The physical components can be inexpensive to produce and make use of passive sensor technology like computer vision fiducials or RFID tags. For this reason, passive tangible systems may give interaction designers greater freedom to choose materials and forms that make sense for an application rather than the technology used to implement it. Some examples of passive tangible interfaces include Smith's GameBlocks programming language (Smith, 2007) and our Tern system. With these interfaces, it is not essential for a user's physical actions (i.e. moving, connecting, and disconnecting blocks) to be tracked in real time. For example, in the case of Tern, it is only when a user compiles his or her program that a connection between the physical and digital is made. For Tern, the choice of a passive tangible interface has two advantages. First, it allows the use of inexpensive and durable interaction objects like wooden blocks. Second, by decoupling physical actions and digital responses, we hoped to introduce an opportunity for visitors to reflect on and discuss the outcome of their program design, ideally reinforcing the learning process. In other words, we hoped to implicitly enforce a workflow that involves design, testing, reflection, and revision. In this sense, it is important to emphasize that it is the technology, not the users, that are passive. As Resnick points out, all too often, it is the other way around (Resnick et al., 1996).

2.7 Tangible Programming and Educational Manipulatives

Much of the research dealing with educational manipulatives has focused on mathematics instruction. Sowell, for example, conducted a meta analysis of the results of 60 studies concerning the use of manipulative materials in mathematics education (Sowell, 1989). While the results of these studies were mixed, Sowell found that for treatment lengths of a year or more there were significant advantages for the use of *concrete* manipulatives in terms of attitude and acquisition for elementary grades (ages 5–11) compared to more *abstract* instruction—Sowell uses the word "abstract" to refer to instruction without manipulative materials including pencil and paper work, reading, and lecture. She cautions, however, that there are still substantial open questions about the situations and concepts for which manipulatives are best suited, and that a *one size fits all* mentality should be avoided.

Zuckerman, Arida, & Resnick (2005) offer a classification of educational manipulatives into two broad categories, "Froebel-inspired Manipulatives" (FiMs) and "Montessori-inspired Manipulatives" (MiMs). FiMs are typically construction kits (e.g. LEGO bricks), that allow children to build structures and models that represent real-world things. MiMs, on the other hand are physical manipulatives that attempt to model concepts or ideas (e.g. Cuisenaire rods to model numerical proportion). In their discussion of Montessori-inspired manipulatives, Zuckerman et al. go on to argue that if we consider tangible interfaces as an extension of educational manipulatives, they offer advantages for learning. The advantages of these *digital manipulatives* include things such as multi-sensory engagement, improved accessibility (for young children or children with learning disabilities), and better support for group learning through multi-hand interaction. Moreover, digital manipulatives can better model temporal changes or computational processes in a way that is impossible with static, non-interactive manipulatives.

Clements provides a different perspective on this research, pointing out that the use of the word “concrete” (as in concrete versus abstract) may be misleading (Clements, 1999a). He draws a distinction between *sensory concrete* knowledge and *integrated concrete* knowledge. The former refers to thinking through and with the use of concrete objects: “at early stages, children cannot count, add, or subtract meaningfully unless they have actual [physical] things” (Clements, 1999a). The latter, meanwhile, refers to knowledge that we construct as we learn and that is well *connected* to other concepts and situations. In a sense, this kind of knowledge is what is commonly referred to as “abstract.” In other words, concepts that are meaningful beyond specific settings and situations and that can be applied to a variety of domains. “Good manipulatives are those that aid students in building, strengthening, and connecting various representations of mathematical ideas” (Clements, 1999a). Moreover, Clements proposes a general definition of concrete manipulative that goes beyond physical objects to include other materials and representations, including objects which may be manipulated on a computer screen.

Uttal, Scudder, & DeLoache (1997) also argue for a new perspective on educational manipulatives. They propose that while “educators have concluded that manipulatives are useful because they are concrete and hence do not require children to reason abstractly or symbolically,” it would be more productive to think of manipulatives as symbol systems that children must learn to relate to underlying concepts. “To learn from manipulatives, children must comprehend how the manipulative represents a concept or written symbol. Concrete objects can be an effective aid in the mathematics classroom, but to use them effectively, teachers must take into account how children do (or do not) understand symbolic relations.”

Uttal et al. (1997) also advance a dual representation hypothesis, which states that children will either attend to a manipulative as an object in its own right or as a representation of something else. Furthermore, the more a child attends to the object itself, the less they will make the connection to the concept that it is intended to represent (and vice verse). Thus, Uttal et al. in part conclude that attractive or intricate physical manipulatives may be counter–productive because children will be more likely to focus on the properties of the object itself rather than the concept to be learned. Zuckerman et al. agree with assessment, arguing that MiMs should “maintain a high level of abstraction of the constructed simulations and structures, so concreteness would come from a childs analogies rather than a structures visual form” (Zuckerman et al., 2006). Furthermore, they encourage building systems that provide a way for children concretize their meanings—for example, providing a way for children to write notes on blocks to annotate their meanings.

With this context, I propose that Tern is not an educational manipulative in the traditional sense—it is not a physical object that somehow embodies “abstract” concepts of programming, thereby allowing children to think more “concretely”. Rather, I propose that it is more productive to think of Tern in the sense of Uttal, Scudder, and DeLoache’s symbol system definition. That is, Tern, is a collection of symbols that can be manipulated according to syntactic rules to produce computation. If one accepts this definition, then the dual representation hypothesis might apply as well. That is, there is a danger that if the blocks are too attractive or interesting, then children might only focus on the superficial qualities of the blocks. Indeed, in observing children use Tern, I have seen evidence to support this. When children first encounter Tern, their actions often seem purely playful—they will make the longest chain of blocks they can, or they will build a tower, or they will twirl a cube on the end of a finger. However, because Tern is a working programming language, it is more than a symbol system. By manipulating the blocks, children can create sequences of instructions that are executed by a computer, and, in this sense, Tern is a tool as well. Thus, Marshall et al. (2003) proposal that productive learning results from a cycle of alternately attending to a task at hand and reflecting on the tangible object used to accomplish the task puts the dual representation hypothesis in a more positive light. Perhaps in playing with the blocks children are entering the first phase of this cycle. More than that, though, perhaps through play children are building a positive emotional relationship with the technology that will not only make future learning experiences more enjoyable, but might also make that learning more effective. While speculative, this line of thinking might be a potential area for future research.

Chapter 3

The Tern Programming Language

In this chapter I describe the Tern tangible programming language, including both its physical design and language specification. I also describe the evolution of the language over time as a result of an iterative development process. In the next chapter, I will provide implementation details, including a description of the computer vision techniques, the tangible compiler, and the runtime interpreter.

3.1 Tern Origins

My interest in tangible programming began in 2004 when I started work on a programming language called Quetzal (Horn, 2006; Horn & Jacob, 2007). My original goal with the Quetzal project was to develop a non-working tangible programming language that could be tested in educational settings. However, I soon moved on to the idea of using computer vision techniques to implement low-cost functional prototypes that would allow me to test ideas in real educational settings. Ultimately, many features of the Tern language evolved directly from iterative design and testing with Quetzal.

I constructed the earliest Quetzal prototypes out of materials such as wood and foam core to test the basic image processing techniques and physical language designs (Figure 3.1). My first attempt to create a fully functional prototype involved using

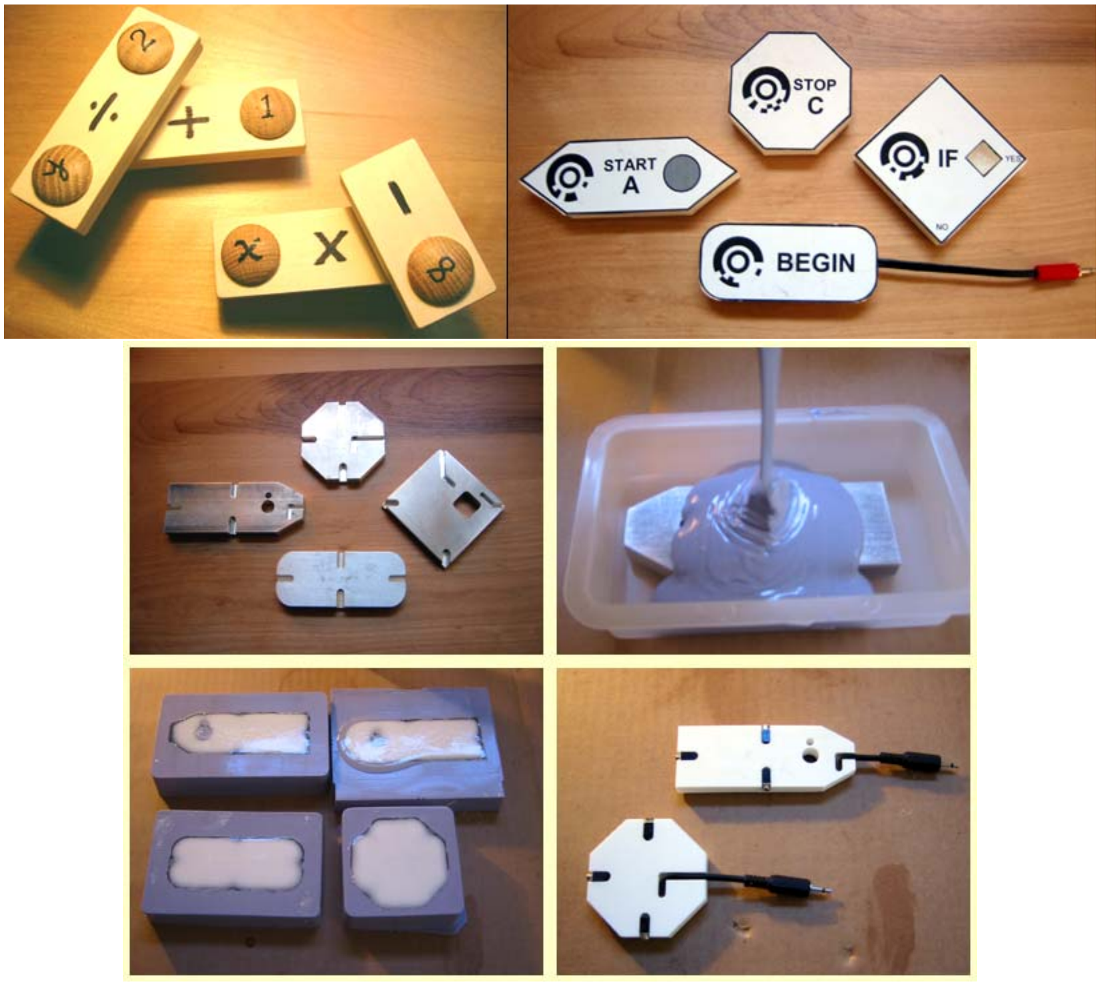


Figure 3.1: Early prototypes constructed of wood (top left) and foam core (top right). The bottom four pictures illustrate the process of creating molded urethane parts from machined aluminum originals.

molded urethane parts based on machined aluminum originals (Figure 3.1). These parts were designed to be connected together into chains of blocks using 2.5mm audio jacks and plugs. Unfortunately, this design proved to be expensive and time consuming to fabricate. Worse, because the individual statements could be connected together using audio plugs on the end of flexible wires, the resulting image processing techniques were unreliable.[1]

To solve these problems, I built the next Quetzal prototype using flat tiles laser cut from a sheet of extruded acrylic. Instead of 2.5mm audio jacks and flexible wire, I allowed statements to be chained together using rigid connection arms and pivot joints. I then glued printed paper faceplates showing both a human-readable statement name and a computer vision fiducial on to the top surface of the tiles (Figure 3.2).

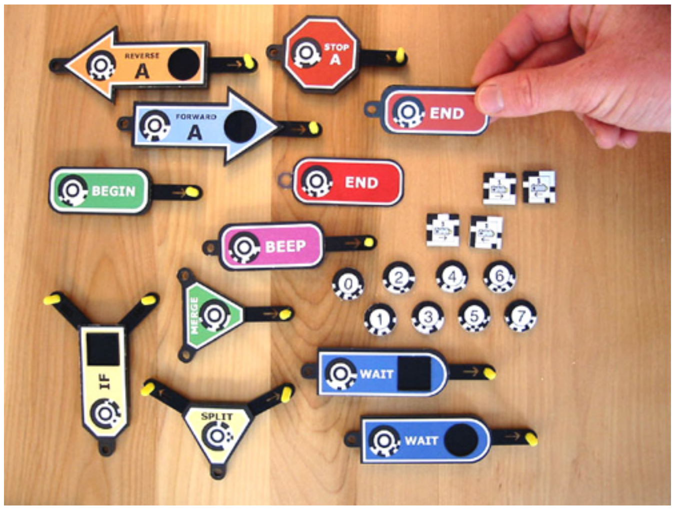


Figure 3.2: The final Quetzal prototype consisted of flat tiles laser cut from an acrylic sheet. Printed paper faceplates show both a human-readable statement name and a computer vision fiducial. The statements can be chained together at pivot points at the end of rigid connection arms.

[1]Because the angular orientation of the connectors plugs is unrelated to the angular orientation of the blocks themselves, both the statements and the connector plugs need to be tagged with a computer vision fiducial to construct a digital representation of a physical chain of blocks.

One feature of the Quetzal language is that, similar to flowchart notation, flow-of-control structures such as loops and branches have physical embodiments that are consistent with the algorithmic structures they represent. For example, to form a loop using Quetzal, a child would create a chain of statements that circled back on itself (e.g. Figure 3.3). Thus, if a child were to trace through the program using her finger, she would eventually end up on the same statement where she started. While somewhat appealing, this language feature also imposes limitations on the type and complexity of programs that can be expressed. For example, it is impossible to use Quetzal to create a loop consisting of a single statement because the individual tiles are rigid and connected at fixed pivot points. Likewise, creating nested control structures (e.g. a loop within a loop or a branch statement within a loop) is nearly impossible.

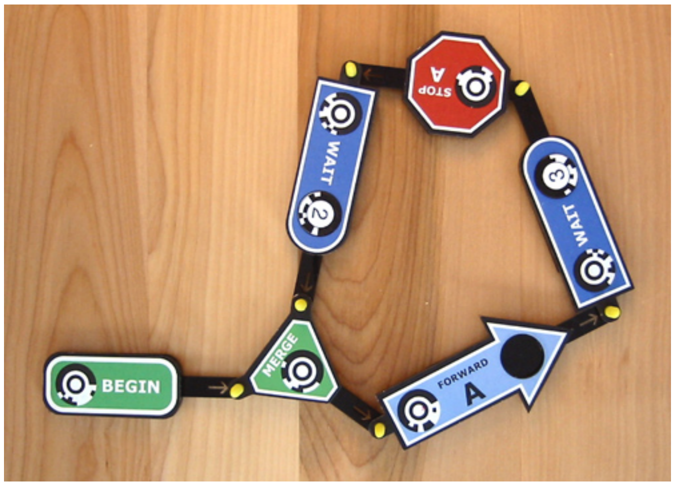


Figure 3.3: This picture shows a simple infinite loop represented in the Quetzal language. Here the physical form of the loop is consistent with the control structure that it represents (i.e. a list of statements that repeats itself over and over again). This feature makes it difficult to create nested control structures in Quetzal.

To address these limitations, I decided to abandon flowchart-style notations and adopt more of a structured programming approach. In this sense, code blocks are demarcated with BEGIN and END tokens, and control structures can be easily nested. With this design decision it is no longer necessary to have physical joints that can pivot as is the case with Quetzal (e.g. Figure 3.3). Instead, much simpler block connections are feasible—for example, interlocking blocks shaped like jigsaw puzzle pieces. These design decisions led to the first set of prototypes for the Tern language (Figure 3.4).

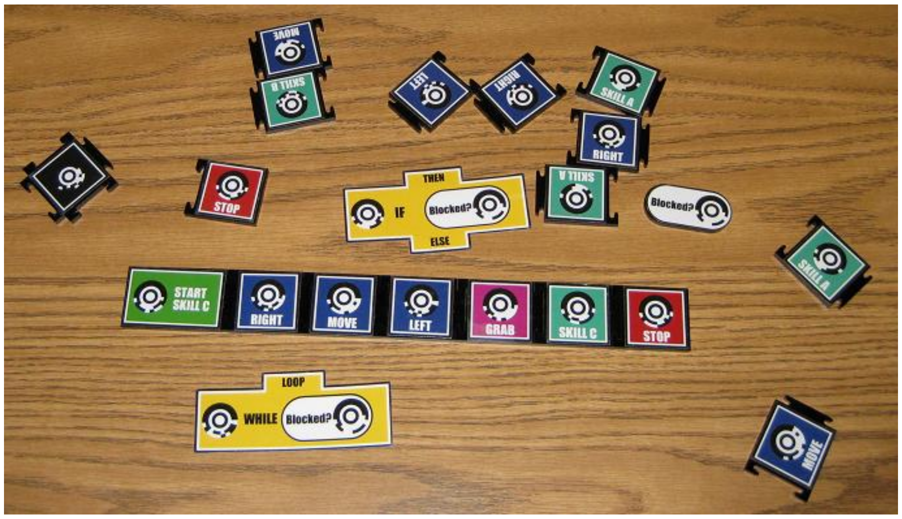


Figure 3.4: I built the earliest Tern prototypes using laser cut plastic and a jigsaw puzzle piece metaphor. The language itself was modeled after the educational programming language, Karel the Robot. This picture shows a recursive subroutine.

Initial Tern prototypes consisted of laser cut plastic tiles; however, in later prototypes, I switched to toy wooden train tracks. The train tracks are more durable, commercially available, and have a more attractive look and feel. One problem with wooden train tracks is that it is not obvious how to represent parameter values for statements. To solve this problem, I attached laser cut parameter sockets to the back side of particular wooden blocks. This allowed me to create different shaped connectors to represent different data types such as numbers and Boolean values. In this way, the language syntax is enforced by the physical form of the blocks themselves. In other words, a Boolean token can only be connected to a statement requiring a Boolean parameter, and a number token can only be connected to a statement requiring a number parameter (e.g. Figure 3.5).

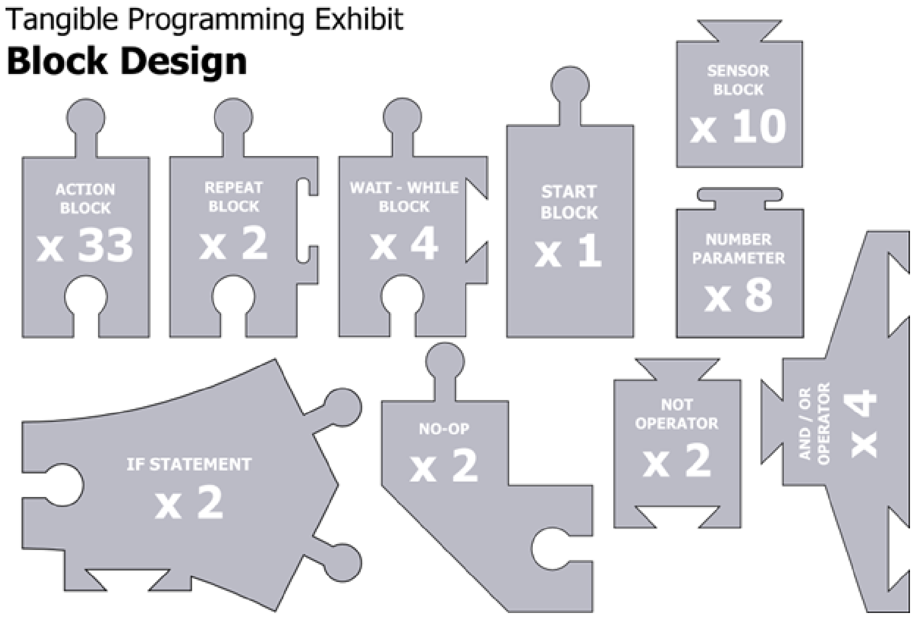


Figure 3.5: Design for tangible programming blocks used in the exhibit.

The version of Tern in use at the *Robot Park* exhibit at the Boston Museum of Science uses these wooden train track parts, which seem to work well in the museum environment. However, the train track blocks have several drawbacks for use in the classroom that became obvious when we observed children using them in our kindergarten interventions. First, the blocks are only big enough to allow space for a computer vision fiducial and a small amount of text. This is problematic for children who are still learning to read. As we discovered with our current prototype, children rely heavily on icons as well as text to interpret a particular block's meaning.

Second, although children have little difficulty chaining these jigsaw puzzle blocks together to form programs, the blocks tend to fall apart when children try to carry their programs around the classroom 3.6. Unfortunately, children need to be able to carry their programs to a specific location in the classroom (the computer with the web camera attached) in order to download their programs to the robot. Finally, the blocks are designed to lie flat on a table surface. As a result, in order for the computer vision system to function properly, we have to suspend a web camera above the programming surface pointing down. This relatively elaborate setup limits the system's portability, increases the teacher's preparation time, and limits the number of stations that can be set up in the classroom for children to use. To address these

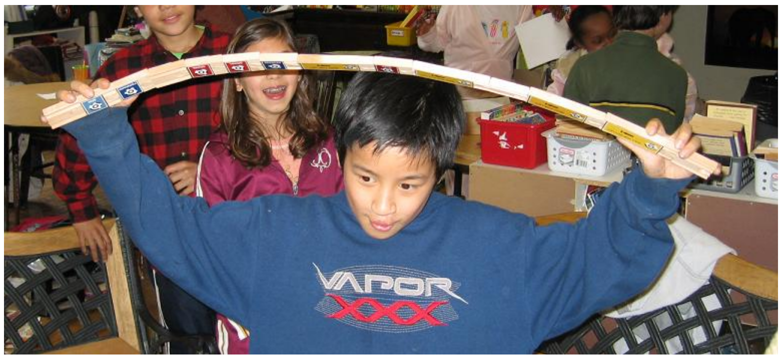


Figure 3.6: The wooden train tracks seem to work well at the Boston Museum of Science; however, they are less ideal in classrooms. One problem is that programs are difficult to carry around a classroom. In this picture a child has improvised a way to carry his program to the scanning station.

problems, I redesigned the blocks using wooden cubes with interlocking pegs and holes instead of jigsaw puzzle tiles. These cubes provide a slightly larger surface area on which to include both text and an icon and the computer vision fiducial (see Figure 3.7). Furthermore, because the cubes interlock, they are easier for children to carry around the classroom. Finally, because I place a computer vision fiducial on every available face of the cube, pictures of programs can be taken from the side rather than from above. This allows the web camera to be placed flat on a table rather than suspended from above. As expected, the prototype based on wooden cubes was much easier for students to use. However, for our next round of evaluation, I plan to make several additional improvements. These include creating slightly smaller and lighter

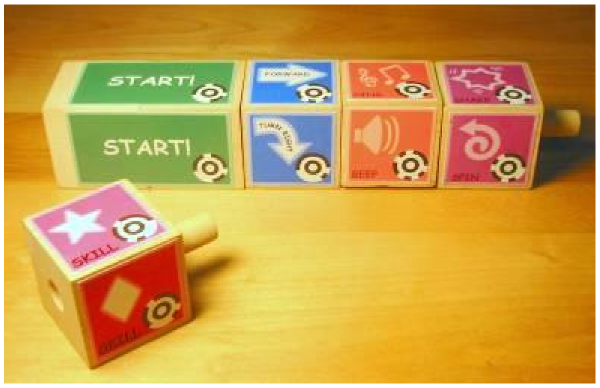


Figure 3.7: To address drawbacks of the wooden train track blocks for use in kindergarten, I redesigned the language to use interlocking wooden cubes with pegs on one side and holes on the other. Mary Murray, an intern a the Boston Museum of Science and a masters student at the Massachusetts College of Art and Design was the originator of this concept.

cubes and exploring better ways to represent the syntax of flow-of-control structures such as loops, parameters, and branches. Children also struggle with the mechanics of downloading their program to a robot. This process involved several discrete steps: 1) turning the robot on; 2) placing the robot in front of an infrared transmitter; 3) placing their program in front of the web camera; 4) pressing the space bar on the computer; and 5) waiting several seconds for their code to download to the robot. From our experience, this process is still too complicated to be practical for young children. It's easy to forget a step, and there are multiple points of possible failure. For example, a child might do everything right but have the robot pointing in the wrong direction. Or, a child might accidentally cover up one of the computer vision fiducials while the computer is taking a picture of the program. In these cases, it can be difficult for children to figure out and fix the problem without the help of an adult. We are investigating a number of potential improvements to this system to simplify the process and reduce the amount of adult assistance necessary.

3.2 Karel the Robot

The inspiration for the syntax of the Tern language came from the educational programming language, Karel the Robot (Pattis et al., 1995). Karel presents novice programmers with a virtual robot inhabiting a grid world on a computer screen. Karel can be programmed to navigate around obstacles (walls) and accomplish tasks with simple objects in its environment (beepers). Figure 3.8 shows a typical programming challenge from Karel the Robot. The Karel programming language was

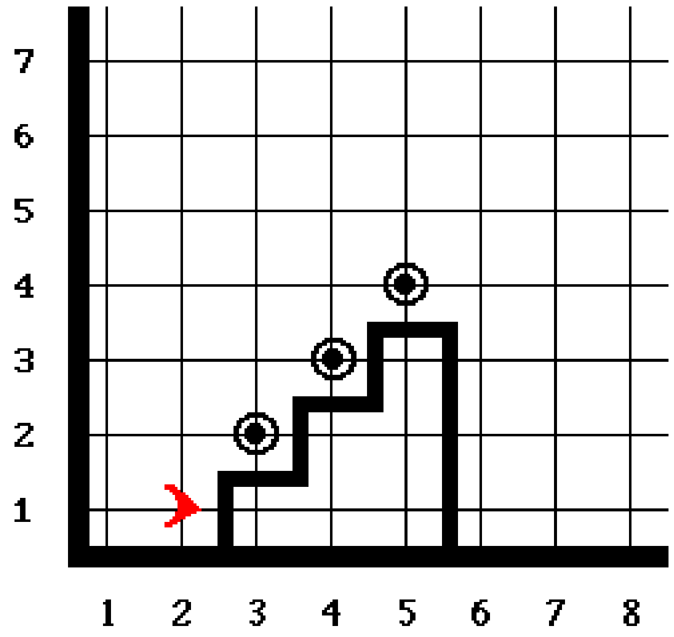


Figure 3.8: Karel the Robot inhabits a grid world on a computer screen. Novice programmers can create programs to navigate the robot around obstacles and to accomplish tasks with simple objects in its environment. Here the challenge is to have the robot climb the stairs and collect all of the beepers.

an appealing starting point for me because it features a very small set of built-in instructions for controlling the robot (e.g. `move`, `turnleft`, and `pickbeeper`). This small instruction set could be easily represented with a reasonable number of tangible programming blocks. Furthermore, Karel instructions are parameterless, further simplifying the task of creating a tangible language. Karel also includes a small number of flow-of-control structures including an `IF/THEN/ELSE` construct, an `ITERATE` loop, and a `WHILE` loop. The `IF` and `WHILE` structures query a small number of Boolean sensor values, and the `ITERATE` structure accepts a single positive integer argument.

The richness of the Karel language, however, comes from the ability of programmers to define new instructions (subroutines) for the robot. For example, an early programming task in Karel is to define a `turnright` instruction. This program following Karel program shows this definition:

```
DEFINE-NEW-INSTRUCTION turnright AS
BEGIN
   turnleft;
   turnleft;
   turnleft;
END;
```

Of course, with a text-based language, programmers have the freedom to define an arbitrary number of subroutines and give them descriptive names. This is not so straightforward with a tangible language, especially one based on passive tangible technology. One challenge is to determine a way to allow for similar capabilities with a tangible representation.

3.2.1 Tangible Karel Concept

My original vision of Tern was to support a full-class programming activity that would allow up to four groups of students (16 to 20 children) to participate in a single virtual world. The teacher would need a single computer with an attached LCD projector and four sets of tangible blocks (one for each group). To conduct the activity, the teacher would project a virtual world onto the wall of the classroom.

The virtual world (see Figure 3.9) would include four robots–one for each team–as well as obstacles and objects for the robots to interact with. The goal of the activity

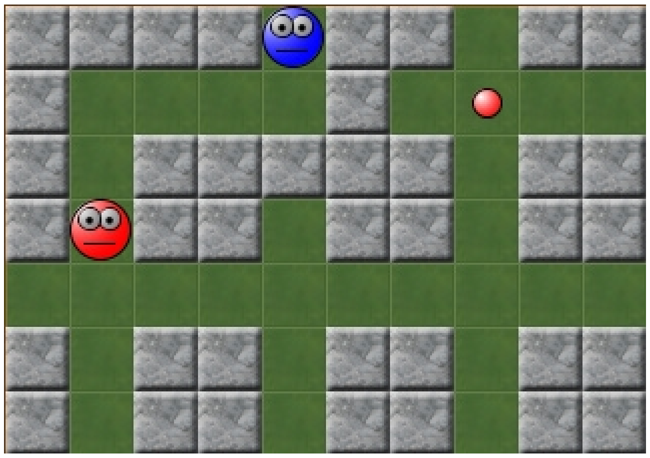


Figure 3.9: My original Tern concept would have allowed groups of children in a classroom to program virtual robots in a grid world.

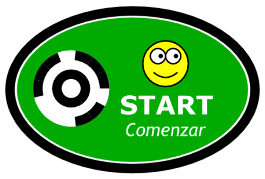
could have been something as simple as, *program your robot to navigate the maze and collect as many red dots as possible.* Students would have then worked together in their groups to define *Skills* (Tern's version of subroutines) and programs for their robot. Students might start with simplistic programs to solve immediate goals for the robot, almost as if they were using a remote control. However, because the process of building and compiling programs is time consuming, ideally groups would realize that they could gain a competitive advantage by writing more general-purpose programs that would allow the robot to navigate autonomously through the maze. In this system, the idea was that Skills would be *linked* into the system. In other words, once defined, a subroutine would persist in the system, available for use in future programs, until it was redefined. This concept of using Tern to program robots in a virtual world never made it past the early prototyping phase and was never tried in a classroom. However, many of the same ideas still exist in the current Tern language for controlling real robots rather than virtual robots.

3.3 Language Specification

In this section I will describe the version of Tern in use at the Boston Museum of Science. All of the figures included in this section reflect what is currently on the floor at the Museum.

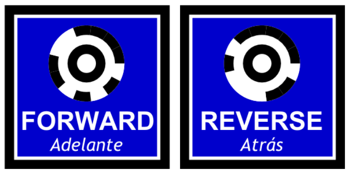
3.3.1 Action Blocks

Action blocks tell the robot to do something. These blocks can be chained together into sequences that the robot will perform one at a time, in order from left-to-right.



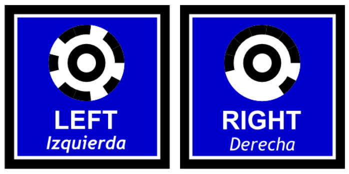
Start

Every program must begin with a Start block. Tern only considers blocks connected to a Start block to be part of a program. There are other types of Start blocks used to declare subroutines discussed below.



Forward and Reverse

These blocks tell the robot to move forward or backward approximately one foot in a straight line.



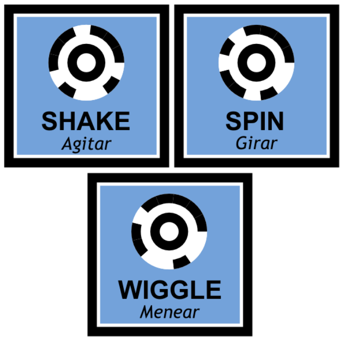
Left and Right

These blocks tell the robot to turn in place 90 degrees to the left or right.

GROWL
Gruñir
SING
Cantar
BEEP
Pitar
WHISTLE
Silbar

Sounds

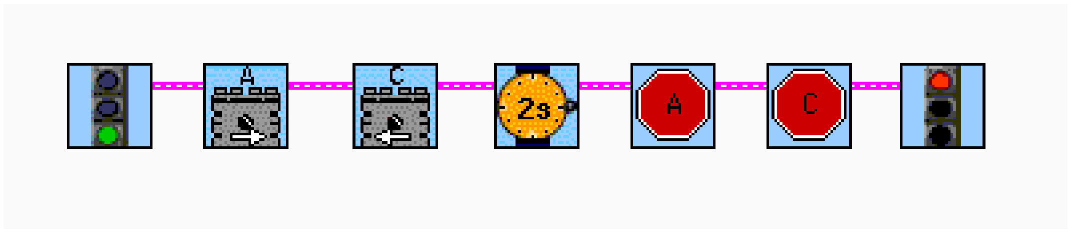
These blocks tell the robot to make different types of sounds—beep, sing, growl, and whistle. The piezoelectric speakers on the iRobot Create and the LEGO Mindstorms platforms are only capable of producing a sequence of beeps; as a result the sounds produced by these blocks are at best stylized imitations of singing, whistling, and growling.



Dance Moves

These blocks tell the robot to perform one of its dance moves—shake, spin around, or wiggle.

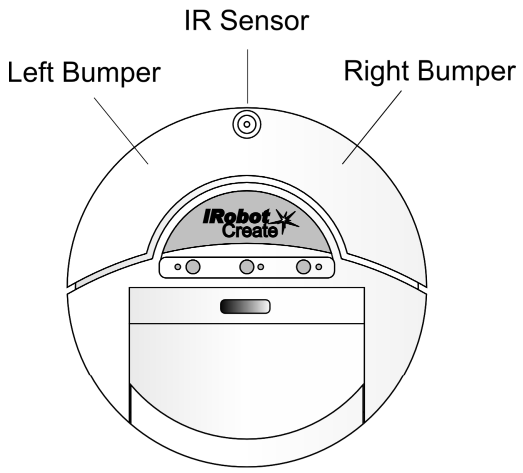
Unlike other educational languages for robotics, these actions are designed to be discrete and observable with a small delay separating one action from another. For example, to make a differential-drive robot spin around using the language, ROBOLAB, would require the following program:



For this language, students must learn that the start and stop commands commands execute instantaneously, causing the motors to turn on and off in unison. Tern offers the higher-level *Spin* block to accomplish the same effect. Kelleher et al. (2007) employ a similar strategy for Storytelling Alice, providing young programmers with a set of high-level actions to help in the creation of animated stories. Ideally the tangible language would provide the ability to program a robot at both levels of abstraction. Teachers could then introduce lower-level blocks as students gained proficiency.

3.3.2 Sensor Blocks

Sensor blocks are used to sample real-time binary sensor data from the robot in order to create programs that react to the robot's environment. All sensor blocks evaluate to a Boolean value of TRUE or FALSE that changes depending on the state of the robot in the world.



Bump Sensors

These blocks check the state of the robot's front bumper. If the bumper is pressed on the right side, BUMP-RIGHT will have a value of TRUE. Likewise, if the bumper is pressed on the left side, BUMP-LEFT will have a value of TRUE. If the bumper is pressed from either side, BUMP will be TRUE. Finally, if the bumper is not pressed at all, the sensor blocks will all have a value of FALSE.

IR Sensor

The Create robot has a sensor that detects near infra-red (IR) light transmitted over a specific carrier frequency. If the robot detects infra-red light, the IR-BEAM sensor will have a value of TRUE. Otherwise it will have a value of FALSE.

Cliff Sensor

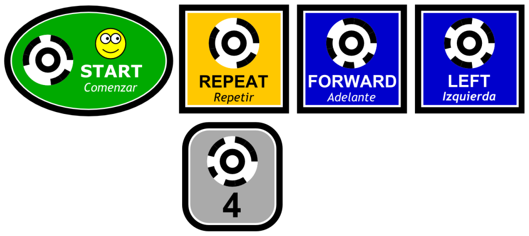
The cliff sensor is used to keep the robot from falling down stairs or off the edge of a table. The cliff sensor works by emitting a beam of infra-red light down towards the floor. If the light beam bounces back, then the CLIFF sensor has a value of FALSE. If the light beam doesn't bounce back, then the CLIFF sensor has a value of TRUE.

3.3.3 Control Flow Blocks

There are several types of control blocks can be used with sensor blocks and number parameters to create programs with loops, branches, and pauses.

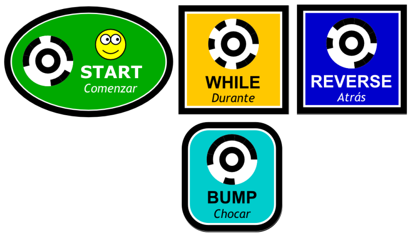
Repeat

This block tells the robot to repeat a set of actions. When combined with a number block, the loop will repeat for a finite number of iterations. For example, this program tells the robot to move forward and then turn left four times in a row. If no number block is provided, then the loop will repeat infinitely. The version of Tern we are using in kindergarten classrooms (ages 5–6) includes an END REPEAT block as well, allowing for nested control structures.



While

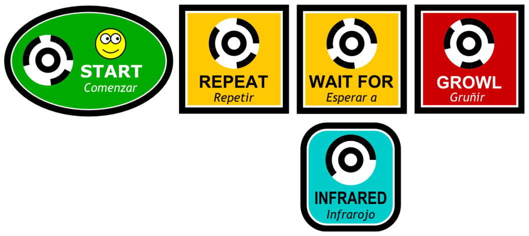
This block tells the robot to repeat a set of actions as long as the value of a sensor is TRUE. For example, this program moves the robot backwards until the bumper is no longer pressed. A while loop will repeat at most 10 times.



Wait For

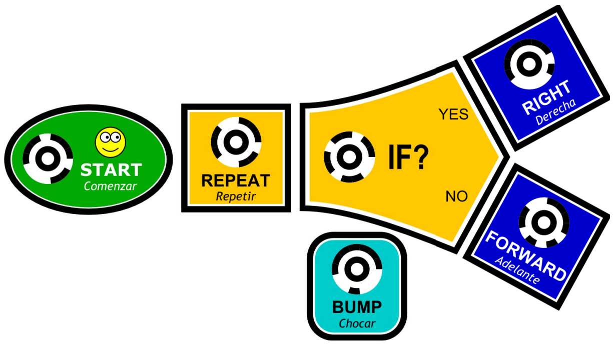
WAIT FOR
Esperar a

The WAIT FOR block pauses a program until the value of a sensor becomes TRUE. For example, this program causes the robot to growl whenever it sees a beam of infra-red light.

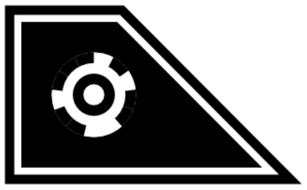


If

The IF block allows you to ask a question in your program. The robot will do one thing or another depending on the answer. For example, this program tells the robot to move forward until it bumps into something. Then it will turn right.



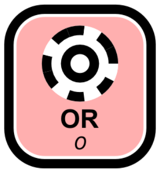
No–Op



This block has no associated action. It's only purpose is to redirect the physical chain of blocks 90 degrees, which is useful if a program takes up too much space on the table.

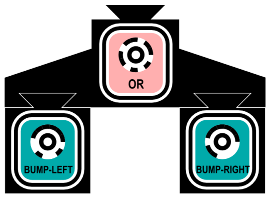
3.3.4 Logic Blocks

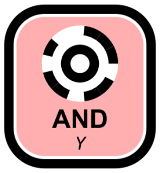
Logic blocks allow for the combination of sensor blocks using logical operations.



OR Block

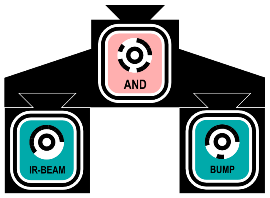
The OR block combines two sensors together. It has a value of TRUE if either sensor is TRUE. Otherwise, it has a value of FALSE.





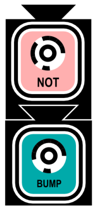
AND Block

The AND block combines two sensors together. It has a value of TRUE only if both sensors are also TRUE. Otherwise, it has a value of FALSE.

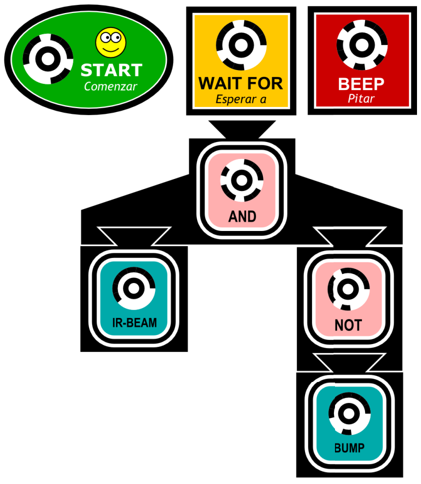


NOT Block

The value of a NOT block is the opposite of the sensor that is connected to it. If the sensor is TRUE, then the value of the NOT block is FALSE. Likewise, if the sensor is FALSE, then the value of the NOT block is TRUE.



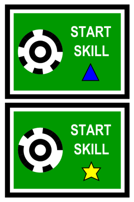
Here is an example program that tells the robot to beep if its bumper isn't pressed and it sees a beam of infra-red light at the same time.



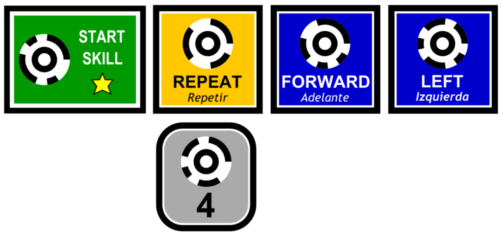
3.3.5 Skills

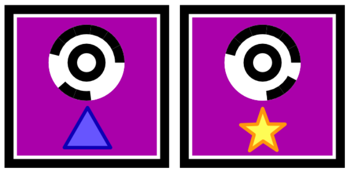
Skills are the equivalent of parameterless subroutines or procedures in other structured languages.

Start Skill



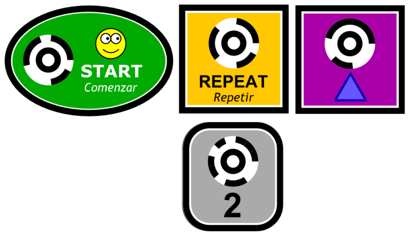
START SKILL There are two blocks available to define skills. These blocks are labeled with two symbols, a star and triangle, representing the skill names. For example, this skill instructs the robot to drive in a square.





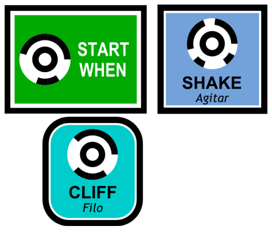
Skill Blocks

Once defined, a skill can be invoked in a program using a skill block. For example, this program uses the star skill defined above to tell the robot to drive in a square two times in a row.



Start When

This block lets you define a skill that runs automatically when a certain event happens. For example, this skill tells the robot to shake whenever it gets picked up (causing its cliff sensors to activate).



3.3.6 Kindergarten Blocks

As described above, the version of Tern in use in kindergarten classrooms (ages 5–6) is based on interlocking wooden cubes. Cubes allow us to represent up to four different statements on a single block. For the most part all sides of a cube are the same. However, in certain cases such as number parameters, light on/off, or sensors, the faces of the cubes have different meanings.

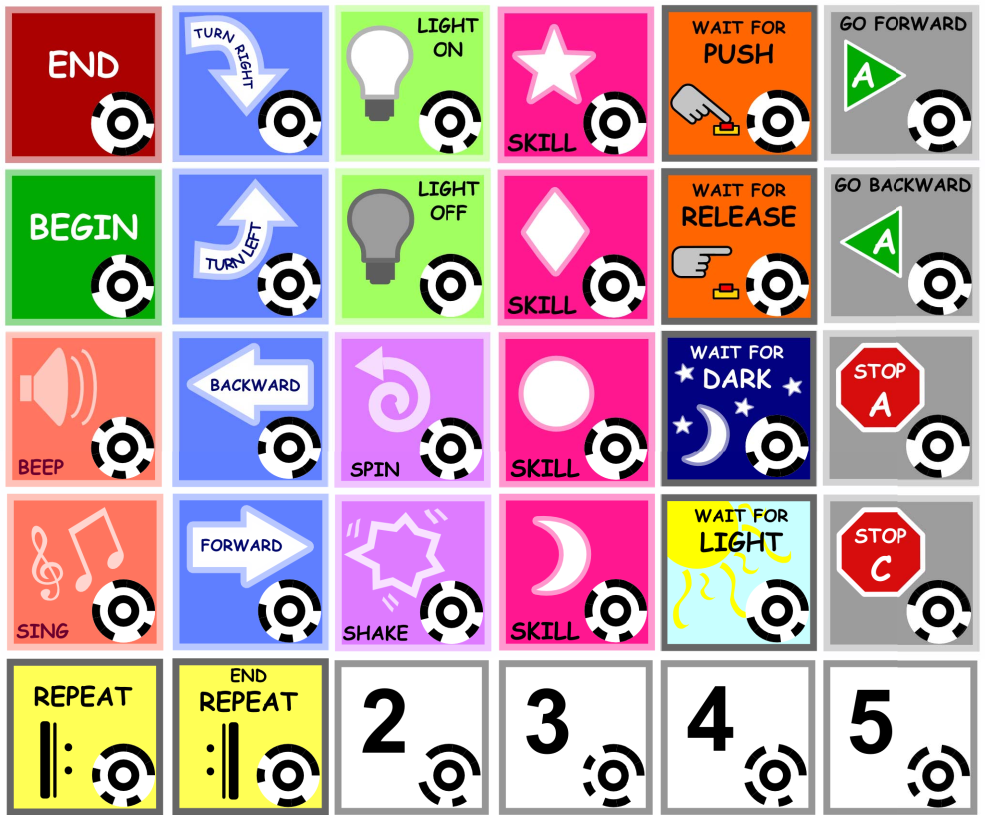


Figure 3.10: This figure shows the latest faceplate designs used for the Tangible Kindergarten project. These blocks combine English text with simple icons representing each block's purpose.

3.4 Future Work

The design of the Tern language is an ongoing process, and the current prototype is far from perfect. In particular, while the basic command blocks (such as FORWARD, SHAKE, etc.) seem to work well, the flow-of-control blocks need improvement. For these blocks, much could be done to use *physical syntax* to help guide children in the construction of logically correct programs. Furthermore, the parameter blocks that work with the flow-of-control blocks could be improved to provide a broader range of possible values and to physically indicate their place in the program. Finally, the idea of subroutines or *skills* needs to be elaborated. For example, it would be ideal if students could create an arbitrary number of skills and provide their own names or symbols to describe these skills. For example, students might use physical tokens of their own choosing, perhaps tagged with RFID, to name a skill.

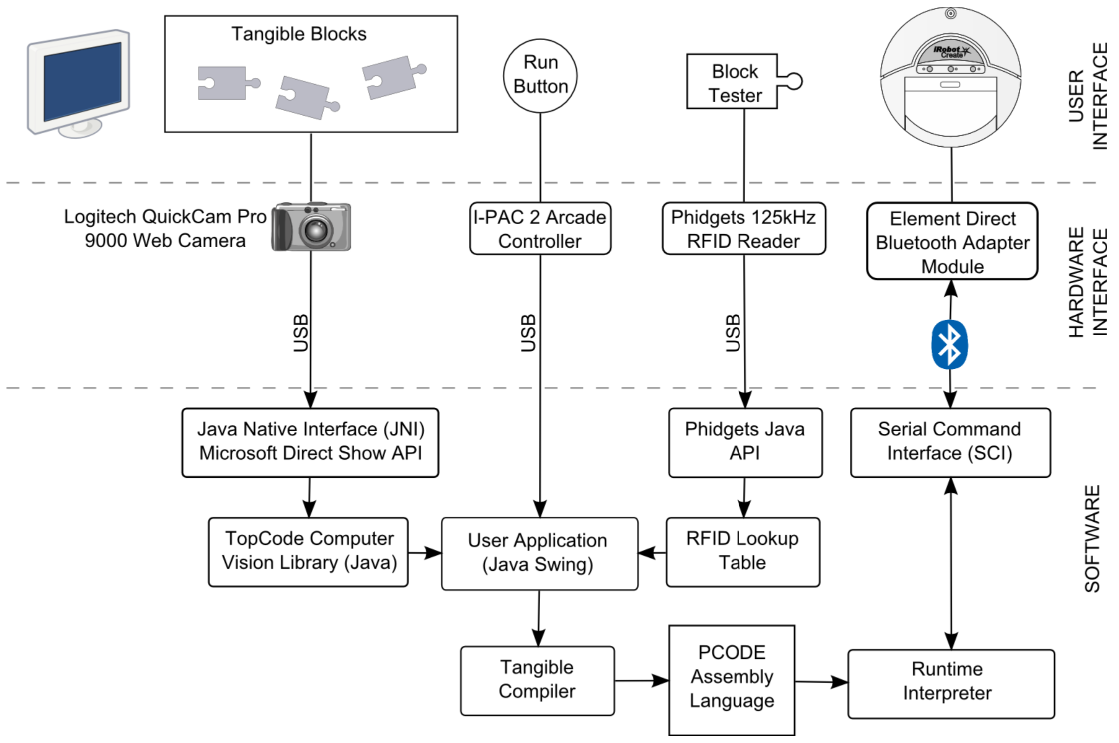
Chapter 4

Tern Implementation

This chapter describes a system–level overview of the tangible programming system deployed at the Boston Museum of Science. This includes a description of the user interface, the hardware interface, and the major software components and libraries. Figure 4.1 provides a summary diagram of these three levels.

Visitors interact with the exhibit through three user interface components. First, visitors can arrange and connect the wooden blocks to create physical programs for the robot to execute. Then, once a program has been created, visitors can *compile* it using an arcade button labeled *Run*[1]. The third user interface component is called *block tester*, which is a block-shaped indentation on the programming console that allows users to test block functionality instantaneously. The block tester is described in more detail below. Visitors also receive feedback from the system through a computer monitor mounted on the far left side of the programming console.

[1]This button press is converted into a keyboard event using a Ultimarc I-PAC arcade controller (http://www.ultimarc.com)



Tangible Blocks
Run Button
Block Tester
USER INTERFACE
Logitech QuickCam Pro 9000 Web Camera
I-PAC 2 Arcade Controller
Phidgets 125kHz RFID Reader
Element Direct Bluetooth Adapter Module
HARDWARE INTERFACE
USB
USB
USB
Java Native Interface (JNI) Microsoft Direct Show API
Phidgets Java API
Serial Command Interface (SCI)
TopCode Computer Vision Library (Java)
User Application (Java Swing)
RFID Lookup Table
SOFTWARE
Tangible Compiler
PCODE Assembly Language
Runtime Interpreter

Figure 4.1: System diagram for the Museum of Science exhibit showing the user interface, hardware interface, and major software modules.

4.1 Image Processing

To convert physical Tern programs into digital code, I created a computer vision fiducial library called *TopCodes*. TopCodes are circular, black-and-white symbols that resemble barcodes. Each statement (or block) in the language is imprinted with a single TopCode that allows the system to determine its position, orientation, relative size, and statement type from a digital image. Because Tern is a compiled language, it is not limited by real-time processing constraints faced by many other computer vision applications. In other words, the system only needs to process an image when a program is compiled, at most a few times a minute rather than ten or more times a second. Because of this, it is possible to use a relatively high-resolution camera and computer vision algorithms that are optimized for accuracy rather than speed.

However, there is a tradeoff to be made. If compile times are too long, there is a danger that users will become frustrated or confused, especially in museum settings. This section discusses the details of the TopCode library.

4.1.1 Digital Camera Interface

For the earliest prototypes of the Tern system, I used a 2.0 megapixel Canon PowerShot S200 digital camera to capture images for for processing. This camera has a shutter, flash, manually operated power button, and requires an external AC power adapter or battery. Canon provides a C-language development kit (CanonSDK) for its PowerShot series of digital cameras, on top of which I implemented a Java native interface (JNI) library. The Java interface could control the flash, optical zoom, and image resolution. Captured images were transferred to the host computer through a USB 1.0 connection with a maximum transfer rate of 12 Mbits/s.

The use of this device was problematic for many reasons. The need for a battery or power supply added complexity to the system and increased setup time in a classroom. The manual power switch and other user-adjustable controls introduced several potential points of failure into the system that were difficult to mitigate through software. Worse, the shutter and relatively slow USB 1.0 connection added approximately seven seconds to the overall compile time, by far the longest part of the process. Finally, the use of a third-party API limited me to a specific device and operating system. Switching to a consumer web camera would have solved these problems; however, at the time, web cameras were limited to VGA resolutions (640 x 480), which was insufficient for accurate recognition of TopCodes on a reasonably large programming surface.

Fortunately, in 2007 several companies released 2.0 megapixel web cameras. To use a web camera instead of the Canon PowerShot, I implemented a Java native interface (JNI) library using the Microsoft DirectShow API. This interface allows Java applications to capture 2.0 megapixel still images on demand from the web camera, which results in dramatically faster compile times (under two seconds overall), eliminates the need for a power supply, reduces possible points of failure, and provides the potential for a cross-platform implementation.

For the museum exhibit we suspended a Logitech QuickCam Pro camera in a light fixture approximately three feet above the programming surface. At this height, a programming surface approximately 4 feet wide and 1.5 feet deep is visible to the camera.

4.1.2 The TopCode Symbol Format

TopCodes (Tangible Object Placement Codes) are black-and-white circular fiducials designed to identify and track tangible objects on a flat surface. By tagging physical objects with a TopCode the library will return an ID number, the location of the tag (in pixel coordinates), the angular orientation of the tag, and the diameter of the tag (in pixels). The TopCode library can identify 99 unique codes and can accurately recognize codes as small as 25 x 25 pixels. The image processing algorithms work in a variety of lighting conditions without the need for human calibration. The core TopCode library is 100% Java.

The TopCode library is based on *SpotCode*[2] symbol format, which was developed by a company called High Energy Magic to commercialize research from the University

[2]http://www.cl.cam.ac.uk/research/srg/netos/uid/spotcode.html

of Cambridge (de Ipina et al., 2002). SpotCodes encode a 42-bit integer in two concentric data rings with 21 sectors each, allowing for a large number of unique codes. A variation of the CRC-32 checksum algorithm is used to detect decoding errors. To simplify the image processing algorithm and to increase accuracy and speed, I reduced the symbol to one data ring with 13 sectors and replaced the CRC-32 algorithm with a simple parity check.

TopCodes are drawn in arbitrary units. The center bulls-eye is drawn in white and is two units in diameter. A black locator ring surrounds the bulls-eye and is one unit thick and four units in diameter. A white locator ring surrounds the black ring and is one unit thick and six units in diameter. Finally, a data ring one unit thick and eight units in diameter completes the symbol. The data ring is divided into 13 equal black or white sectors, with a black sector representing a binary zero and a white sector representing a binary one. Bits are read in a clockwise direction around the symbol with the lowest-order bits read first, as shown in figure 4.2. For accurate decoding each TopCode symbol should occupy a region at least 25 × 25 pixels in an image. Tern uses TopCodes that 0.65 inches in diameter.

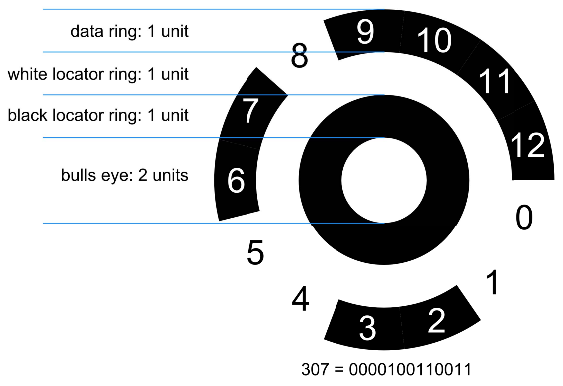


Figure 4.2: A TopCode symbol encoding the number 307. The 13 sectors in the data ring have been labeled in this figure for clarity. A black sector indicates a binary zero, and a white sector indicates a binary one. Bits are read from the outer ring, in a clockwise order around the symbol. The lowest-order bits are read first.

Because TopCodes are circular, they may be read at any angular orientation in the plane. Thus, only the angular orientation that results in the lowest numeric value is considered valid. A simple checksum function is used to reduce the occurrence of false positive symbol recognition. Specifically, the algorithm requires that codes have exactly five white sectors (five sectors with binary 1 values).

4.1.3 TopCode Decoder

Based on the TopCode symbol format, I implemented an efficient algorithm in Java to recognize and decode multiple TopCode symbols in a single image. The algorithm begins with an adaptive thresholding operation which converts full-color images into binary images (black and white pixels only). This thresholding algorithm, developed by Wellner (Wellner, 1993), requires only a single pass over the pixel data to accurately distinguish black pixels from white pixels in a wide variety of lighting conditions. The algorithm requires no human calibration and is capable of compensating for shadows or glare that alter lighting conditions within a single image.

After thresholding, the algorithm scans the image line by line to identify candidate TopCodes. Any pattern of pixels corresponding to a cross-section of TopCode locator ring is marked. Marked pixels are at the center of a pattern: WHITE1–BLACK1–WHITE2–BLACK2–WHITE3, where horizontally adjacent pixels of the same color contribute to one element in the pattern. The pattern must also conform to certain proportion rules. Specifically, the sum of the pixels contributing to BLACK1 and BLACK2 should be roughly equal to the pixels contributing to WHITE2. Furthermore, the absolute value of the difference of the pixels making up BLACK1 and BLACK2 should be less than either the pixel count of BLACK1 or BLACK2 alone.

Any marked pixel surrounded by four adjacent marked pixels (horizontal and vertical neighbors) constitute the center of a candidate TopCode. This stage of the algorithm also requires a single pass over the image; however, it can be combined with adaptive thresholding so that only a single pass is required to both perform thresholding and to identify candidate TopCodes. Although fast, one disadvantage of this technique is that the viewing plane of the camera must be roughly parallel to the programming surface to accurately identify TopCodes. Related algorithms that first locate nested ellipses are accurate even when the camera is not orthogonal to the programming surface (see de Ipina et al., 2002).

The final stage of the decoding process attempts to recognize each of the candidate TopCodes. If the decoding process for a symbol results in a valid ID number (one that has both a valid checksum and corresponds to one of the statements in the Tern language), the TopCode is added to a list of valid codes in the image.

To decode an individual TopCode, its center must first be determined. The TopCode location algorithm will only identify *seed* pixel locations somewhere in the center bulls-eye of candidate codes. To find a more accurate center, the algorithm scans up, down, left, and right from the seed location until it reaches a black pixel in each direction. The midpoint of the left-to-right and top-to-bottom distances between black pixels provides adequate x- and y-coordinates for the center of the code.

The next step is to determine the diameter of a TopCode and, in turn, its base unit size measured in pixels. Again, the algorithm scans in four directions from the center of the code to the outer edge of the black locator ring. The sum of these four distances divided by eight provides an approximate measure of the unit size. The overall diameter is eight times the unit size. If the vertical distance to the black locator ring differs from the horizontal distance by more than one unit, the candidate code is rejected.

The algorithm must also be able to determine a TopCode's angular orientation in order to accurately read its data sectors. Since the orientation of a TopCode in an image is unknown, the decoder takes 10 readings of the code at small angular increments ($2\pi/(21 \times 10)$ radians) and uses the reading that returns the highest confidence level (see below). Readings that are poorly aligned will result in a low confidence level or will fail to decode a TopCode altogether. It would be possible to return as soon as the first valid reading was made, however, the Tern compiler requires more fine-grained information to accurately decode programs. If none of the ten readings produces a valid TopCode, then the candidate is rejected.

To read the bits of a TopCode, a linear cross-section of the entire symbol (*a core*) is sampled at 13 evenly-spaced angles around the center of the code. A core consists of eight evenly-spaced pixel samples taken from the binary image, moving in a straight line from left-to-right across the symbol. Each sample is stored as an integer value between 0 and 255 representing average pixel intensities in a 3×3 pixel region around the sample point. Black pixels have an intensity of 0 and white pixels have an intensity of 255. Thus, a sample consisting entirely of black pixels would have a value of zero, and a sample consisting entirely of white pixels would have a value of 255.

The exact pixels sampled in a core are determined by three parameters: 1) the approximate center pixel of the TopCode; 2) the approximate unit size of the TopCode; and 3) the angular orientation of the core. Each core reading has a corresponding *confidence level*: a number between 0 and 255 that represents the accuracy with which the pixels in the core were decoded. If the three parameters are accurate, the core should have a high confidence level. However, if the parameters are inaccurate, or the area being decoded is not actually a TopCode, then the core sample should have a low confidence level. The confidence level is determined according to the following formula:

$$C = s_1 + s_3 + s_4 + s_6 + (255 - s_2) + (255 - s_5) +$$

$$|2s_0 - 255| + (255 - |2s_7 - 255|)$$

where C is the confidence level, and $s_0 \ldots s_7$ are the eight core samples taken from left to right across the symbol (as shown in the figure below).

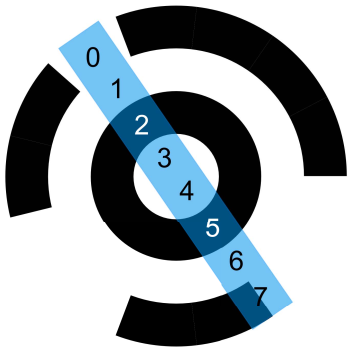


Figure 4.3: A core consists of eight evenly-spaced pixel samples taken from left-to-right in a straight line across the symbol. 13 core samples are taken (one for each sector) during symbol decoding.

Note that samples 1, 3, 4, and 6 should be white (full intensity) and samples 2 and 5 should be black (zero intensity). Sample 0 is the data sector currently being read. Its intensity value should be as close to black or white as possible, if the core is well-aligned with the symbol's sector boundaries. Using the formula $|2s - 255|$ minimizes medium intensity samples (gray values) and maximizes fully black or fully white samples. Finally, because there are an odd number of sectors, core sample 7 should fall roughly between two data sectors. This means that the sample should often have a medium intensity—when it falls between a black sector and a white sector. Thus, the formula rewards gray intensity values for these two samples $(255 - |2s - 255|)$. The total confidence of the entire TopCode reading is just the average of the confidence levels for the 13 core samples.

The actual bits of the code are read from core sample 0. If the intensity of a sample is greater than or equal to 128, a binary 1 is recorded. Otherwise, a binary 0 is recorded. After all 13 core samples are recorded, the bits of the code are rotated to their lowest value and then compared to the checksum. If valid, the algorithm has found a TopCode.

To summarize, the decoding process produces four important pieces of information for each TopCode in an image:

1. The center of the TopCode
2. The diameter of the TopCode
3. The TopCode's angular orientation
4. The TopCode's 13-bit ID number

Taken together, these four characteristics allow the compiler to decode an entire program.

4.1.4 Drawbacks of Computer Vision

While the TopCode library works well in a variety of *normal* lighting conditions, there are, nonetheless several drawbacks to using computer vision to implement a programming language compiler. The primary limitation has to do with occlusion.

If a TopCode is blocked by a hand or an arm, for example, there is no way for the vision system to decode the full program. In the worst case scenario, a child will not realize the problem, and the resulting program will not run as expected. I have tried to include visual cues to help children detect these types of problems [3]; however, occlusion is still commonplace. Another drawback relates to the size of the programming area that can be recognized by the camera. In the museum, we constructed the programming table itself to correspond to the camera's field of view. We were also able to control the lighting condition to result in reliable computer vision. In the classroom, however, we have much less control over these variables. Glare, shadow, and other inconsistent lighting conditions may result in computer vision failures. If the operator of the system is not familiar with the computer vision system, he or she may not be able to correct the problem.

4.2 Tangible Compiler

The Tern system includes a *Tangible Compiler* that is responsible for using information provided by the TopCode computer vision library to convert physical programs into digital code. The section provides a brief overview of the Tangible Compiler algorithm.

The first task of the compiler is to convert a list of TopCodes provided by the computer vision library into a list of `Statement` objects. `Statement` is an abstract Java class (one that cannot be instantiated), and each type of block provided by Tern is represented with a subclass of `Statement`. So, for example, there is a `Start` class, a `Shake` class, and a `Forward` class, each of which extends `Statement`. Each statement

[3]In the latest version of Tern, I use animation and sound to create an effect in which the TopCodes *pop* out in order on the computer screen

subclass has a preassigned TopCode ID number and implements a small number of abstract methods.

To convert TopCodes to statement objects, the compiler uses a `StatementFactory` object. The `StatementFactory` maintains a lookup table of TopCode ID numbers and corresponding statement object types. So, for example, the StatementFactory will generate `Wiggle` statements for TopCodes with ID 155 and `Bump` statements for TopCodes with ID 55.

As these statement objects are instantiated, each statement registers the location of its *sockets* and *connectors* in relation to the location of its corresponding TopCode. For the purposes of the compiler, a socket represents an incoming connection point, while a connector represents an outgoing connection point (see figure 4.4). Each statement's TopCode provides a unit length value and an angular orientation. This information allows statements to define vectors that determine the location of its sockets and connectors in relation to the TopCode's center coordinate (see figure 4.5). It is possible for statements to have at most one socket and zero or more connectors. For example, the Start block has one connector and no sockets; an End block has one socket and no connectors; and an If block has one socket and three connectors. When statements have more than one connector, they are given labels to differentiate one from another. Once the locations of the connectors and sockets have been determined, the compiler pairs connectors and sockets that are coincident within a fixed error radius. The result is a linked-list data structure of statement objects. It is possible to have several disconnected chains of statements as well as individual unconnected statements in a program (see figure 4.6). This will not result in a syntax error as it did in Quetzal (Horn & Jacob, 2007). The goal of this step is to create a faithful representation of the state of the physical blocks in computer

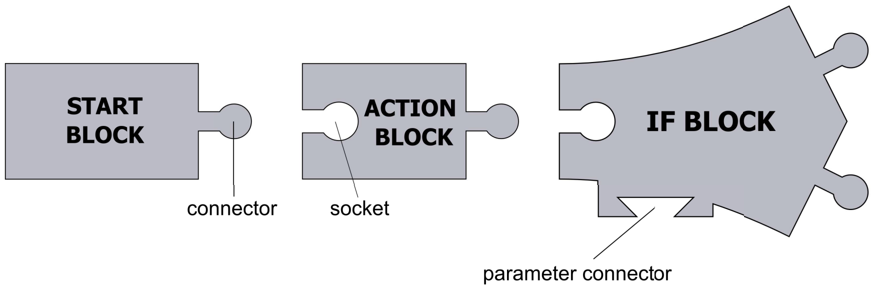


Figure 4.4: Sockets represent incoming connection points while connectors represent outgoing connection points. Statements may have zero or one sockets and zero or more connectors. For example, the If block (shown above) has one socket and three connectors. One of its connectors is for a parameter value.

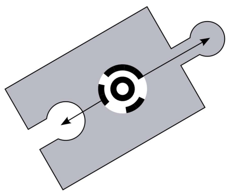


Figure 4.5: TopCodes provide a unit length, angular orientation, and an (x,y) position. This information allows statements to define vectors that determine the positions of its sockets and connectors.

memory. That is, if two blocks are physically connected, then their corresponding statement objects should be linked together. Likewise, if the physical blocks are disconnected, then their corresponding statement objects should not be linked. After forming chains of statement objects, the compiler identifies all statements that implement the `StartStatement` interface. Typically, there will only be one physical Start block available. However, when subroutines are allowed, there may be several blocks that implement the `StartStatement` interface—one for the main program and one for each possible subroutine definition. The compiler then calls a `compile()` method on each of these statement objects. This, in turn, calls a `compile()` method on the next statement in the chain using polymorphic object recursion. The `compile()` method

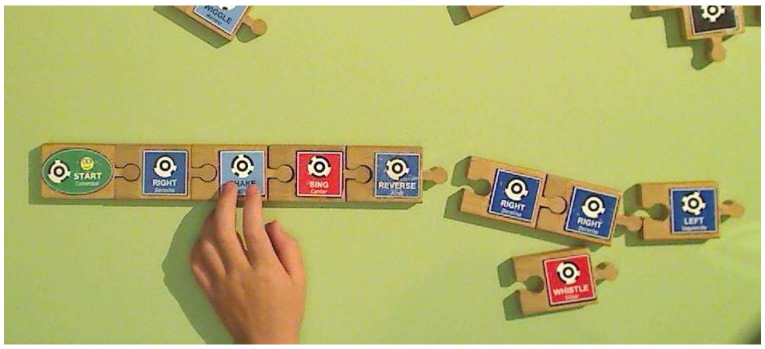


Figure 4.6: Physical programs may contain several disjoint chains of blocks as well individual unconnected blocks. Only the chain that begins with a Start block will be compiled.

is an abstract method, declared in the `Statement` superclass and defined differently in each of the subclasses. To make this concrete, here is the compile method defined in the `Forward` class.

```
public void compile(Program program) throws CompileException {
   program.addInstruction("; --FORWARD");
   program.addInstruction("load-address forward");
   program.addInstruction("call");
   program.addInstruction("pop");
   if (this.next != null) next.compile(program);
}
```

Two things happen in this method. First, the method generates several lines of assembly code. Here the `Program` object passed as a parameter value simply accumulates lines of code from each statement object in the chain. Second, the method calls the `compile()` method for the next statement in the chain, if that statement exists.

4.2.1 Parameters and Sensors

Certain statements such as If, While, Repeat, and Wait accept optional parameter or sensor values. Physically, a parameter or sensor is a block with a specially shaped socket or connector that corresponds to its data type, either Boolean or Integer. For the compiler, parameters and sensors are simply subclasses of `Statement`, which typically define a single socket and no connector. Likewise, statements that accept a parameter value will define a special connector for the parameter block (see figure 4.4). When the `compile()` method is called on these statements, they will use the designated parameter connector to get a value from the parameter. If no parameter is connected, a default value will be substituted. The `compile()` method for a parameter simply adds its value to the accumulated program. For example, here is the `compile()` method for a Bump sensor statement:

```
public void compile(Program program) throws CompileException {
   program.addInstruction("; -- BUMP SENSOR");
   program.addInstruction("load-literal " + BUMP_CODE);
   program.addInstruction("sensor");
}
```

And here is the corresponding `compile()` method for a While loop block. Note, that if there is no parameter connected, a default value of zero (false) is substituted in. This code has been simplified for clarity.

```
public void compile(Program program) throws CompileException {
   program.addInstruction("; --WHILE");
   setDebugInfo(program);
   String loop = ":loop" + program.genLabel();
   String done = ":done" + program.genLabel();
   program.addInstruction(loop);

   // check condition
```

```
      if (param != null) {
         param.compile(program);
      } else {
         program.addInstruction("load-literal 0");
      }
      program.addInstruction("load-address " + done);
      program.addInstruction("if-false");
      program.addInstruction("yield");

      // execute the next statement in the loop
      if (this.next != null) next.compile(program);

      // loop
      program.addInstruction("load-address " + loop);
      program.addInstruction("goto");
      program.addInstruction(done);
   }
```

4.3 Behavior-Based Assembly Language

In the sample code above, the `compile()` methods for the various statement types generate assembly code. This is an example of *PCODE*, a stack-based assembly language that I created in collaboration with Daniel Ozick, a former iRobot software engineer. PCODE is a behavior-based language that builds on the notion of subsumption architectures (Brooks, 1986). PCODE supports a single data type, a 16-bit integer, and provides three first-class language elements: *functions*, *processes*, and *behaviors*.

4.3.1 Functions

A function is a named section of code that can accept any number of arguments from the stack and leaves a single return value on the stack. The instructions `call`, `return`, and `frame` facilitate the stack discipline for function calling (see table C.1 for details), including pushing a return address onto the stack and jumping back to the return address when the function call is complete.

4.3.2 Processes

A process in PCODE is similar to a thread in a language like Java or C++. Each process maintains its own stack, instruction pointer, stack pointer, and frame pointer. However, unlike threads, processes are not arbitrarily preempted by an kernel. Instead, a process must include a `yield` instructions to transfer control to the next waiting process. Furthermore, processes have no priority levels. Instead, processes are serviced in a fixed, round-robin order. Thus, it is incumbent on the programmer to write process code that does not starve other processes or extend the cycle time (the time it takes to process all of the processes and behaviors in turn) beyond an acceptable performance limit. Here is an example that shows the first 20 lines of the blink process. This process flashes the LEDs on the iRobot Create to create a *heartbeat* effect.

```
process blink
:while32
load-literal 1
load-address :done32
if-false
load-literal 1
load-literal 2
led-on-off
load-literal 0
load-literal 3
led-on-off
load-literal 700
timer
:sleep33
yield
```

```
load-address :done33
if-timer
load-address :sleep33
goto
; ...
```

Looking at this example, it is clear that it would be error-prone and tedious to write programs in assembly language by hand. Thus, to improve this process, I created a simple, high-level language based on the syntax of the Python programming language. The same blink process can be defined in the high-level language using this code:

```
process blink:
   while true:
      led2 on
      led3 off
      sleep 700ms
      led2 off
      led3 on
      sleep 700ms
```

I implemented a compiler for the high-level language using Python and the open source *pyparsing* module[4], which allows for the definition and execution of simple context-free grammars.

4.3.3 Behaviors

A *behavior* is similar to a process but has several important differences. First, behaviors have fixed priority levels, and only one behavior may be active in a given cycle. Furthermore, behaviors must include a small piece of code called a *checkpoint* that determines whether or not they should be activated. In every cycle, the interpreter

[4]http://pyparsing.wikispaces.com/

runs this checkpoint code for all latent behaviors with a higher priority level than the current active behavior. If any behavior's checkpoint returns true, it will take the place of the current active behavior. In general, behaviors are used to allow the robot to *react* to sensor events from its environment. For example, in Tern, user programs run as a low-priority behavior. One problem is that a user program might accidentally cause the robot to drive off the edge of a table and crash onto the floor—this was an actual problem for one of our prototype installations at the museum. Fortunately, the iRobot Create has built-in cliff sensors that help it avoid falling down stairs or off of ledges. Tern includes a higher-level behavior that will interrupt the user's programs and stop the drive motors if the robot's cliff sensors are activated. This behavior is implemented with the following high-level code:

```
behavior [5] cliff-avoid:
   start-when: <cliff>
   stop()
   while <cliff>:
      go(-100, 0)
      yield
   stop()
```

Here the number five in square brackets indicates the behavior's priority level, and the `start-when` line is the behavior's checkpoint code. The full PCODE language specification is provided in Appendix C.

4.4 PCODE Interpreter

I implemented two different PCODE interpreters. The first runs on the iRobot Create *Command Module*. The command module consists of an Atmel AVR ATMega168 microcontroller packaged to plug into the iRobot Create cargo bay. The Command Module can be programmed through a USB interface using an open source implementation of the GNU C compiler. The command module plugs into the iRobot Create and issues commands to the robot's onboard processor through a serial interface. PCODE programs are stored as an array of bytes in the command module's flash memory. The command module and the iRobot Create are a self-contained unit. That is, users can create tangible programs that are compiled into PCODE instructions and then *permanently* stored in the command module. These programs can then be run anywhere at any time without needing a tether to the PC that originally generated them.

The second PCODE interpreter works differently. I implemented this interpreter in Java specifically for the Museum of Science exhibit. This interpreter is designed to run on a PC and remotely control the iRobot Create through a wireless Bluetooth serial connection. In this way, the PC knows which Tern instruction the robot is currently executing and can also receive sensor data in real time. The PC, in turn, displays this information to museum visitors on a computer monitor that is part of the exhibit installation. A screen shot is shown in figure 4.7.

4.5 Serial Command Interface

iRobot Corporation publishes an open command interface (OCI) for its Create and Roomba line of robots to allow developers to control their robots through exposed serial ports. This command interface includes opcodes to set the state of various actuators (including the drive motors, LEDs, the speaker, and auxiliary I/O ports) and to query the state of onboard sensors. A third-party company, Element Direct,

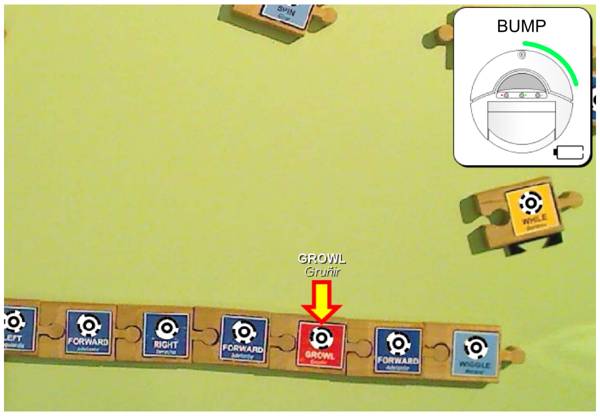


Figure 4.7: The exhibit computer displays the Tern statement currently being executed by the robot as well as realtime sensor data. This figure shows a cropped screenshot from the exhibit computer monitor. The screenshot shows a debug arrow over the Growl statement as well as the current state of the robot's sensors, showing that the right bump sensor is activated.

sells a Bluetooth serial adapter for the iRobot Create that allows the creation of a virtual serial connection between a Create and the host exhibit computer. I created a Java interface layer to facilitate communication between the runtime interpreter and the iRobot Create.

4.6 The Block Tester

The *block tester* is a user-interface component of the exhibit that allows visitors to instantaneously *test* the functionality of a block without having to go through the process of creating and compiling a program. It consists of a black laser-cut piece of extruded acrylic with a block-shaped hole cut in the middle. This part is mounted flat on the lower-left side of the programming surface. Directly beneath this part on the bottom side of the programming console, I mounted a Phidgets 125kHz RFID reader.[5] This reader can recognize one RFID tag at a time and has a range of

[5]http://www.phidgets.com

approximately four inches for tags in plane with the antenna. The reader includes a jack for an external LED indicator light. We threaded an LED through a hole in the programming surface to indicate to visitors when a block is recognized by the system.

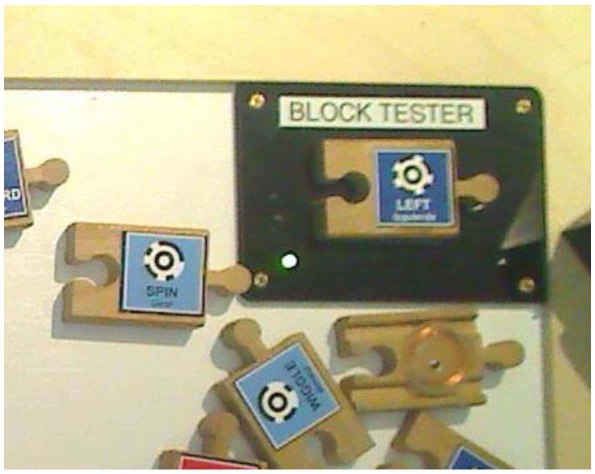


Figure 4.8: The block tester is a block shaped indentation on the programming surface of the exhibit installation. This photograph shows a prototype of the block tester in use. One block below the block tester is turned over revealing its RFID tag. The current exhibit uses blocks with RFID tags embedded inside the blocks.

The Phidgets board connects to the exhibit computer through a USB port. Phidgets also provides a Java language API that triggers software events when tags are added or removed from the reader's field of view. For the block tester to work, we embedded a single 125 kHz RFID tag (approximately 1 inch in diameter) inside the wooden blocks (see Figure 4.9). In software, a lookup table provides a translation between a unique RFID tag and a Tern statement type. When a tag is recognized, the exhibit computer performs three actions. First, it produces a small snippet of assembly code that is sent to the runtime interpreter and immediately executed by the robot. Second, the application displays a help message on the computer screen that explains the function of a particular block (Figure 4.10). This is particularly useful for flow-of-control blocks that produce no observable effect on the part of the robot. Finally,

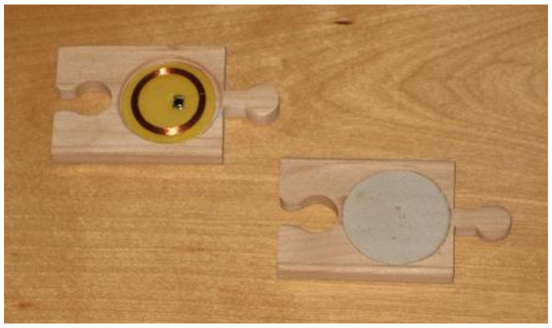
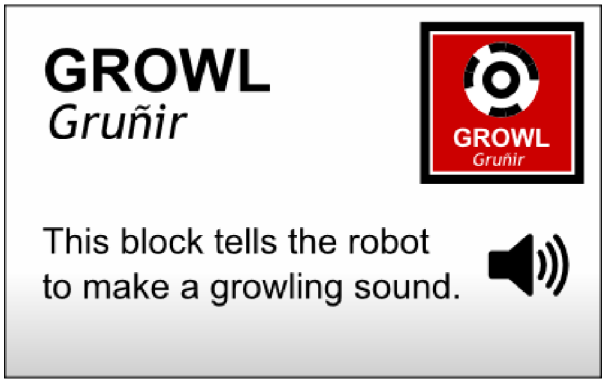


Figure 4.9: We embed a one inch diameter 125 kHz RFID tag inside each programming block. A software lookup table provides a translation between the unique RFID tag number and a Tern statement type. An RFID tag has been placed in a hole drilled in the block shown on the left. The hole has been filled with a wood filler and sanded smooth on the block on the right.

the exhibit computer plays an audio clip that *says* the name of the block in both English and Spanish. The design decision to include the block tester was based on



GROWL
Gruñir

This block tells the robot to make a growling sound.

Figure 4.10: When a block is placed in the block tester, a help message describing the block is displayed on the exhibit computer screen. This figure shows an example help message, the Growl block. In addition to displaying this message, the exhibit computer also plays an audio file that *says* the name of the block in English and Spanish, and it produces a snippet of assembly code that causes the iRobot Create to immediately act out the command.

two goals: First, we thought that the block tester might serve as an entry point into the exhibit and then, in turn, into programming. Our hope was that in their initial exploration of the exhibit, visitors would notice the block tester and place a block in it. The immediate reaction of the robot might then encourage further exploration of the exhibit. In other words, we were providing early *success* for visitors—what Sue Allen terms *immediate apprehendability* (Allen, 2004). Second, we thought that the block tester might serve as a handy reference for visitors in the process of creating a program—imagine the equivalent of a tooltip for a tangible interface. The block tester design decision was a tradeoff with a potential downside. We were concerned that once visitors discovered the block tester they would ignore the programming capabilities of the exhibit, using the block tester exclusively as a sort of remote control system. Through informal observations, we confirmed that these fears were not unfounded. Indeed, some visitors do use the block tester exclusively, never creating a program. However, while we have not done a formal evaluation of this aspect of the exhibit, the current consensus of the museum staff is that the block tester adds more to the exhibit than it detracts.

Chapter 5

Robot Park: Tangible Programming at the Boston Museum of Science

In their mission to engage, inspire, and open minds, science museums have been at the forefront of innovative interaction design for children. They seek to provide memorable learning experiences that cannot be duplicated in schools, libraries, or the home. In an effort to engage diverse populations, science museums have adopted a constructivist approach, offering self-guided, authentic experiences that allow visitors to construct their own knowledge by exploring scientific or technological concepts in a hands-on way (Allen, 2004; Humphrey & Gutwill, 2005; Zheng et al., 2007). In this sense, museums attempt to foster learning that is intrinsically motivated. In other words, visitors themselves should be the principle driving force behind their own explorations (Allen, 2004; Papert, 1980; Resnick et al., 1996). For these types of interactions, visitors are encouraged to form their own opinions and hypotheses, discuss them with friends and family, and test them in real time.

Not surprisingly, computer technology plays a role in many interactive exhibits, empowering designers to enrich the visitor experience. Yet, with this power, there is also a temptation to program not just the computer, but also the learning experience itself. Rather than facilitating self-guided, authentic experiences, computer-based exhibits often lead visitors through scripted presentations with predetermined-determined outcomes. Physical manipulation is replaced by pressing buttons or rolling trackballs. As Ansel (Ansel, 2003) points out, most computer-based exhibits could just as easily be experienced from home over the Internet. And, despite the fact that people tend to visit museums in social groups (families or class visits), computer-based exhibits tend to encourage single user interaction, sometimes to the detriment of the social group as a whole (Heath et al., 2005; Hornecker & Buur, 2006; Serrell, 1996). By combining

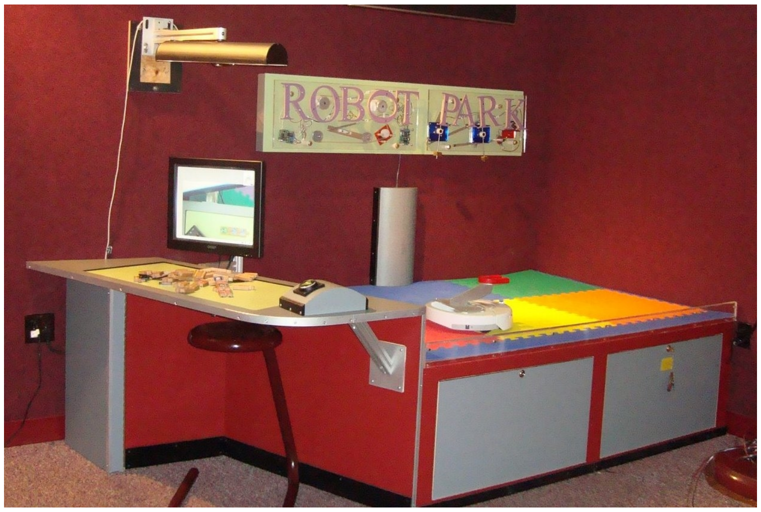


Figure 5.1: The *Robot Park* exhibit is a permanent installation in Cahners ComputerPlace at the Boston Museum of Science. Visitors use our tangible programming interface, called Tern, to control a robot on display.

the capabilities of computer technology with the richness of physical interaction, Tangible User Interfaces (TUIs) have been cited as an appealing alternative to traditional screen-based computer interaction for informal science learning (Hornecker & Stifter, 2006). Unfortunately, however, TUI technology can be expensive and unreliable, often making it impractical for use outside of laboratory settings. Of course, this is not always the case, and there are many examples of thoughtfully designed TUIs in informal science learning settings (see Hornecker & Stifter, 2006; Rizzo & Garzotto, 2007). For example, Hornecker and Stifter describe a museum exhibit that combines a physical abacus with a digital display to create an engaging, collaborative, and interactive informal learning experience (Hornecker & Stifter, 2006).

In this chapter I describe the design and evaluation of *Robot Park*, a tangible computer programming and robotics exhibit that I developed in collaboration with the Boston Museum of Science (Horn et al., 2008, 2009). This exhibit is a permanent installation at the Museum and has been on display since October 2007. In its first year, the exhibit was visited by approximately 20,000 people. The goal of the exhibit is to provide a hands-on learning experience for children to introduce concepts of computer programming and robotics. In describing the exhibit I discuss five design considerations for the use of tangibles in museum settings that guided both the development and the evaluation of this exhibit. In doing so, I revisit the notion of passive tangible interfaces. For the exhibit, the use of a passive tangible interface served both as a way to address practical issues involving tangible interaction in public settings and as a design to promote reflective thinking.

For the evaluation of the exhibit I worked with Erin Solovey from the Tufts University Human-Computer Interaction Lab to conduct a study involving observations of 260 people that compares mouse-based computer programming to tangible computer programming in the museum setting. Our results show that the tangible language and the graphical language are equally easy for visitors to understand. However, the tangible language offers several significant advantages from an informal science education perspective. Among these, the tangible interface is more inviting and provides better support for active collaboration. Furthermore, the use of the tangible interface results in a more child-focused experience, in which children seem to be more actively involved and self-motivated. Parents, in turn, take on more of a supporting role and less of an instructing role. While there is still much work to be done in this area, I hope that this study will provide concrete evidence that tangible interaction can be an effective way to promote intrinsically motivated educational activities for children.

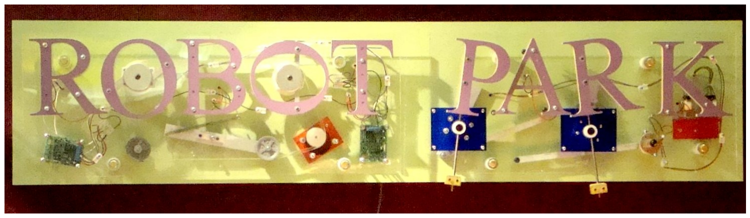


Figure 5.2: The Robot Park sign will light up and move when visitors program the robot to move to a special target area at the back of the exhibit platform.

5.0.1 Exhibit Overview

The Robot Park exhibit allows museum visitors to control an iRobot Create robot by creating computer programs using the Tern tangible programming language. These programs consist of chains of wooden blocks shaped like jigsaw puzzle pieces that represent actions for the robot to perform, such as SHAKE, BEEP, or TURN LEFT; control-flow structures such as a WHILE loop and a REPEAT loop; and robot sensor values such as a bump sensor, and an infra-red light detector. Visitors press a button to compile their programs, which are converted into digital instructions using a reliable, low-cost computer vision system called TopCodes (Chapter 3). Visitors' programs are then transmitted wirelessly to the robot through a Bluetooth connection. The robot, in turn, immediately begins to act out its instructions. This entire process takes around two seconds to complete. A computer monitor displays a picture of the visitor's program, and an arrow on the screen highlights the instruction that the robot is currently executing. The exhibit also features a Block Tester, which is a block-shaped indentation on the upper-right side of the programming console. When a visitor places a block in the indentation, the robot immediately performs that action, and the system displays a help message about the block on the computer monitor.

To help engage visitors, the exhibit includes a built-in challenge activity. By programming the robot to drive to a special target at the back of the robot platform, visitors can cause the Robot Park sign (Figure 5.2) to light up and move. I also

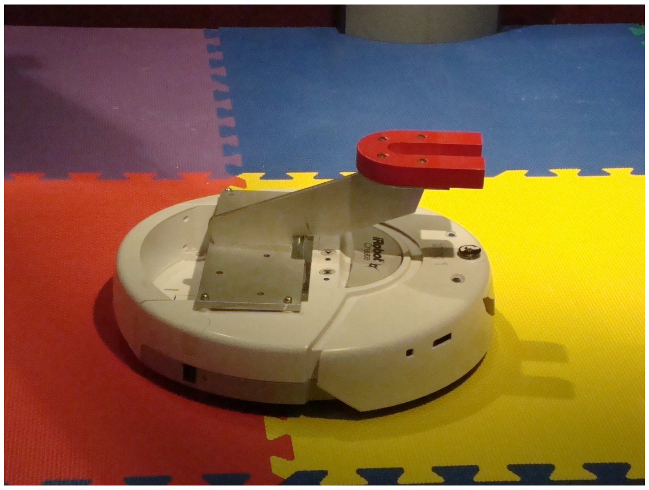


Figure 5.3: Visitors' programs are sent to an iRobot Create robot through a wireless Bluetooth connection. The large red horseshoe magnet on the robot activates the Robot Park sign shown in Figure 5.2

created a booklet with a set of challenge activities that visitors can try. The booklet consists of a set of two sided cards (see Figure 5.4 for an example); on one side is the challenge, and on the reverse side is a possible solution.

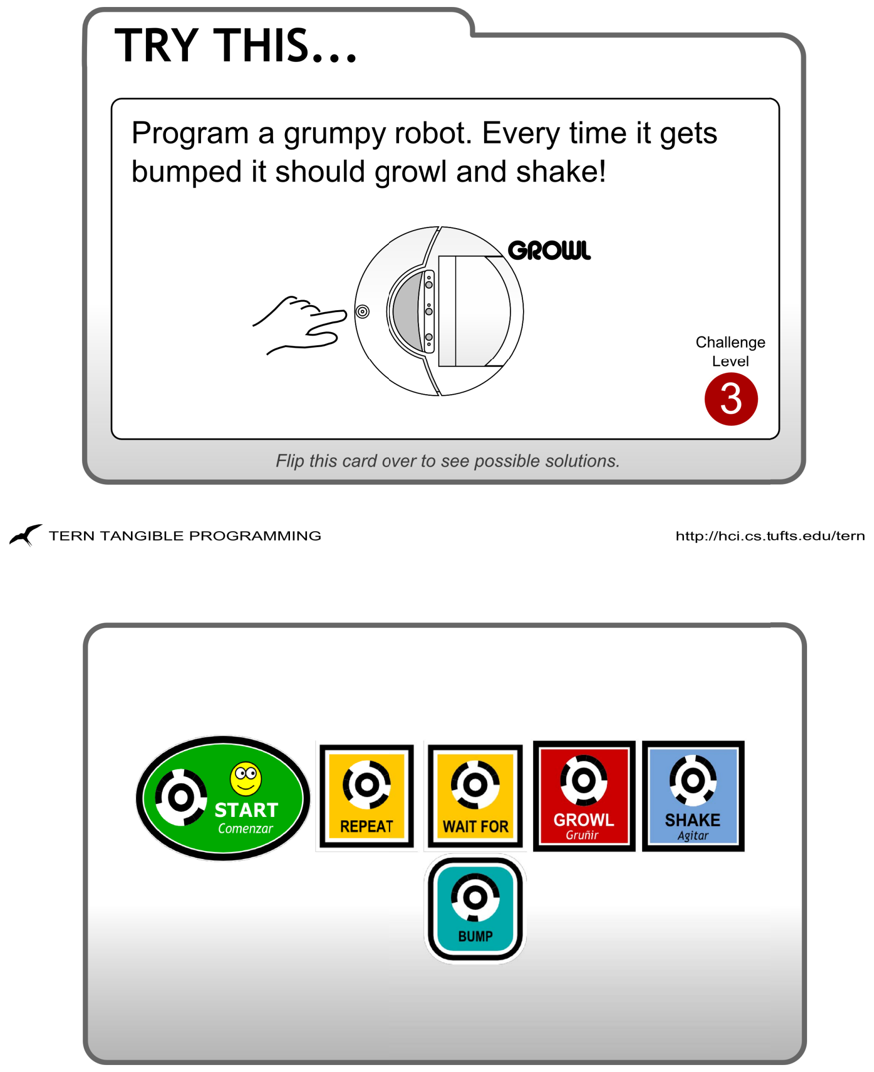

TRY THIS...
Program a grumpy robot. Every time it gets bumped it should growl and shake!
GROWL
Challenge Level
3
Flip this card over to see possible solutions.
TERN TANGIBLE PROGRAMMING
http://hci.cs.tufts.edu/tern
START
Comenzar
REPEAT
WAIT FOR
GROWL
Gruñir
SHAKE
Agitar
BUMP

Figure 5.4: I created a set of challenge cards for the exhibit with a challenge on the front side of the card (top) and a solution on the reverse side (bottom).

5.0.2 Guiding Theme and Learning Objectives

In informal science learning settings such as museums, educational priorities are often different from those of classrooms. Without a teacher or curriculum requirements to guide activities, there is less of a concern for standards or students' performance on tests and quizzes. In a sense, helping to shift children's attitudes and preconceptions is at least as important as conveying high-level science and technology concepts. At the Boston Museum of Science, exhibits are designed with the belief that "it may be more important for visitors to acquire experience, observation, perception, experimentation, imagination, discovery, thinking like a scientist than to learn any specific facts in any specific fields of science" (Boston Museum of Science, 2001). Thus, we are less concerned with teaching complicated robotics and computer programming concepts than with giving visitors a positive, hands-on experience that might inspire them to learn more about computer programming and robotics on their own. The goal is for visitors to walk away thinking: *I programmed a robot today, and it was easy and fun!* Secondary learning objectives include conveying basic vocabulary ideas for concepts such as robots, sensors, actuators, and computer programs.

5.0.3 Audience

We designed the exhibit with elementary and middle school children in mind, although we hope to engage older children and adults as well. Our goal is to engage girls as well as boys and be inviting to visitors of diverse backgrounds. The programming blocks themselves are labeled in both English and Spanish, and the exhibit will eventually be completely bilingual.

5.0.4 Setting

The exhibit is on display in Cahners ComputerPlace, one of the Museum's three Discovery Spaces. Unlike other areas of the Museum, the Discovery Spaces are staffed full-time by volunteers who guide visitors through activities and interpret exhibits. One advantage of this setting is that it allows for incremental, live prototyping of exhibits with visitors. For example, our earliest prototype exhibits consisted of nothing more than a table top, a computer monitor, and a robot on the floor (see Figure 5.5). Furthermore, staff members provide us with invaluable feedback, identifying both positive and negative aspects of visitor interactions. Although installed in a staffed area, we designed the exhibit to be successful in unstaffed environments, and, in practice, visitors regularly interact with the exhibit without any assistance from staff members.

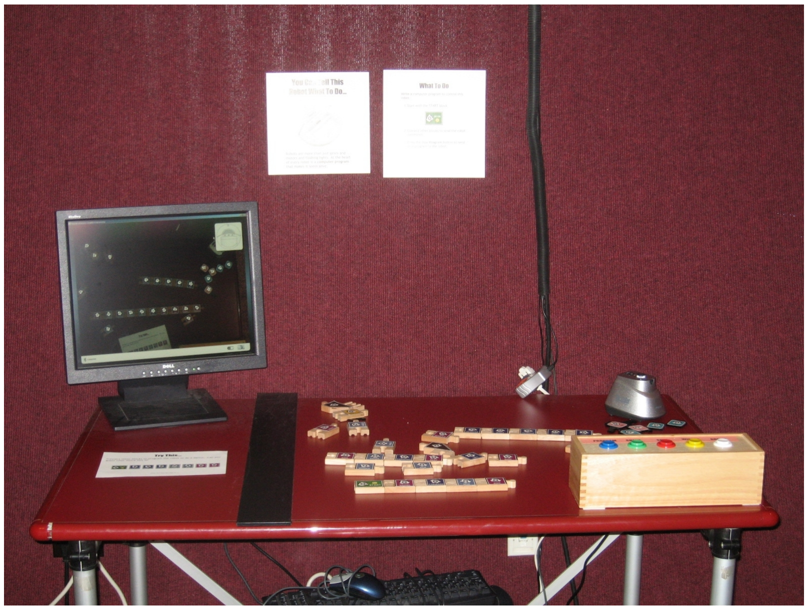


Figure 5.5: Our earliest exhibit prototypes consisted of nothing more than a table top, a computer monitor, and a robot on the floor.

5.0.5 Installation

Following the Museum's guidelines for universal access, the installation was designed according to the Americans with Disabilities Act standards for accessibility. We considered the height and reach of children in designing the programming console and we provided enough room for two or three active participants. The exhibit is wheelchair accessible, and we provided ample surrounding space for passive observers. We were also careful to design the installation so that both the robot and the programming console are easy to observe. Thus, the programming console is angled outward towards the room to increase visibility.

The robot is prevented from falling off the edge of the platform by a short plexiglass barrier that runs the length of the front edge of the exhibit. We designed this barrier to give the impression that the robot is *on display*, yet also to give visitors the freedom to touch both the robot and other interactive elements on the platform—currently the exhibit features a set of large cardboard building blocks on the platform to encourage visitors to create obstacle courses for the robot. Our goal was to produce an effect similar to that of a tide pool exhibit at an aquarium, where visitors are encouraged to *respectfully* touch the organisms on display.

After brainstorming and debating several different themes for the exhibit (including a high-tech cityscape, and a mechanical flower garden), we settled on the Robot Park concept. Here we made a conscious effort to create an exhibit that would be appealing to girls as well as boys.

5.0.6 Implementation

We started work on the exhibit in May 2007 beginning with a prototype of the Tern programming system. To the language I added parameters, sensors, structured loops, and other flow control blocks. I then developed a custom runtime interpreter and an assembly language for controlling robots in real time over a wireless Bluetooth connection. I also made substantial changes to the computer vision system, improving both speed and accuracy, while at the same time switching from a point-and-shoot digital camera with a shutter and optical zoom to a two megapixel web camera.

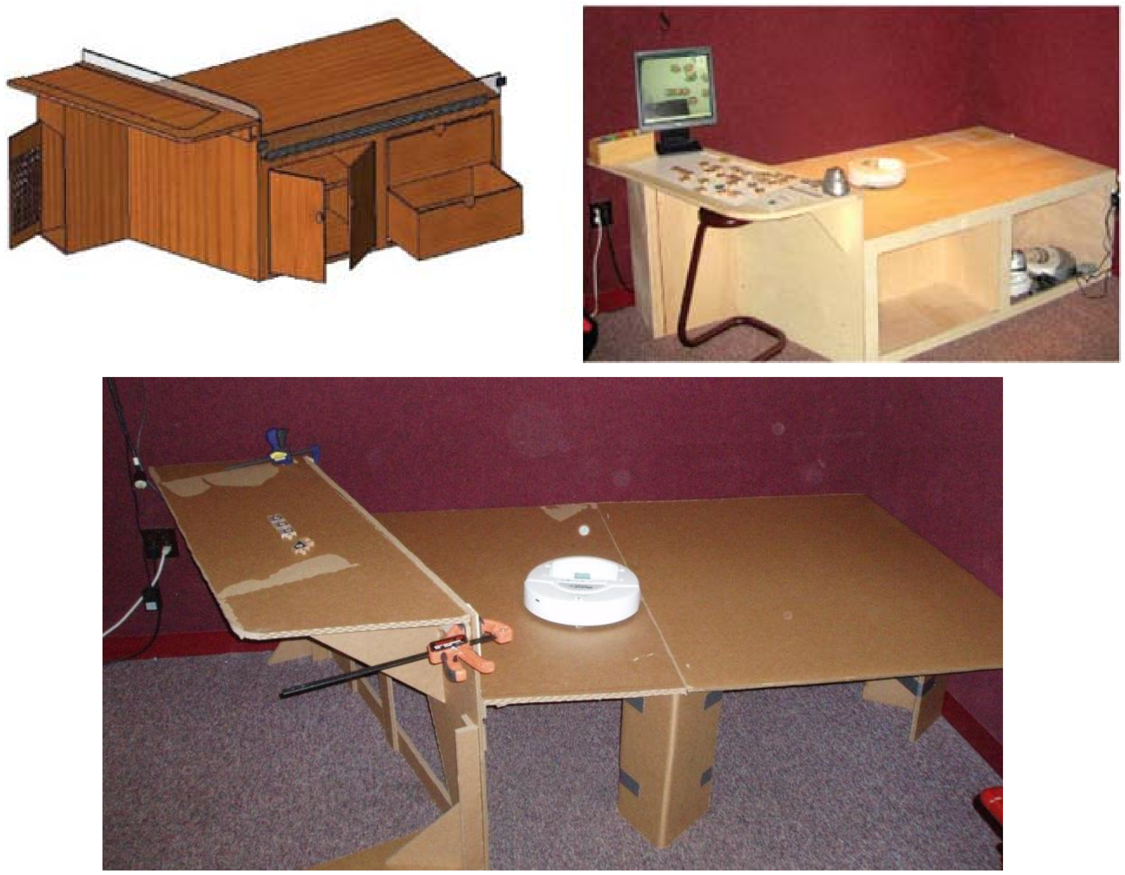


Figure 5.6: Schematic diagram (top left) and plywood prototype (top right) of the Tangible Programming exhibit at the Boston Museum of Science. Initial prototype installations were created out of corrugated cardboard (bottom).

The camera is mounted three feet above the programming surface and captures still images of visitor programs. Each of the programming blocks is imprinted with a TopCode symbol that allows the system to determine the position, orientation, and type of each programming block. I deployed an initial prototype on the museum floor in September 2007 and have gone through many iterative revisions since, fixing technical problems and making improvements based on feedback from the museum staff and visitors.

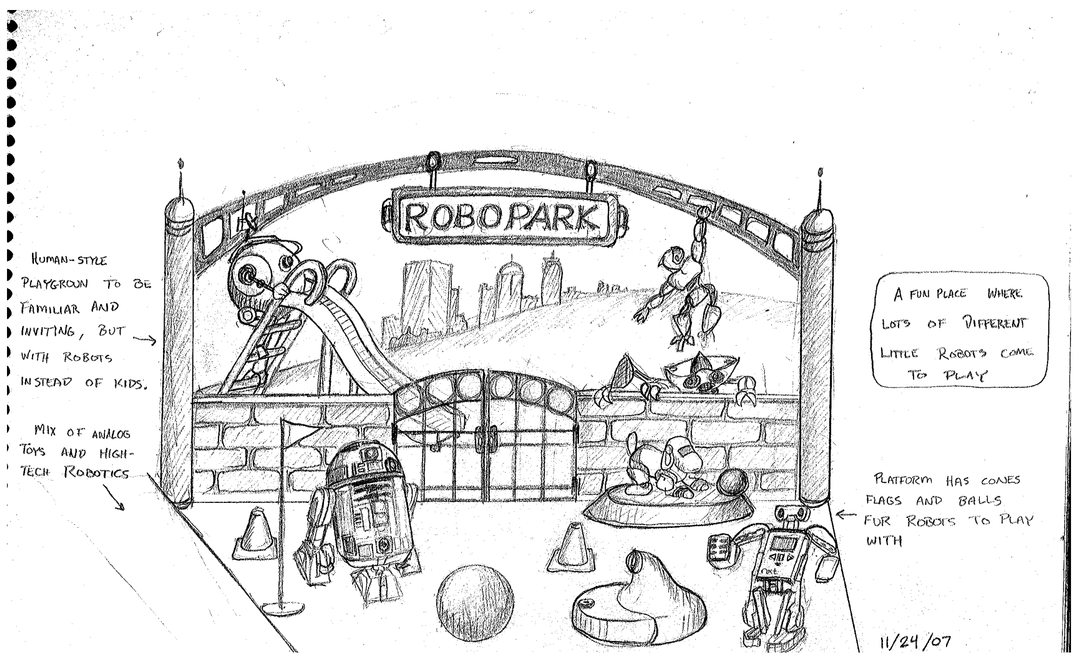

ROBOPARK
HUMAN-STYLE PLAYGROUND TO BE FAMILIAR AND INVITING, BUT WITH ROBOTS INSTEAD OF KIDS.
MIX OF ANALOG TOYS AND HIGH-TECH ROBOTICS
A FUN PLACE WHERE LOTS OF DIFFERENT LITTLE ROBOTS COME TO PLAY
PLATFORM HAS CONES FLAGS AND BALLS FOR ROBOTS TO PLAY WITH
11/24/07

Figure 5.7: A concept sketch for the robot park exhibit theme.

5.0.7 Programming Language

I designed the programming language itself to be as simple as possible while still allowing for interesting interactions. The goal is for a child with no prior programming experience to be able to learn how to use the exhibit in a minute or two. The language syntax is conveyed entirely through a physical jigsaw puzzle metaphor, and, following the lead of systems like PicoBlocks and Scratch (Resnick, 2007), it is impossible to produce a syntax error (see Chapter 2).

5.0.8 Exhibit Computer Logs

I set up the exhibit computer to log anonymous data on visitors' use of the exhibit. These logs captured a digital photograph of the programming surface every time a visitor pressed the compile button. Somewhat accidentally, these photographs also

Total sessions count:	761
Total participant count:	1,263
Total compile count:	4,396
Avg sessions per day:	28.19
Avg participants per day:	46.78
Avg compiles per session:	5.78
Avg participants per session:	1.66
Avg session length:	0:04:20

Table 5.1: This table summarizes the results of a by-hand inspection of the exhibit computer logs for the month of November, 2007.

often captured parts of participants hands and arms as well (see Figure 5.15). By inspecting the photographs by hand, I was able to use this data to make estimates for both the number of visitor sessions and number of individual visitors per session. I analyzed these logs in detail for the entire month of November, 2007. There was much uncertainty involved in this data collection method; however, I believe that the results provide a reliable rough estimate of visitors use of the exhibit. Table 5.1 summarizes these results. In addition, Figures 5.8 and 5.9 show the frequency of session length and participant count across these sessions.

5.1 Design Considerations for TUIs in Museums

Recent literature on science museums includes many practical design considerations for exhibit developers (Allen, 2004; Humphrey & Gutwill, 2005; Serrell, 1996). Some of these design considerations seem especially applicable to educational tangible interfaces. The unrestricted nature of science museums is at once an appealing aspects for visitors and one of the biggest challenges facing exhibit developers (Allen, 2004). Because visitors are free to choose when and where to spend their time (unlike classroom settings), effective exhibits must be both inviting and easy to understand.

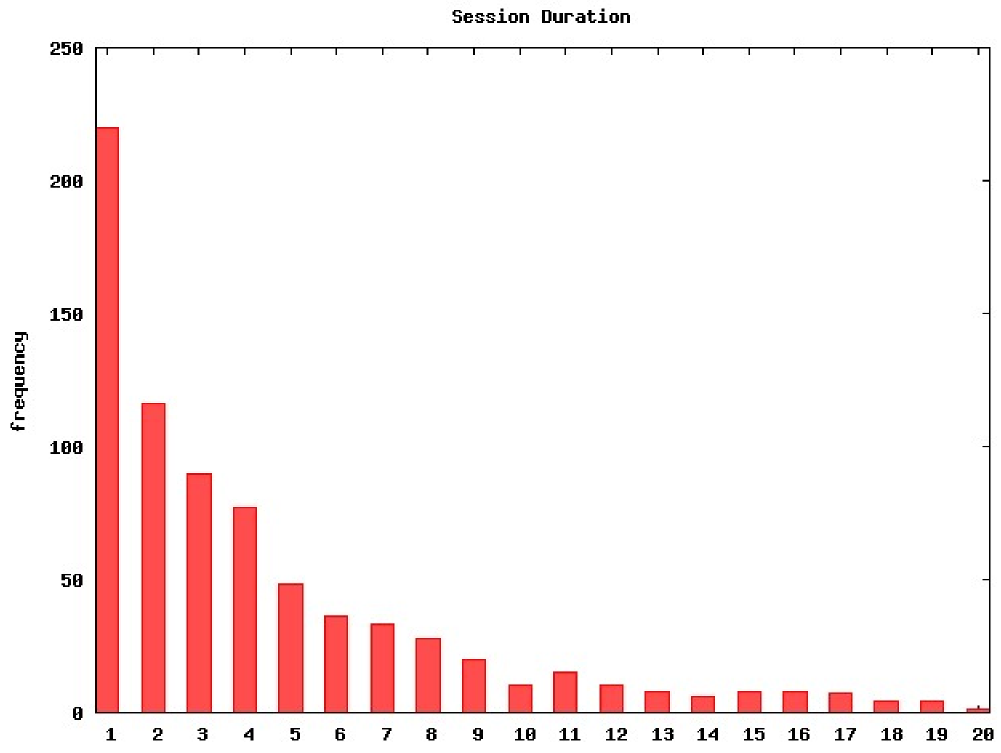


Figure 5.8: Frequency of session length (in minutes) based on an analysis of the exhibit computer logs for the month of November, 2007. Session length is measured as the amount of time a social group spends interacting with the exhibit.

.

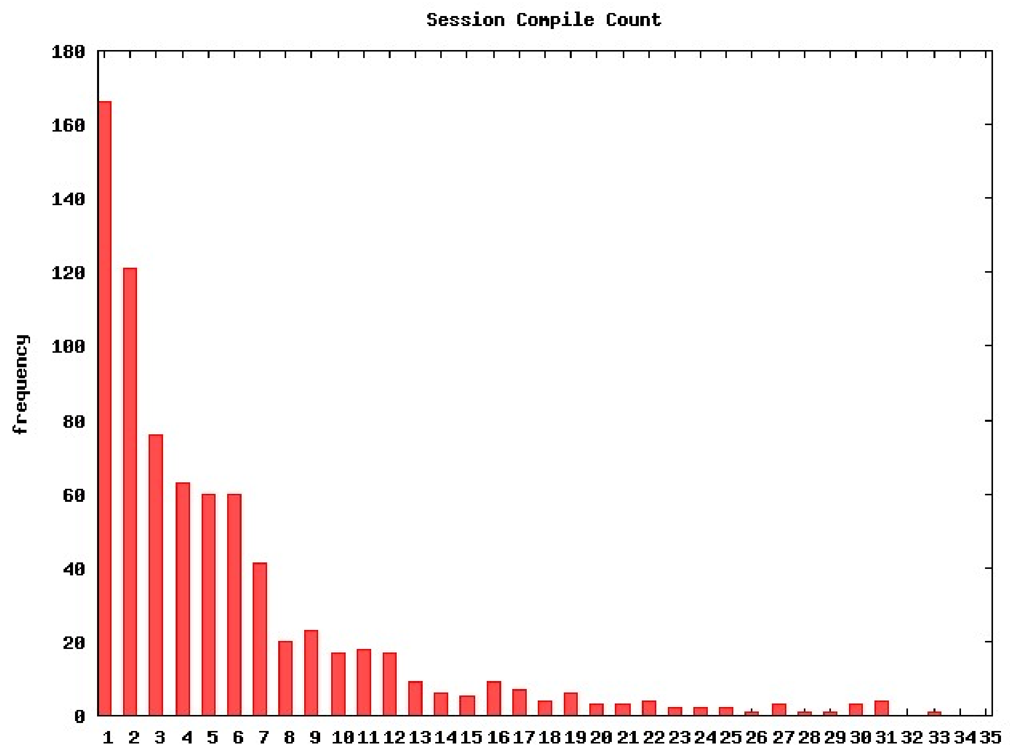


Figure 5.9: Frequency of number of programs compiled per session based on an analysis of the exhibit computer logs for the month of November, 2007.

.

Furthermore, they must hold the visitors' attention and motivation throughout the interaction process. All of these issues must be addressed while keeping development and maintenance costs reasonable. Thus, I highlight five design considerations for tangible user interfaces for children in a museum setting.

- Inviting — Exhibits need to catch the attention of visitors and invite them to interact.

- Apprehendable — Visitors with no prior experience should be able to easily learn how to use an exhibit. Allen, building on Norman's notion of affordances (Norman, 1988), suggests the term *immediate apprehendability* as the quality that "people introduced to [an exhibit] for the first time will understand its purpose, scope, and properties almost immediately and without conscious effort" (Allen, 2004).

- Engaging — An interactive exhibit strives to hold the attention of diverse visitors throughout the exploration process.

- Supportive of Group Interaction — Science museums are usually visited by families and school groups (Allen, 2004; Serrell, 1996) rather than by individuals. As such, exhibits should support social learning and group interaction with both active participants and passive observers (Serrell, 1996).

- Inexpensive and Reliable — Museums are non-profit institutions that rely heavily on support from individuals, corporations, foundations, and government agencies. Thus, keeping development and maintenance costs low is a priority.

5.2 Pilot Study

In January 2008, we conducted a two-phase pilot evaluation of the exhibit, in which we attempted to measure the effectiveness of the exhibit with respect to four of the attributes listed above: *inviting*, *active collaboration*, *apprehendability*, and *engagement*. This section briefly summarizes the results of this evaluation.

5.2.1 Participants

In our first pilot session, we observed 55 museum visitors, 35 male and 20 female. Of these, 23 were children (seven girls and sixteen boys). Seven visitors used the exhibit alone. The remainder of visitors interacted with the exhibit in groups from two to six people. In all we observed 19 interaction sessions.

In our second round of evaluation, we observed 100 museum visitors, 56 male and 44 female. Of these, 43 were children (14 girls and 29 boys). Fourteen visitors used the exhibit alone. The remainder of visitors interacted with the exhibit in groups from two to seven people. In all we observed 35 interaction sessions.

5.2.2 Exhibit Revisions

During the first pilot session, we noticed some usability problems. First, the RFID Block Tester was not functional at the time of our observations, leaving visitors with no way to quickly test the functionality of a block. Second, there was a device on the programming console that is used to activate the robot's infrared sensor.

Visitors commonly confused the infrared device for some sort of robot remote control. Finally, when the robot moved towards the edge of the platform a safety system would terminate the visitors program to prevent the robot from falling over the edge. At this point visitors had to physically pick up the robot and place it back on the platform, although not all visitors realized that their program had been stopped. Also, not all visitors realized they were allowed to touch the robot.

In our next pilot session, we made changes that we thought would improve these factors. First, the Block Tester was implemented, allowing visitors to learn about the blocks with immediate feedback. Second, the infrared device was moved to the side of the workstation, so that it did not distract visitors from learning the basics before moving to the advanced functionality of the infrared sensor. To prevent the confusion with the robot moving near the edge of the platform, we simply placed the robot in the center of the platform between sessions so that users would be less likely to confront this problem before having a chance to learn how the exhibit worked. The goal was to measure improvements in the user experience after making these changes.

5.3 Pilot Study Results

5.3.1 Inviting

78.2% of visitors in the first session (S1) and 79% of visitors in the second session (S2) who noticed the exhibit stopped to interact with it. Among children, 85.7% (S1) and 92.9% (S2) of girls and 87.5% (S1) and 93.1% (S2) of boys who noticed the exhibit stopped to try it. Given that museum visitors have complete freedom to decide whether or not to stop at an exhibit, these results seemed surprisingly high. Unlike in the comparative study, we included both active and passive participants in our tally. If we consider only active participants, then 60% (S1) of visitors who saw the exhibit actively interacted with it. For children the numbers were 71.4% (S1) of girls and 75% (S1) of boys [1].

5.3.2 Active Collaboration

The average group size was 2.47 people in the first session and 2.35 people in the second session. Groups were as large as six people in S1 and seven people in S2. Figure 5.10 shows the frequency of group size for both sessions. Overall, 76.7% (33 out of 43) of participants in the first session were active rather than passive. Qualitatively, groups tended to include both passive and active participants, and group members often transitioned between active and passive roles. Passive group members seemed to be able to easily watch active participants working and frequently contributed suggestions.

5.3.3 Apprehendable

Of the nineteen groups who used the exhibit in S1 only ten (52.6%) successfully learned how to use the exhibit. Of those ten groups almost all figured out the exhibit in a minute or less. Of the remaining nine groups, four gave up in the first minute. The other five groups spent between two and five minutes trying to figure out how the exhibit worked before giving up. These results were discouraging, but we also noticed

[1]We only recorded active vs. passive information in our first session observations.

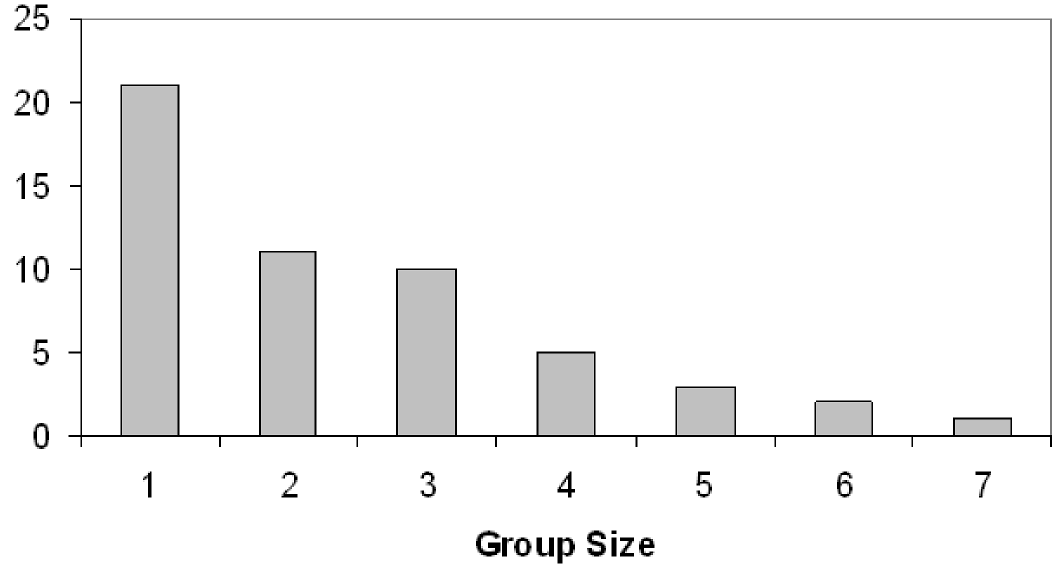


Figure 5.10: Frequency of group size observed in session 1 and session 2 combined.

three specific reasons why visitors were struggling (absence of block tester, confusion by infrared beam, and robot getting stuck near the platform edge) as described above.

In the second session, we made the changes described above. Of the 35 groups who used the exhibit, 24 (68.6%) successfully learned how to use the exhibit. The 24 groups that understood the exhibit took between nine seconds and four minutes to figure out the exhibit, with most figuring it out in about a minute. Of the remaining seven groups, three left the exhibit within the first minute. The rest spent between 1.2 and 4 minutes trying to figure out how the exhibit worked before giving up. Note that in our comparative study this figure was higher in the tangible condition; 29 out of the 39 (74%) groups were able to develop an understanding of the exhibit. This might be because of improvements made to documentation and signage at the exhibit and optimizations made to the computer vision system resulting in faster compile times.

5.3.4 Engagement

We measured engagement in terms of overall session length for each group. Of the ten groups from S1 and 34 groups from S2 that successfully learned how to use the exhibit, engagement was high. The average session length for these groups was 5.3 and 5.0 minutes (S1 and S2, respectively) and the maximum session length was 16 (S1) and 26 minutes (S2). However, for the remaining groups (nine in S1 and twelve in S2), the average session length was 1.9 and 1.3 minutes. This brought the overall average session length down to 3.7 minutes (S1) and 3.8 minutes (S2). These results led us

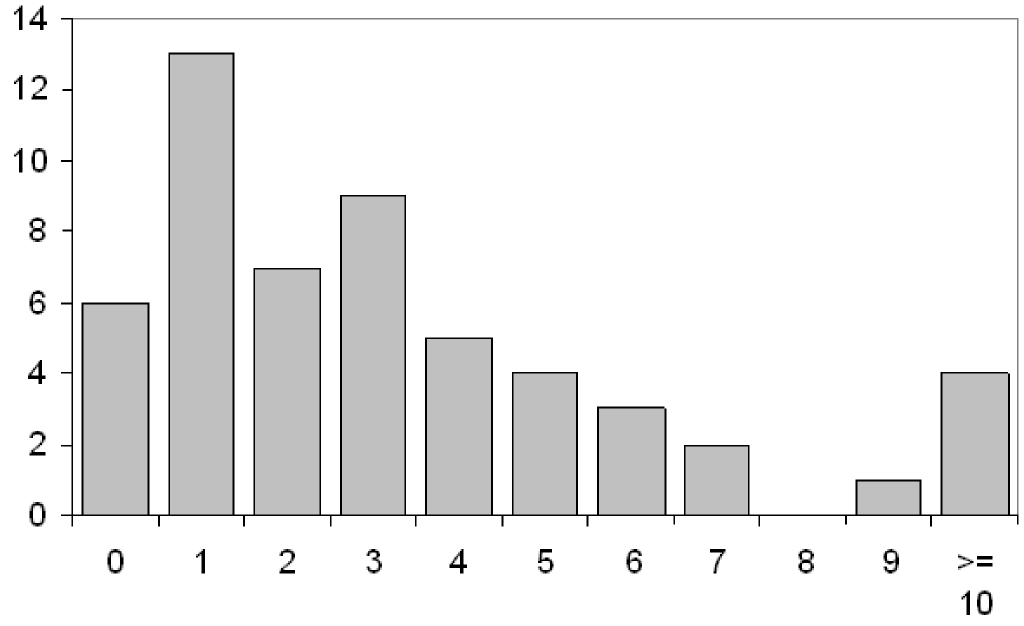


Figure 5.11: Frequency of session duration (rounded to the nearest minute) for S1 and S2 combined. The last column groups all sessions equal to or greater than 10 minutes. The actual session lengths were 10, 14, 16, and 26 minutes.

to believe that if we could further improve the apprehendability of the exhibit overall engagement would improve as well. Indeed, in the comparative study, average session lengths in the tangible condition were more than a minute longer (5.03 minutes). Figure 5.11 shows the frequency of session lengths (rounded to the nearest minute) for the first and second sessions combined. The broad distribution indicates that the exhibit has the ability to draw visitors in to prolonged engagements.

5.4 Interaction Style Comparison Study

Based on our findings from the pilot study, we conducted a between subject study to compare the effectiveness of a tangible and a graphical programming interface for the Robot Park exhibit at the Boston Museum of Science. The study consisted of observations of museum visitors and an analysis of logs generated by the exhibit computer. In order to provide context for our quantitative data, we also interviewed 13 family groups who had used the exhibit. Based on the prior evaluation, we were interested in the following six questions.

5.4.1 Research Questions

Inviting: Does the type of interface affect how likely visitors are to interact with the exhibit?

Apprehendable: Does the type of interface affect whether or not visitors are able to develop an understanding of how the exhibit works? That is, are they able to figure out how to create programs and send them to the robot?

Active Collaborative: Does the type of interface affect how well visitors are able to interact with the exhibit in groups? Does the exhibit support simultaneous active participants?

Engaging: Does the type of interface affect how long visitors interact with the exhibit?

Programs: Does the type of interface affect either the number or the complexity of programs that visitors create to control the robot?

Child-Focused: Does the type of interface affect how children interact with the exhibit and with other members of their social group?

5.4.2 Method

During the evaluation, visitors used the exhibit without interacting with researchers and without help from the museum staff. Quantitative data involving visitor behavior were logged automatically by the exhibit computer as well as manually by the researchers. During periods of observation, researchers sat ten feet away from the exhibit and watched visitors' interactions with the exhibit. A sign was posted next to the exhibit, notifying visitors that the exhibit was being observed, which might have affected visitors' decisions to use the exhibit. For this study, we were interested in observing *family groups*, which we define as groups of individuals consisting of at least one child and one adult who visit the museum together. To increase our chances of observing family groups, we conducted all of our observations over a period of three weeks on weekend days. Visitors were not recruited to participate in the study; rather, participants were people who happened to visit Cahners ComputerPlace at the Museum on one of our observation days. Since visitors were not recruited from the museum population as a whole, we must assume some bias in the sample population. Visitors likely had an existing interest in computers and robotics that motivated them to visit ComputerPlace in the first place. Two researchers collected data for this study. To reduce intra-researcher bias, the two researchers spent roughly equal amounts of time observing the two interface conditions.

5.4.3 Conditions

We observed museum visitors in two independent conditions, tangible and graphical. In the tangible condition (TUI), visitors interacted with the exhibit using the Tern wooden block interface described above. In the graphical condition (GUI), we replaced the wooden blocks with a single standard, two-button computer mouse. To allow for mouse-based programming in the GUI condition, we created a comparable visual programming language (Figure 5.12). We were careful to make the visual and the tangible languages as similar as possible, and we included an identical set of blocks in the two interfaces. In addition, to help ensure an intuitive graphical language, we modeled the mouse-based interaction conventions on the popular Scratch programming language (Resnick, 2007)[2] One important difference between the graphical and tangible languages is that there is no restriction on the number of blocks that can be dragged onto the screen in the graphical interface, whereas, there are a limited number of tangible blocks available to visitors. All other aspects of the physical exhibit installation remained the same. We set up only one interface for visitors to use on a given observation day, and we alternated conditions on subsequent days. Furthermore, the two researchers who collected data for the study spent roughly equal amounts of time observing each condition. We chose to compare the tangible language to a system with a single mouse because this still seems to be the predominant form of interaction in schools, after school programs, and museums. In the future, I would like to broaden the comparison to include multi-touch or multi-mice systems. For now, I will note areas where I think that alternate screen-based systems might be advantageous in my discussion of the results.

[2]This interaction style uses a click-and-drag operation to move programming blocks on the computer screen with the mouse. It is important to note that this type of operation is sometimes difficult for young children, and it has since been pointed out that a click-and-carry operation might have been more appropriate in this situation.

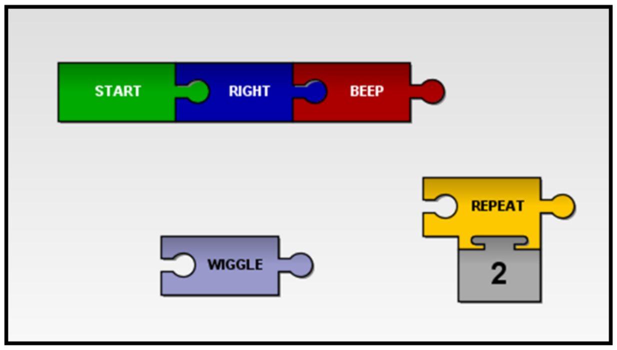


Figure 5.12: We created a graphical programming language equivalent to Tern using the same jigsaw puzzle metaphor and an identical set of blocks.

5.4.4 Participants

We observed a total of 260 individuals at the Museum of Science (108 for the GUI condition and 152 for the TUI condition). Of these, 104 of the participants were children (47 for the TUI condition and 57 for the GUI condition). We defined a child as an individual 16 years old or younger. However, for these observations we did not interview the visitors, so our participant ages are estimates. Based on these estimates, observed children were between 2 and 16 years old (46 were approximately 8 years old or younger; 34 were between the ages of 9 and 12; and 24 were between the ages of 13 and 16). As we mention above, we were interested in observing family groups. In the GUI condition there were 25 total groups, 16 of which contained at least on parent and one child. In the TUI condition, there were 39 total groups, 18 of which contained at least one parent and one child. For the family group interviews, we recruited thirteen child-parent pairs. Of the children, six were girls and seven were boys. The ages ranged from 5-16 years of age, with an average age of 9.

5.4.5 Quantitative Data

To evaluate the exhibit and answer the six questions described earlier, we had to convert each desirable quality to a measurable metric that we could use for comparison of the two interfaces.

To measure the *inviting* quality of the exhibit, we kept a tally of the number of people who noticed (looked or glanced at) the exhibit while within a five foot radius of the installation. Of the people who noticed the exhibit, we recorded the number of people who touched the exhibit with their hands. The time that a visitor first touched the exhibit was recorded as the start of a session. Session data were recorded on a per-group basis.

To measure *apprehendability*, we noted whether or not a social group was able to develop an understanding of how the exhibit worked. In other words, did visitors understand that pressing the run button caused the robot to execute a program? For our purposes, programming the robot one time was not sufficient evidence of understanding. Instead, we required evidence that visitors were purposefully putting pieces together to create more than one program. We recognized that it might be possible for a visitor to understand how the exhibit works without compiling more than one program; however, for the purposes of comparison, we felt that it was more important to avoid false positives than false negatives.

To determine the extent to which the exhibit supports *collaboration*, we compared the number of active participants to the number of passive participants for each interaction session. An active participant is someone who touches or interacts with the exhibit in some way, while a passive participant is a visitor who simply observes other members of his or her social group using the exhibit. Visitors often switch between active and passive roles during an interaction session; however, for this study, any hands-on interaction during a session classified a participant as active, regardless of the nature of the interaction. We recognize that collaboration is a complicated concept with many shades of meaning. Our measure is not comprehensive, but we feel the results are still worth noting.

To measure *engagement*, we recorded the duration of each interaction session. This was recorded as the time the first group member started interacting with the exhibit to the time that the last group member left the exhibit. This method is based on prior studies of engagement with interactive elements in museums (Humphrey & Gutwill, 2005). Like collaboration, we recognize that session length is a narrow definition of engagement, a phenomenon that has intellectual, physical, emotional, and social aspects; however, museum research has shown that session length correlates well with physical, intellectual, and social aspects of engagement (Humphrey & Gutwill, 2005).

To analyze *visitor computer programs* we set up the exhibit computer to log every program compiled by participants. This was in the form of a screen shot for the GUI condition and an image captured by the digital camera for the TUI condition (Figure 5.15). In analyzing these logs we were interested in three questions: does the type of interface affect (1) the number of programs created by visitors per session; (2) the length of programs created; and (3) the complexity of programs created?

To determine the extent to which the exhibit is *child-focused*, we noted which member of each family group initiated an interaction session and whether that person was a parent, a child, or a parent and child together. We also analyzed the data for differences between children and adults for each of the other measures described above.

5.4.6 Qualitative Data

After collecting the quantitative data, we returned to the museum on several additional weekend days to conduct interviews with family groups. In all we interviewed thirteen parent/child pairs who had used the exhibit for more than five minutes on their own before being approached by a researcher. In some cases there was more than one child in a family group that we interviewed. In these cases, we interviewed the child who seemed most involved with the activity. After initially agreeing to participate, each family group was given time to read and sign a consent form. The interviews took about fifteen minutes to complete. Of the children who participated in the interviews, twelve had never programmed before, and one had programmed with LEGO Mindstorms at a camp. All reported that they used a computer at least once a week at school or at home. Four of the groups had used the GUI condition and nine had used the TUI condition prior to participating in the interview.

In the interviews, we gathered background information and impressions from the visitors, and we asked them to work together to complete a few simple programming tasks. We also introduced the visitors to the interface (either GUI or TUI) that they had not been using before the interview. After trying out the second interface, we asked the children to fill out a short questionnaire about the two interfaces. Eleven out of the thirteen children answered these questions, while the two remaining children left before finishing the questionnaire. For children who could not read, we read the questions out loud. Two researchers conducted the interviews. One researcher was responsible for asking the questions and verbally interacting with the participants; the other researcher, meanwhile, recorded transcribed the conversations on a laptop computer. No audio or video recording devices were used. Appendix A provides the questions that we used to conduct the interviews at the Museum.

5.5 Results

This section presents the results of our study organized around our research questions. Namely, the extent to which each interface was inviting, apprehendable, collaborative, engaging, and child-focused. We also analyze the programs created by visitors.

5.5.1 Inviting

Based on our prior formative evaluation of the exhibit, we expected the tangible interface to be highly inviting. We hoped that the use of familiar objects (such as wooden blocks) would transform an unfamiliar and potentially intimidating activity like computer programming into an inviting and playful experience. Our results,

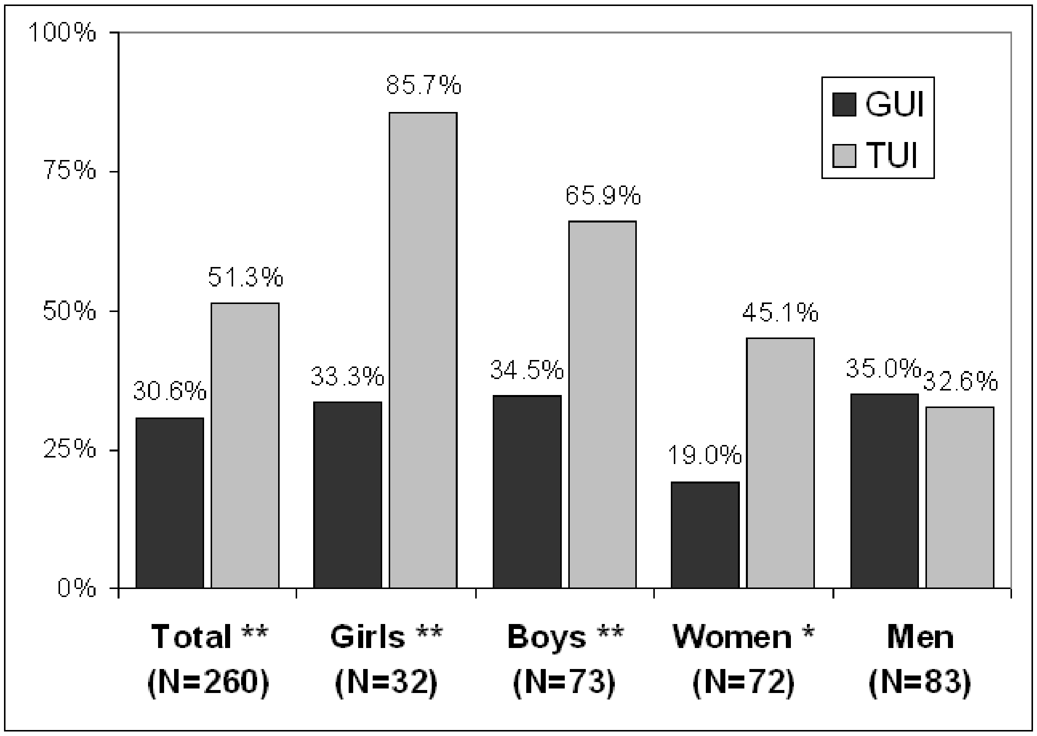


Figure 5.13: Percentage of visitors who interacted with the exhibit after noticing it. (* $p < .05$, ** $p < .01$)

shown in Figure 5.13, indicate that the choice of interface matters a great deal. Overall, visitors were significantly more likely to try the exhibit with tangible blocks than with a mouse (based on a two-tailed z-test). This was especially true for children and for girls in particular. For the graphical system, 33.3% of girls who noticed the exhibit also tried it. This number rose to 85.7% when the tangible system was presented, an increase of over 50%.

To gain some insight into this effect, during the family group interviews, we asked the children which interface they thought was more fun. Seven of the visitors who were interviewed reported that the blocks were more fun than the mouse; one said that the mouse was more fun; and two reported that they were the same (1 subject had no response).

5.5.2 Apprehendable

Before conducting this study, our hypothesis was that the graphical condition would be easier for visitors to understand than the tangible condition. This was because we felt that, in general, visitors would be very familiar with graphical user interfaces. On the other hand, with passive tangible interfaces such as the tangible programming exhibit, there is a non-continuous link between physical actions and digital responses. As such there is a danger that users will have a difficult time understanding how a system is supposed to work. For the tangible programming exhibit, visitors must discover that they have to press a button to trigger the camera to take a picture of their program to send to the robot. One of our primary concerns was that visitors would give up and walk away before figuring out how the exhibit worked.

Despite our expectations, the results of our study showed that there was no significant difference between the two conditions. Of the 25 groups that we observed in the graphical condition, 18 (72%) successfully developed an understanding of how the exhibit worked. In the tangible condition, 29 out of the 39 (74%) groups were successful. Typically, visitors would give up if they were not able to figure out the exhibit within the first two minutes of interaction (only two groups overall spent more than two minutes before giving up).

Correspondingly, out of the family groups we interviewed, four reported that the blocks were easier to use and five reported that the mouse was easier to use (with one reporting that they were the same, and three not responding to this question). Children's age did not appear to play a factor in interface preference.

5.5.3 Active Collaboration

Tangible interfaces are often claimed to improve support for collaborative interaction. We expected that this would be true for our tangible system as well, both because the interface is made up of many physical manipulatives for active participants to share, and because passive participants can easily observe the actions of active participants. Hornecker and Buur refer to these properties as multiple access points and non-fragmented visibility (Hornecker & Buur, 2006). In addition, we were careful to design the exhibit installation so that it would be easy for people standing around the platform to observe both the robot and the programming console.

For the purposes of this study, we define active collaboration as simultaneous active participation, and we measure it by comparing the number of active and passive participants in each group. The average number of active participants per group in the graphical condition was 1.32 (SD = 0.48), while the average in the tangible condition was 2.0 ($SD = 1.05$). This difference is statistically significant (one-tailed t-test, $p < 0.01$). Similarly, the ratio of active to passive participation in the GUI condition was 1.18, while the ratio in the TUI condition was 3.55. Figure 5.14 shows the average number of active participants in each condition by age and gender. The difference between conditions was especially large for children (based on a one-tailed z-test). This result was reflected in our interviews with family groups. More children

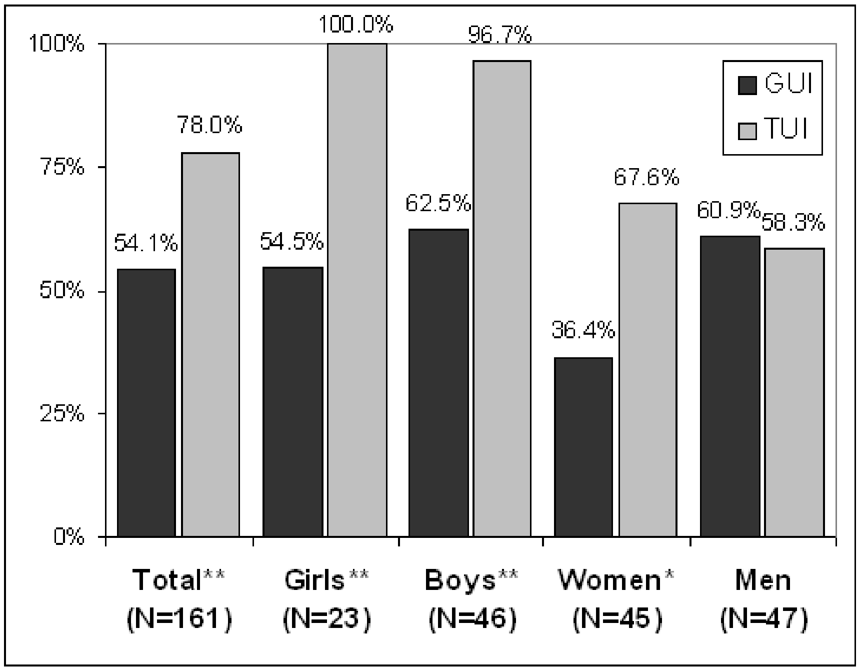


Figure 5.14: Percentage of participants who were active (out of active and passive participants combined) in each condition (* $p < .05$, ** $p < .01$).

said that they would prefer to use the blocks for working with friends or family (7 blocks, 3 mouse, 1 no response). However, the responses were evenly split when asked which interface would be preferable for working alone (4 answered the blocks, 4 the mouse, 1 the same, 1 had no opinion, 1 didn't answer).

	GUI	$(N = 25)$	TUI	$(N = 39)$
Single Active Participant	2.55	$(SD = 2.09)$	1.77	$(SD = 1.82)$
Multiple Active Participants	7.13	$(SD = 7.42)$	6.65	$(SD = 6.47)$

Table 5.2: Average session lengths for groups with a single active participant and groups with multiple active participants for both conditions. The differences within both conditions are statistically significant ($p < 0.05$).

5.5.4 Engaging

We measured engagement in terms of the overall session length for each group. The average session length was 4.02 minutes for the graphical condition ($SD = 4.87$) and 5.03 minutes for the tangible condition ($SD = 5.84$). The variance for session length was high in both conditions, and a two-tailed t-test showed no significant difference between the two means. To put these results into perspective, recent research on engagement in science museums has found average session lengths of 3.3 minutes for successful, engaging exhibits (Humphrey & Gutwill, 2005). We did, however, observe a significant difference in average session length between groups with only a single active participant compared to groups with multiple active participants (see Table 5.2). This suggests that for engagement, the type of interface might be less important than actively involving multiple participants. Thus, multiple mice (see Scott et al., 2000; Stewart et al., 1998), large touch-enabled displays, and tangible blocks might all be equally effective in engaging visitors because they all provide good support for multiple active participants.

5.5.5 Child-Focused

We examine the roles of children and adults under both conditions and provide evidence that the tangible system seems to encourage more active participation on the part of children. For our measure of inviting, we note that children were much more likely to try the tangible interface, whereas for adult men, there was no significant difference between the conditions (Figure 5.13). We also noticed an increase in the percentage of active participation (compared to passive participation) that was especially large for children (Figure 5.14). This data combined with qualitative observations suggests that parents tended to take on more of a supporting role (offering advice and suggestions from the sidelines) rather than an instructional role. For example, during one of the interviews in the GUI condition, a seven-year old girl worked with her father to complete the programming tasks. The father had control of the mouse during this session. When we introduced to the tangible interface, the girl immediately took over the job of creating programs, while her father looked on and offered advice.

Considering data from family group sessions (omitting adult-only or child-only groups), 17 out of 18 sessions in the TUI condition were either initiated by a child or initiated by a parent and child simultaneously. In the GUI condition, on the other hand, 11 out of 16 sessions were initiated by a child or parent and child simultaneously. The remaining sessions were initiated by an adult. This difference is statistically significant using a two-tailed z-test ($p < 0.05$).

5.5.6 Computer Programs

Finally, using the exhibit computer logs, we analyzed the actual computer programs that visitors created during the first two days of observations (one day for each condition). This included 13 groups in the GUI condition and 20 groups in the TUI condition. Prior to conducting the study we hypothesized that visitors would create longer and more complex programs with the tangible interface because we felt that it was easier to manipulate and rearrange physical blocks than to manipulate icons on a computer screen with a mouse. However, despite this hypothesis, we found no significant differences between the conditions.

Number of Programs

The average number of programs created per group in the GUI condition was 4.85 ($SD = 7.09$). The average in the TUI condition was 7.19 ($SD = 5.72$). This difference was not significant. These results include groups that compiled no programs. If we omit the groups with zero programs the averages were closer together (7.88 for GUI vs. 8.39 for TUI).

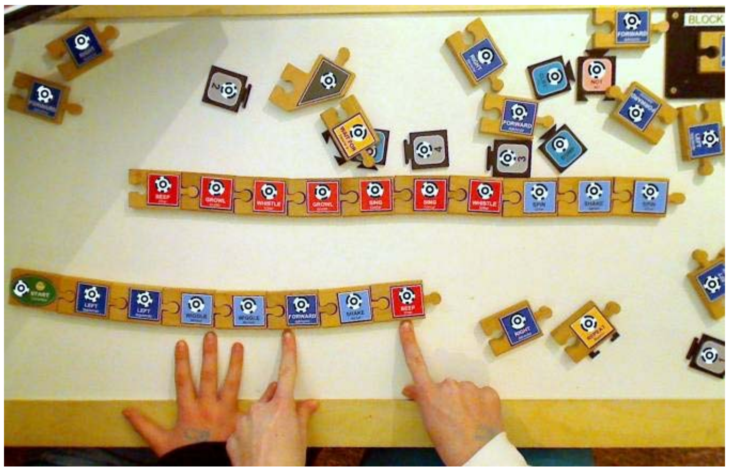


Figure 5.15: An example screen shot from a program created in the tangible condition. The digital photograph has been cropped for clarity. The field of view of the digital camera includes the entire programming surface.

5.5.7 Program Length

We measured program length in terms of the number of blocks (or statements) included. The average program length in the GUI condition was 8.06 statements ($SD = 3.04$), while the average program length in the TUI condition was 9.13 statements ($SD = 5.71$). Again, this difference was not statistically significant.

5.5.8 Program Complexity

To measure complexity, we assigned a complexity score from 1-4 to each program compiled. Programs with a score of 1 contained only action blocks (no flow of control statements). Programs with a score of 2 contained at least one control block but no parameter values, while programs with a score of 3 contained a single control flow block with a parameter value. Finally, programs with a score of 4 contained multiple control flow blocks with parameters. We found no significant differences in complexity levels between conditions.

5.6 Discussion

Overall, on the six measures, the tangible interface was more inviting, more supportive of active collaboration, and more child-focused than the mouse-based interface. We also found that the tangible and graphical interfaces were equivalently apprehendable and engaging, and the resulting visitor programs were not significantly different. For the measure of engagement, we noted that regardless of the condition session times were longer when there was more than one active participant involved. This suggests that other types of interfaces designed to support collaboration (such as multi-touch or multi-mouse systems) might be equally effective in encouraging engagement.

5.6.1 Girls And Programming

In this study, we observed that girls were significantly more likely to use the exhibit in the tangible condition than in the graphical, mouse-based condition. It has been noted that as technology becomes more pervasive in our society, it is important that it supports all members of society (AAUW, 2000; Kelleher et al., 2007). Although women and girls currently are under-represented in the field of computer science (AAUW, 2000; Vegso, 2006), Kelleher et al. (2007) point out that performance and interest in programming has been shown to depend on previous experience programming and time spent programming, rather than on gender. Thus, researchers and educators have been developing methods and tools to motivate girls to learn about computers and to provide them with positive learning experiences with programming. For example, see Storytelling Alice (Kelleher et al., 2007) and Lillypad Arduino (Buechley et al., 2008)). We feel that our results provide strong evidence that tangible programming languages might be another approach to create more gender-neutral computer programming activities for both formal and informal education.

5.6.2 Active Collaboration and Sharability

The tangible condition proved to be more supportive of active collaboration than the graphical, mouse-based condition. Hornecker, Marshall, and Rogers outline rudimentary components of sharable interfaces (Hornecker et al., 2007) that identify many of the key features that we felt contributed to the success of the tangible interface for our exhibit. A sharable interface, in this case, refers to the support for multiple, co-present collaborators around a common task, and it includes components of entry points and access points. Entry points entice people to interact with a system. In our case, the wooden blocks seemed much more effective for this purpose than a computer mouse. This is perhaps because the blocks are familiar and playful objects. But we suspect that this is only part of the story. It could also be that the blocks are non-threatening objects presented in a novel and curious context. The tangible interface was much better at luring people into socially-motivated interaction. With the graphical interface, on the other hand, an observer might feel equally motivated to the activity but also may feel unable to do so without taking the mouse away from the active user. Hornecker et al. (2007) describe the *honey-pot* effect as a social entry point, whereby passive observers are drawn into active participation by watching their family and friends interact with the exhibit. The honey-pot effect, however, is most effective when coupled with easy access to the system for multiple participants as is the case with a tangible interface.

Sharable interfaces also provide access points, which can refer to both perceptual access and manipulative access (Hornecker et al., 2007). The tangible interface is clearly superior in terms of manipulative access because it offers many objects that can be independently manipulated in a meaningful way by multiple participants. We also suspect that the tangible interface provides better perceptual access as well. This is simply because the display space that bounds the interaction is larger and more visible. The tangible interface is manipulated on a large horizontal table top that can be viewed at any angle, while the graphical interface is displayed on a vertical computer screen that can only be viewed from limited angles. Thus, it is both easier for one collaborating active participant to understand the actions of another, and for passive participants to understand the actions of active participants.

Chapter 6

Tangible Kindergarten Project

In her 2008 book, *Blocks to Robots: Learning with Technology in the Early Childhood Classroom*, Marina Bers writes that:

> For decades early childhood curriculum has focused on literacy and numeracy, with some attention paid to science, in particular to the natural world-insects, volcanoes, plants, and the Arctic. And, while understanding the natural world is important, developing children's knowledge of the surrounding man-made world is also important. This is the realm of technology and engineering, which focus on the development and application of tools, machines, materials, and processes to help solve human problems [...] We live in a world in which bits and atoms are increasingly integrated. However, we do not teach our young children about this. In the early schooling experiences, we teach children about polar bears and cacti, which are probably farther from their everyday experience than smart faucets and cellular phones (Bers, 2008).

There are several reasons that might explain the lack of focus on technology in early childhood, but two of the most common claims are that young children are not developmentally ready to understand complex and abstract concepts such as computer programming, and that there is a lack of technology with age-appropriate interfaces to engage very young children in the role of designers and programmers of their own technologically-rich projects. Tangible Kindergarten is a three year research project funded by the National Science Foundation that seeks to address these issues, in part through the use of tangible programming languages kindergarten classrooms. Tangible Kindergarten is a collaboration between the Tufts University Human-Computer Interaction Laboratory in the Computer Science Department and the Tufts University Developmental Technologies Research Group in the Child Development Department. The project has three complementary goals: 1) to develop in-depth, age-appropriate curriculum on computer programming and robotics; 2) to develop age-appropriate technology ready for use in classrooms; and 3) to document young children's ability to learn powerful ideas from computer programming and robotics when given access to appropriate curriculum and technology. In this chapter I will describe work completed in the first six months of this project, including a pilot study in kindergarten classrooms. This work shows how Tern can also be used in formal learning settings.

6.0.3 Developmentally Appropriate Practice

Our work is rooted in notions of developmentally appropriate practice (DAP), a perspective within early childhood education concerned with creating learning environments sensitive to children's social, emotional, physical, and cognitive development. Developmentally appropriate practice (DAP) is a framework produced by the National Association for the Education of Young Children (NAEYC) that "out-lines practice that promotes young children's optimal learning and development" (for the Education of Young Children, 2009). Developmentally appropriate practice is based on theories of child development, the strengths and weaknesses of individual children uncovered through authentic assessment, and individual children's cultural background as defined by community and family (Bredekamp & Copple, 1997).

In part, DAP is built upon the theory of developmental stages introduced by Jean Piaget. This theory suggests that children enter the concrete operational stage at age six or seven. According to Piaget, at this age, a child gains the ability to perform mental operations in her head and also to reverse those operations. As a result, a concrete operational child has a more sophisticated understanding of number, can imagine the world from perspectives other than her own, can systematically compare, sort, and classify objects, and can understand notions of time and causality (Richardson, 1998). Based on this developmental model, one might make the argument that a child's ability to program might be predicted by his or her general developmental level, and, that by extension, a pre-operational kindergartener, typically five years old, may be too young to benefit from or understand computer programming. However, since its introduction, various inconsistencies have been identified with Piaget's stage model. For example, studies have shown that when a task and its context are made clear to children, they exhibit logical thought and understanding well before the ages that Piaget had suggested as a lower limit (Richardson, 1998).

In the early days of personal computing, there was lively debate over the developmental appropriateness of computer technology use in early elementary classrooms. Today, however, the pressing question in no longer whether but how we should introduce computer technology in early elementary school (Clements & Sarama, 2002). For example, a 1992 study found that elementary school children exposed to exploratory software showed gains in self-esteem, non-verbal skills, long-term memory, manual dexterity, and structural knowledge. When combined with other non-computer activities, these students also showed improvements in verbal skills, abstraction, problem solving, and conceptual skills (Haugland, 1992). Other studies have demonstrated that computer use can serve as a catalyst for positive social interaction and collaboration (Clements & Sarama, 2002; Wang & Ching, 2003). Of course, the developmental appropriateness of the technology used by young children depends on the context—what software is being used, and how is it integrated with and supported by the broader classroom curriculum. Many examples of unproductive or even counterproductive computer use in classrooms have been documented (see Oppenheimer, 2003).

6.1 Tangible Programming Curriculum for Kindergarten

Research has shown that mere exposure to computer programming in an unstructured way has little demonstrable effect on student learning (Clements, 1999b). For example, a 1997 study involving the visual programming language, KidSim, found that elementary school students failed to grasp many aspects of the language, leading the authors to suggest that more explicit instruction might have improved the situation (Rader et al., 1999). Therefore, an important aspect of our work is to develop curriculum that utilizes tangible programming to introduce a series of powerful ideas from computer science in a structured, age-appropriate way. The term *powerful idea* refers to a concept that is at once personally useful, interconnected with other disciplines, and has roots in intuitive knowledge that a child has internalized over a long period of time (Papert, 1991). Our goal is to introduce these powerful ideas in a context in which their use allows very young children to solve compelling problems. This goal relates to our hypothesis that children will not only be able to understand these ideas, but that they will also be *powerful* in the sense that they are personally useful and interconnected with other disciplines.

Table 6.1 shows example powerful ideas from computer programming that we have selected to emphasize in our curriculum through tangible programming.

Powerful Idea	**Description**
Computer Programming	This is the fundamental idea that robots are not living things that act of their own accord. Instead, robots act out computer programs written by human beings. Not only that, children can participate in this process and create their own programs to control real robots. Levy and Mioduser point out that there is power in being able to think of a robot both as a "living" autonomous agent capable of acting independently and as a technical construction composed of human-designed components (Levy & Mioduser, 2008).
Command Sequences and Control Flow	The idea that simple commands can be combined into sequences of actions that will be acted out by a robot in order.
Loops	The idea that sequences of instructions can be modified to repeat indefinitely or in a controlled way.

Sensors	The idea that a robot can sense its surrounding environment through a variety of modalities, and that a robot can be programmed to respond to changes in the environment.
Parameters	The idea that some instructions can be qualified with additional information that changes their behavior.
Branches	The idea that one can ask a question in a program, and, depending on the answer, instruct a robot do one thing or another.
Subroutines	The idea that a set of instructions can be treated as a single unit that can be invoked from other parts of a program.
Linking and Libraries	The idea that programs are built upon libraries of other programs and commands that can be created by other people.

Table 6.1: Powerful ideas from computer programming and robotics that we emphasize in our curriculum through the use of tangible programming.

Based on these powerful ideas we developed a preliminary, 8-hour curriculum unit for use in kindergarten classrooms (children ages 5–6) that introduces a subset of these concepts through a combination of whole-class instruction, structured small-group challenge activities, and open-ended student projects. We piloted this curriculum in five kindergarten classrooms in the Boston area, using the Tern programming language in conjunction with educational robotic construction kits. In our most recent rounds of testing (conducted in the fall of 2008), we provided students with pre-assembled robot cars to teach preliminary computer programming concepts. These cars were built using LEGO Mindstorms RCX construction kits. As the unit progressed, students disassembled these cars to build diverse robotic creations that incorporated arts and craft materials, recycled goods such as cardboard tubes and boxes, and a limited number of LEGO parts. Here I provide a brief overview of the activities that we included in the curriculum:

1. **Robot Dance**: In this introductory activity, we introduce students to the concept of robots and programming. Children participate in a whole-class activity in which they use the tangible blocks to "teach" a robot to dance the Hokey-Pokey.

2. **Experimenting with Programming**: Students work in small groups to create and test several programs of their own design. At this point they combine simple command blocks (no sensors or control flow blocks yet) into sequences of actions that are acted out by the group's robot. Robots can perform simple movements, make sounds, blink a light, and perform dance moves (such as shake or spin around).

3. **Simon Says**: In this whole class activity, students pretend that they are robots and act out commands presented by the teacher. For example, the teacher holds up a card that says SHAKE, and the students all shake their bodies. The point of this activity is to help students, who may or may not be able to read, learn the various programming commands that are available to use. As this activity progresses, the teacher will begin to display two or more cards at the same time that the students will act out in order. This activity is repeated on several occasions throughout the unit.

4. **Hungry, Hungry Robot**: In this challenge activity, we introduce the idea of loops as students work in small groups to program their robot to move from a starting line (a strip of tape on the floor of the classroom) to a finish line (marked with a picture of cookies). Children are prompted with the challenge: Your robot is very hungry. Can you program it to move from the start line to the cookies? As this activity progresses, students quickly realize that the robot needs to move forward several times in succession to reach its goal. Because they are only provided with a small number of blocks to move the robot forward, they learn to create programs that incorporate a simple counting loop.

5. **If you're happy and you know it**: The teacher reads a story in which different animals show that they are happy in different ways. Students are then asked to work in small groups to program a robot to act like one of the animals in the story.

6. **Bird in a Cage**: In this whole class activity, the teacher tells a story about a bird that sings when it is released from a cage. The "bird" is a robot with a light sensor light sensor attached. When the bird is removed from a cardboard box (the cage), the increase in light intensity triggers it to sing. The teacher then introduces the concept of sensors and shows the class how to write the program that causes the bird to sing. This activity is repeated on two separate occasions, once as a whole-group activity, and once as a small group challenge activity.

7. **Final Projects**: Students work in teams of two to design and build a robot animal that includes one motor and one sensor. Students are encouraged to create a story that involves their robot animal. This activity is open-ended and student projects are completed over a period of several days.

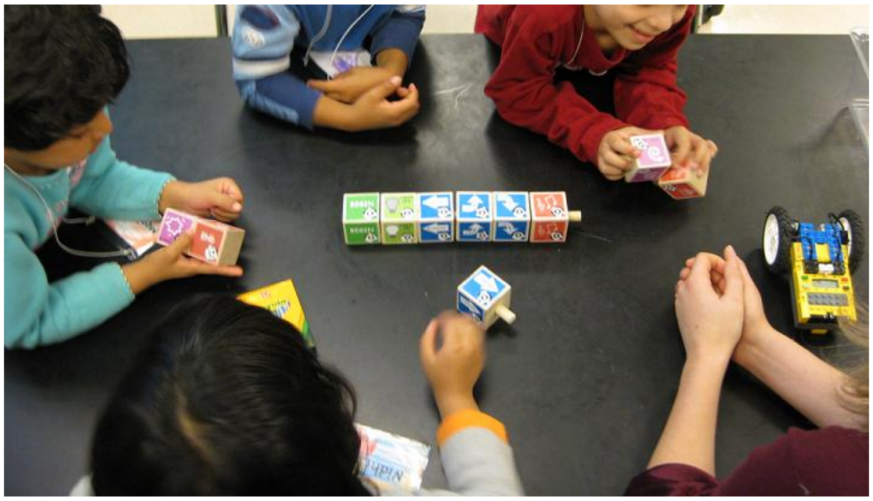


Figure 6.1: This photograph shows a group of children using the tangible Tern language to program a LEGO Mindstorms robot during the Tangible Kindergarten pilot study

6.2 Research Questions

The research presented in this chapter is guided by the following questions: given access to appropriate technologies, are young children capable of programming their own robotics projects without direct adult assistance? At the same time, can young children understand the underlying powerful ideas behind computer programming? Finally, should we revisit assumptions about what is developmentally appropriate for young children when designing curriculum—can we leverage new interactive technologies to introduce more complex concepts?

These ideas are not new. Researchers at MIT's Lifelong Kindergarten Group have suggested that with new digital manipulatives children are capable of exploring concepts that were previously considered too advanced, in part because of the ability of technology to bring abstract concepts to a concrete level (Resnick et al., 1998). However, for the most part, this claim has not been empirically validated with children as young as five years old. And, as noted above, it can be difficult in this research

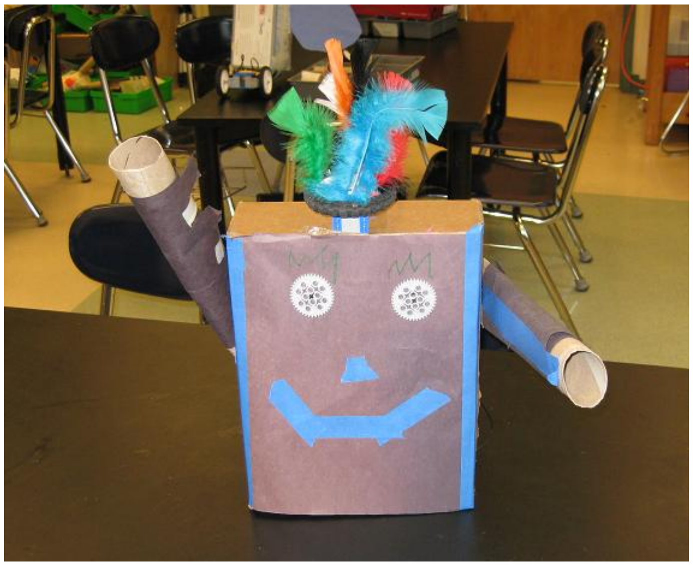


Figure 6.2: This photograph shows one of the student group's final project, a robot consisting of LEGO parts, arts and crafts materials, and recycled goods.

to distinguish what young children truly understand and can do on their own, versus what was done by the supporting adults (Cejka et al., 2006). In the end, the functioning robotic projects tend to hide the challenges faced by children in the process of making them.

6.3 Pilot Study

We have conducted a two-year design-based research study exploring the use of tangible programming technology in kindergarten classrooms. Both our curriculum and the underlying technology have evolved as a result iterative prototyping and testing. This research was conducted at two public schools in the greater-Boston area, one urban K-8 school and one suburban K-5 school. In all, we have worked with 93 children (ages 5–7) and ten teachers in five classrooms. Data were collected through observation notes, photographs, and video tape. We also collected student work (both the programming code as well as the robotic artifacts) and conducted one-on-one interviews with children and teachers in the classroom. Three researchers were involved in this study: two collected observation notes, photographs, and video recordings, while the third acted in the role of *lead teacher*. In the future, we hope to conduct research in classrooms with actual teachers leading the activities.

6.4 Participants

I conducted the first intervention in the spring of 2007 in a kindergarten / 1st grade classroom with 20 children (ages 5–7) at the urban school in Boston. This intervention consisted of two whole-class activities in which I introduced concepts of robotics and programming with blocks, and then three follow-up sessions with small groups of children. These activities took place over the course of five weeks. Afterward, I made substantial revisions to the technology, moving from the jigsaw puzzle design used at the Museum of Science to the interlocking cube design described above. The Tangible Kindergarten team also revamped and expanded the curriculum. Our second intervention consisted of a 8-hour curriculum module in four classrooms at the suburban school with 73 children (ages 5–6) and eight teachers. In order to understand better the implications of tangible versus graphical programming in kindergarten, we divided these four classrooms into two groups. Two of the classrooms used the tangible programming blocks (TUI) described above, and two of the classrooms used a comparable graphical programming language (GUI) similar to what was used in the Museum of Science study (see Figure 6.3). Table 6.2 summarizes the five classrooms involved in this study. All classes at the suburban school were taught in the school's science activity room. In this room there were four desktop computers in the back corner of the classroom for students to use. In addition, there was an LCD projector

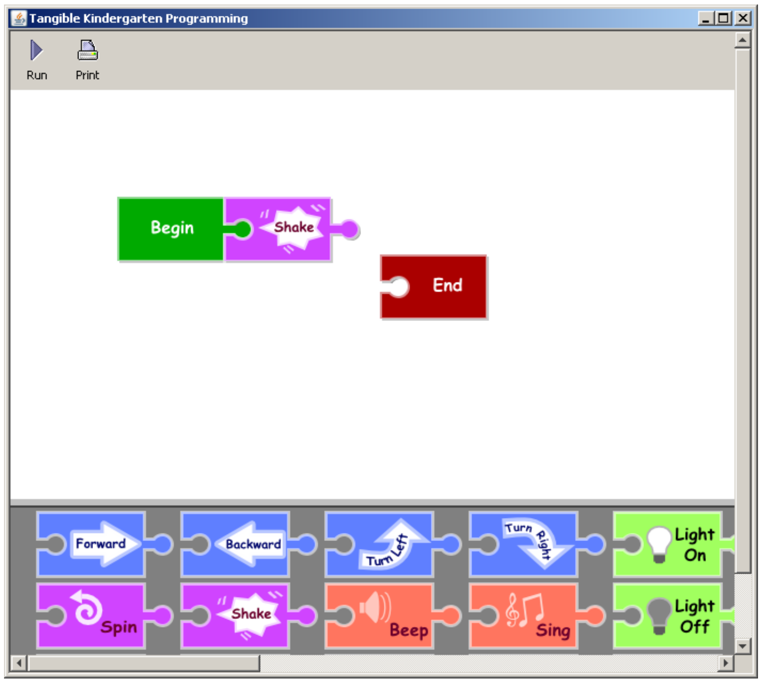


Figure 6.3: Two classrooms in our pilot intervention used a graphical programming language, comparable to the tangible programming blocks. The graphical and tangible languages offer identical statement sets. However, unlike the interlocking cubes of the tangible language, the graphical language uses a visual jigsaw puzzle metaphor.

in the front of the classroom. For whole-class activities in the graphical condition, we projected the graphical programming language interface onto a wall of the classroom. In the tangible condition, no projector was used.

School	Condition	Semester	Hours	Time of Day	Girls	Boys	Robots
Urban	TUI	Spring'07	4 hours	Afternoon	–	–	iRobot Create
Suburban	TUI	Fall'08	8 hours	Morning	11	7	LEGO RCX
Suburban	GUI	Fall'08	8 hours	Morning	11	9	LEGO RCX
Suburban	TUI	Fall'08	8 hours	Afternoon	11	7	LEGO RCX
Suburban	GUI	Fall'08	8 hours	Afternoon	8	10	LEGO RCX

Table 6.2: Classroom details for the Tangible Kindergarten pilot study. Note: I do not have a gender data for the twenty children from the urban school.

6.5 Results

In this study, we had two primary hypotheses. First, given access to appropriate technology, young children are capable of programming their own robotics projects without direct adult assistance. Second, children are able to understand the underlying powerful ideas from the domains of computer programming and robotics through the curriculum.

Based on observation notes and an analysis of video tape, we found that children were able to easily manipulate the tangible blocks to form their own programs. For children in the GUI condition, however, we observed a range of capabilities in terms of being able to manipulate the computer mouse. For many children the process of building graphical programs was tedious. One girl, for example, used the mouse to construct a program consisting of 11 statements in three minutes and eleven seconds (a rate of 17.4 seconds per statement)[1]. As she worked, it often took her two or more tries to successfully connect a new statement to the program. That said, she did not exhibit any recognizable signs of frustration or boredom, and she worked diligently until she had finished her program. A boy in the other GUI class started out almost completely incapable of dragging icons from the statement menu to the main screen. However, after practicing for several minutes, he discovered that by holding the mouse sideways he could use his thumb to press and hold the left mouse button for click+drag operations (see Figure 6.4). This technique greatly improved his ability to create programs. Other children had little to no problem using the mouse to create graphical programs.

On major limitation of the mouse-based interface is that it requires the use of click

[1]To put this in perspective, a girl working in the TUI condition on the same day constructed a program consisting of 11 blocks in one minute and five seconds (a rate of 5.9 seconds per statement)

and drag operations to place icons on the computer screen with the mouse. Changing the interface to allow for click and carry operations would likely make it much easier for young children to participate with the mouse-based interface.

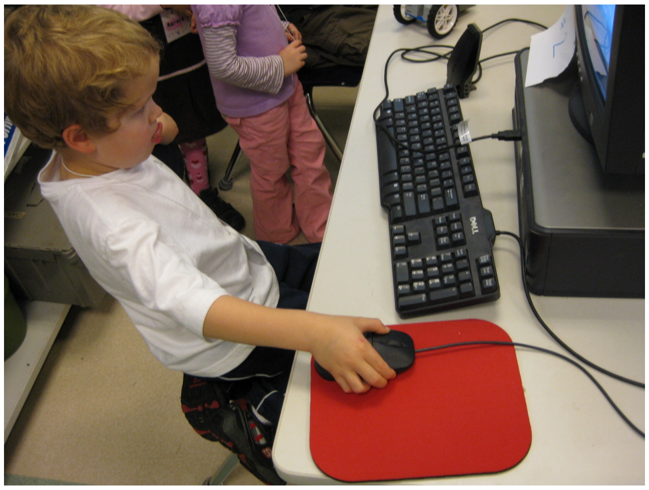


Figure 6.4: One boy in the graphical condition struggled to use the mouse at first, until he discovered that if he held the mouse sideways he could use his thumb to click and hold the left mouse button. This greatly improved his ability to create programs.

In terms of being able to understand the syntax of the programming language, we found that students, for the most part, were able to differentiate the blocks and discern their meanings in both the GUI and TUI conditions. While not all of the children could read the text labels on the blocks, we saw evidence that children who could not read the blocks were able to use the icons as a way to interpret meaning. For example, in the initial sessions, we asked children to look at the blocks and describe what they saw. Some children were simply able to read the text labels on the blocks. Other children said things like: "mine says arrow" or "mine says music." In reality the text on the blocks reads "Forward" and "Sing." We used the Simon Says activity to reinforce the meaning of each block in terms of the robot's actions.

The children also seemed to understand that the blocks were to be connected in a sequence and interpreted from left to right. For example, one student in an introductory TUI session pointed out, “they connect; some have these [pegs] and some have these [holes].” Another student added, “the End block doesn’t have one [peg],” with another classmate adding, “the Start block doesn’t have a hole.” Likewise, in the GUI classes, we saw no evidence that children were confused by the visual jigsaw puzzle metaphor (see Figure 6.3). The children also seemed to understand that the icons should be connected from left-to-right on the computer screen.

During the second session, researchers asked individual children to describe their programs. One girl explained, “my program is: Begin then Beep then Forward then Sing then End.” Here she was reading the text labels on the blocks in her program from left to right, resting her finger on each block in turn. The use of the word *then* suggests that she might have understood not only the implied left-to-right ordering of the blocks but also the temporal step-by-step sequence in which the blocks would be executed. Understanding the idea of sequencing is powerful, not only in the domain of computer programming but for most analytical activities that children would encounter in schooling as well as life.

In the second intervention round, we moved beyond programs consisting of simple sequences of actions and introduced more advanced constructs such as loops, sensors, and numeric parameter values. Through these activities we found evidence that children could engage these concepts, reasoning about possible outcomes of different blocks and forming and testing solutions to challenge activities. For example, in our *Hungry, hungry robot* activity, we prompted teams of four students with this challenge: Your robot is hungry. Can you program it to drive to the cookies? We had placed the robot on a starting line, represented by a strip of tape on the floor. The cookies were represented by a picture taped on the wall approximately four feet from the starting line. Groups were provided with a small collection of blocks, including two Forwards, a Repeat, an End Repeat, and a number parameter block. The number parameter block was labeled with the numbers 2 through 5, with one number on each face of the cube. This number block could be included in a program after a Repeat block to specify how many times a robot should repeat a sequence of actions. Each Forward block would cause the robot to drive forward approximately one foot. So, for example, one solution to the challenge would be the following program:

$$\text{Begin} \rightarrow \text{Repeat (4)} \rightarrow \text{Forward} \rightarrow \text{End Repeat} \rightarrow \text{End}$$

Many other possible solutions exist, and students could add creative elements to their programs. For example, one group included a Sing block to indicate that the robot was eating its cookies after it arrived at the wall.

Many groups started this challenge by creating programs that consisted of many blocks arranged in a seemingly random order. However, after testing these programs, most groups quickly decided that the Forward block was necessary to drive the robot towards its goal. In the graphical conditions, some groups experimented by including different numbers of Forward blocks in their programs. The lead teacher (one of the researchers from the Tangible Kindergarten project) then introduced the concept of a Repeat loop. After the challenge activity, the lead teacher discussed students' solutions with the entire class. In her journal, she described this experience:

> After giving the students some time to work in their groups, I brought them back to the rug to share what they found. One group had determined that they needed 8 forward blocks. After learning the repeat syntax, they realized that the numbers only went up to 5, so they needed 3 more forward blocks. The program they made was:

Repeat (5) → Forward → Forward → Forward → End Repeat

> When we tested to see if that program would work, the students saw that the robot actually went forward 15 times. One student noticed that 15 was a "5 number," and another said it was the 3rd "5 number." This was very exciting for me, since I saw the beginnings of multiplication.

In the TUI condition[2], on the other hand, groups had only two Forward blocks to work with, so we introduced the Repeat concept earlier. As with the GUI condition, groups were able to successfully use the Repeat block in conjunction with the Forward block to achieve the desired outcomes. However, as this was their first encounter with the Repeat block, groups required prompting and instruction from adults.

Later in the unit, we introduced students to the idea of sensors through the *Bird in the cage* activity. Here the cage was a cardboard box and the bird was a robot with a light sensor attached. The lead teacher told a story of a bird who liked to sing when released from her cage. We used this program to have the robot act out the story.

Begin → Repeat(forever) → Wait for Light → Sing → End Repeat → End

The students were curious how the robot was able to sing when it was removed from the cage. The teacher used this curiosity to prompt students to explore the idea of sensors and to develop hypotheses about how the robot is able to sing when it emerges from the box. Finally, the teacher demonstrated how she had created the program that controls the robot. The following transcript is from one of the TUI condition classrooms:

[2]Only one of the classes in the tangible condition participated in this activity.

Teacher:	Do you recognize all of the blocks?
Student 1:	No. The moon one. [Wait for Dark]
Student 2:	The sun and moon [Wait for Light]
Teacher:	Can one of you read this?
Student 2:	Wait for light
Student 1:	It means you're going to wait for the dark to come.
Teacher:	What are these?
Students together:	Repeat
Teacher:	What do they do?
Student 3:	Start all over again.
Teacher:	The bird is outside.
Student 2:	The witch catches the bird
Student 1:	If we turn this block over we could make him sing when it's dark outside. It might be shy so he sings in the dark.

Here the child was pointing out that it would be possible to use a *Wait for Dark* block instead of a *Wait for Light* to achieve the opposite effect, a bird that sings only when it is inside the cage[3]

For the remainder of the curriculum unit, the children worked in pairs to create robotic creatures that ideally incorporated one moving part and one sensor. The children struggled with many aspects of this activity and required substantial help from teachers. The lead teacher described her interactions with one group:

> *I worked with M. and N. for the majority of the time. They struggled with understanding that the RCX [LEGO computer] was what would power the*

[3]Alternating sides of the same cube had *Wait for Light* and *Wait for Dark* labels.

robot. M. spent most of the time searching for materials, and I sat with N. trying to understand her plan for their robot (a parrot that snaps it's beak). She had an egg carton, and was decorating it with stickers. When I asked her where the beak was, she pointed to the egg carton. When I asked her how it would move, she picked up two large pompoms and said that the motors (the pompoms) would make it move. N. had little interest in the RCX and how it worked. When pressed, she said that the buttons and the wires would make the beak move...

Here the students seemed oblivious to basic robotics concepts such as motors and power supplies. In retrospect, we should have expected this type of misunderstanding because, until this point in the intervention, the children had been working with pre-assembled cars. Our curriculum did not include time for children to explore the RCX brick and understand its role in robotic constructions.

As part of the final project presentations we asked students not only to demonstrate their robots, but also to show the class the blocks they used to program it. In many cases, students selected blocks that had little to do with their actual programs. I believe that in the current version of the curriculum, students were not given enough time to experiment with programming on their own, either individually or in small groups. In many cases, the programs were loaded quickly by the teacher in order to finish projects in time for the final presentations. This might explain some of the difficulty students had recreating their programs during the final project presentations.

In other cases, however, students were able to recreate programs more accurately. For example, in this transcription, two girls described their toucan robot:

Teacher: So what does it do?

Student 1:	We have to make the program first
Student 1:	[Starts connecting blocks] Begin. Wait for Dark...
Teacher:	Can you tell us what we're doing while you're doing it?
Student 2:	Yes. I think we should use a repeat block
Student 1:	Great idea
Teacher:	E., why don't you show us your robot while S. builds the program.
Student 2:	This is our robot... [she starts to demonstrate how it works]
Student 1:	The program isn't done yet!
Student 2:	[Looking at the blocks] It's supposed to repeat the shake
Student 1:	Yes. It's going to repeat the shake over and over again.

[The program student 1 has created is:]

Begin → Wait for Dark → Beep → Turn Left → Repeat → Shake → End]

Student 2:	[Runs the program on the robot for the class to see. The actual program waits for dark, turns, beeps, and then shakes once.]
Student 1:	These are the three blocks we used [She puts these blocks in the middle of the rug: Begin, Wait For Dark, Beep]

Here it is clear that there is some confusion on the part of the students about the concept of a program being stored on the RCX robot. However, the blocks the students chose to explain their robot are consistent with the program that they had actually loaded on the robot earlier in the day. Moreover, they seemed to understand the notion of repeating an action. And, although there was no verbal evidence, the first student seemed to understand the notion of waiting for dark given her correct inclusion of that block in her program.

6.6 Discussion

The purpose of these pilot activities was to test the technical feasibility of the project and to develop an intuitive sense for young children's conceptual capabilities with respect computer programming. These results are preliminary and should be interpreted with caution; however, we saw some evidence to support our two hypotheses. Namely, children were able to program their own robots without direct adult help, and children were able to understand some of the powerful ideas from computer programming and robotics introduced through the curriculum.

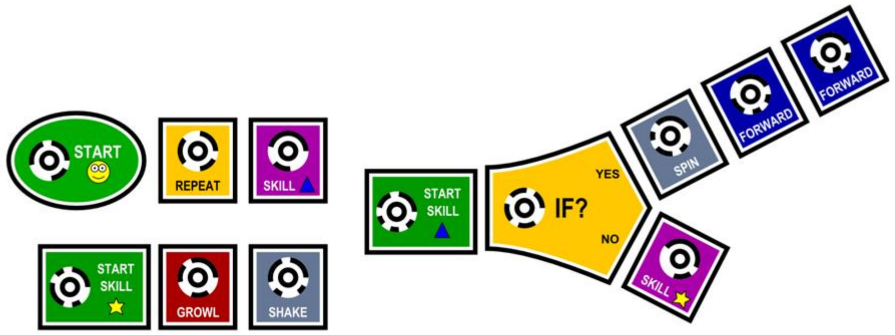
In terms of the first hypothesis, our work with both a tangible and a graphical condition suggests that while many children do struggle with the mouse-based interfaces, graphical languages can nonetheless serve a useful role in the classroom. That said, for certain activities and for certain children, the tangible version of Tern was clearly advantageous. On the other hand, some of the most independent student work that we observed was done with the graphical interface. Thus, one immediate outcome of this pilot study is that we have begun work on a hybrid tangible / graphical programming system, designed to allow teachers greater flexibility to incorporate programming activities in the classroom. The goal is to provide a single application that supports either tangible or graphical programming and allows for a fluid transition from one form to another.

In terms of the second hypothesis, what will be more interesting is to document a range of student abilities and determine whether our intervention results in an increase in understanding. Thus, in an effort to develop a more nuanced understanding of children's capabilities, we have begun to develop a model of the progression of children's understanding of computer programming. In our observations, children seemed to exhibit two levels of understanding: syntactic and semantic. And, although we saw only preliminary evidence of this, we hypothesize that young children could advance beyond semantic understanding to a system-level understanding.

Syntactic Level: In the first whole-class session, we introduced the activity by teaching a robot how to dance the hokey-pokey using the tangible programming blocks. The children sang along as the robot dance. We then asked the children to program a robot to tell a story through a sequence of actions. What we observed is that children tended to construct long chains of action statements without pausing to think. When we asked children to explain their stories, they would respond by simply reading the blocks in order: “My story is: FORWARD, SHAKE, BEEP...” This is an example of purely syntactic understanding. The children could read the blocks, and they knew that a sequence was implied.

Semantic Level: Later, we began to introduce higher-level concepts such as loops and sensors. As children used these blocks to solve specific problems such as the *Hungry, hungry robot* challenge, we posit that the children were acting with a semantic-level understanding. That is, they began to understand the meaning of individual blocks in the context of a program, although not necessarily in the context of a robot acting out the program in the world.

System Level: Near the end of the intervention at the urban school in 2007, one seven-year-old boy was able to correctly use his finger to trace a program containing a loop, a conditional branch, and two subroutines. Unfortunately, I did not record this program, but it was similar to the one shown below. Such a sophisticated level of understanding was surprising to me, and the child responded correctly to several follow-up questions that I asked him about the program.



This child had a very advanced semantic-level understanding of the programming language, and I believe that he was on the cusp of a system-level understanding. That is, he had developed a level of fluency with the language that would allow him to focus on understanding the purpose of a program in the context of the system as a whole. With a system-level understanding, a child uses programming as a tool to accomplish strategic goals with the robot. For example, a semantic understanding of loops could be expressed with language such as:

This will do the forward block three times.

A system-level understanding, on the other hand, would be expressed with language like this:

I need to make the robot move all the way through the door, so I'm going to make it go forward three times. That should be enough.

Our goal is to introduce programming concepts in a structured way to help children advance to a system-level understanding. We hypothesize that only at this level will children begin to feel a sense of intellectual ownership and empowerment, supported by the confidence that they can use programming as a tool to accomplish goals with a robot. Furthermore, Mayer (Mayer, 1981) defines understanding of computer programming concepts as the ability to transfer concepts to novel situations (i.e. new forms of computer programming or different types of tasks). We propose that this understanding, as Mayer defines it, only occurs after the child has reached the system-level.

Chapter 7

Conclusion

In this dissertation I have described the design and implementation of a tangible computer programming language for education called Tern. I also described research studies investigating the use of Tern in both formal and informal educational settings: as part of a computer programming and robotics exhibit at the Boston Museum of Science, and as part of a curriculum unit piloted in several local kindergarten classrooms with 93 children ages 5 to 7. The research study at the Boston Museum of Science provides evidence that tangible and graphical programming languages are equally easy for museum visitors to understand. However, the tangible interface is more inviting and better at encouraging active participation among members of family groups. These results were particularly strong for children, who seemed more actively engaged with the tangible exhibit than with the graphical exhibit. The design, implementation, and evaluation of this exhibit is the primary contribution of this dissertation.

I also described a Tangible Kindergarten pilot study, in which I provided preliminary evidence that if children are given access to age-appropriate technology integrated within a broader curriculum, they are capable both of programming on their own without direct adult help and of understanding certain powerful ideas from the domains of computer programming and robotics. However, much work must still be done to document the range of children's understanding and their ability to learn new concepts and build on existing knowledge. Furthermore, while the results from this pilot study are preliminary, they shed light on process of transitioning this technology from informal to formal learning settings. In so doing, they reveal limitations of the current design and highlight directions for future work.

7.1 Implications

Taken together, my work with tangible programming in both formal and informal learning settings has a number of implications for future work in this area. In this section, I review some of these implications as well as lessons learned through the process of iterative design and testing of Tern over a period of three years.

7.1.1 Interaction style and gender

Results from our research in the Museum suggest ways in which emerging interaction techniques such as tangible user interfaces might help create more inviting, engaging, and intrinsically motivating computer-based learning activities. In particular, our results revealed that the choice of interaction technique can have a dramatic effect on an exhibit's appeal to girls as well as boys. When combined with other recent research on involving girls in computer programming activities (Kelleher & Pausch, 2005; Buechley et al., 2008), our results have implications for future work seeking to ameliorate gender inequities in fields related to computer science. Furthermore, it might be worthwhile to explore the effect of interaction style or more intangible aspects of children's learning, such as emotional predispositions towards computer programming or positive experiences working in groups.

7.1.2 Interaction style matters, but other things matter more...

While our research in the Museum has shown that the choice of interaction technique matters, our results also suggest that this choice has little effect on the quantity or complexity visitor-created programs. Furthermore, other factors can be equally, if not more, important for engaging children. For example, my informal observations over two years in the museum suggest that things like the visual design of the exhibit installation (e.g. bare plywood versus paint or the inclusion of the interactive Robot Park sign) have a large impact on overall use and engagement. And, it is not surprising that in classroom settings factors such as time of day, teaching style, curriculum, and student group dynamics play a large role in the overall success of our interventions. Even the fact that the programming language (whether graphical or tangible) produces a tangible output—a robot that physically moves around the classroom—likely has a significant impact on student engagement.

7.1.3 There are many avenues to collaboration...

In the museum study, we found that tangible blocks were better at encouraging active participation among members of family groups. Furthermore, having multiple group members involved in the activity led to significantly longer engagement times. These results, while perhaps encouraging for tangible interface designers, are not surprising. Indeed, previous research has done an excellent job documenting the advantages of simply providing children with multiple input devices to improve educational outcomes (see Inkpen et al., 1995b,a; Scott et al., 2000, 2003). Thus, approaches such as Single Display Groupware (Stewart et al., 1998) and multi-touch displays may provide more practical means of supporting collaboration among children.

7.1.4 Focus on Simplicity

In designing tangible programming systems there is a temptation to start with a specification for a full-fledged computer language. In my experience, however, this *Turing complete* mentality can be detrimental, especially when designing for young children and first time programmers. Before spending time on features like conditionals, variables, subroutine, and parameters, the fundamentals of interaction must be sound, and the learning goals of the supported activity should be well understood. In particular, the physical (or industrial) design of the basic components is essential. As an example, in my earliest designs, I spent substantial effort creating a complete language specification before beginning the physical design work. This turned out to be a mistake—the result was a full-fledged programming language that was almost completely unusable. Even if the implementation of more advanced language features is less than perfect, users (even young children) will be resourceful and forgiving if the fundamental aspects of the user experience are sound. Furthermore, the desired learning objectives may well be satisfied with a less complete set of language constructs.

7.1.5 Functionality and Practicality

Wizard of Oz prototyping is difficult to conduct with programming language prototypes, and it is even more difficult outside of laboratory settings. Thus, to test designs, in my experience it is important to have functional prototypes. The computer vision techniques that I developed to support *compiled* tangible languages have proved to be useful for creating reasonably rapid and functional prototypes. This *passive tangible* approach has also resulted in durable, cost-effective, and reliable systems that are practical for use in classrooms and museums. These properties are also useful in conducting long-term research with technology-based interventions. One drawback of the computer vision approach is that it effectively limits the design space to two dimensions, in many ways simply replicating screen-based interaction techniques in the physical world.

7.1.6 Allow for multiple representations and interaction styles

A common strategy in kindergarten classrooms is to provide a variety of ways for students to participate in learning, often through the use of multiple representations for a concept to be learned. The Tangible Kindergarten pilot study revealed that many young children can successfully and independently engage in mouse-based programming activities in the classroom. And, had we used more child-friendly mouse operations (such as click-to-carry instead of click-and-drag), children would likely have been even more successful. The takeaway message for me is not that the tangible approach should be abandoned. On the contrary, there were many situations for which it was advantageous to use physical blocks. Rather, I now believe it is important to develop a programming system that supports the seamless integration of tangible and graphical (and perhaps even text-based) programming in the classroom. Providing multiple representations affords students and teachers the flexibility to choose the most appropriate tool for a given situation. It also opens exciting new possibility for programming language design—imaging, for example, being able to create a subroutine in a graphical system that is then *embodied* in a physical block.

7.1.7 Technology in the classroom? Keep it informal.

In science museums, the context is one of informal science education—there are no teachers or curriculum to guide children through complex concepts. If a museum visitors can't figure out an exhibit in the first minute or two, they will give up and move on to another activity. Furthermore, museums face constraints of durability, reliability, and cost of maintenance and development. As non-profit, public institutions, museums are often cash-strapped, so exhibits must work reliably and well. If maintenance is too much of a burden, then an exhibit simply won't be used. As Serrell puts it, the most common text label for computer-based exhibits is, "Sorry, out of order" (Serrell, 1996).

Not surprisingly, many of the constraints faced by museums are equally relevant in classrooms—technology that doesn't work will sit in the corners of the room gathering dust. However, in classrooms, the context is one of formal education. Teachers face large numbers of children, a regimented school day, and stiff educational standards mandated by local, state, and federal governments. Thus, I now believe that apprehendability is an essential design goal for classrooms as well as museums, and the reason has as much to do with teachers as it does with students. Here, the issue comes back to Cuban's notion of contextually constrained choice. Just like museum visitors, if teachers can't immediately understand how a new type of technology works and how it will fit into the routine of the school day, they will be unlikely to try it with students.

7.1.8 Allow for easy customization

In the Museum we attempted to design layered programming activities that could be experienced by visitors on different levels depending on their backgrounds. In addition, we provided opportunities for the museum staff to enhance the activity by *interpreting* the exhibit for visitors. For example, the staff could bring out extra programming blocks for visitors to experiment with more sophisticated programming constructs. These blocks are ordinarily stored out of sight because they can be confusing for visitors who encounter the exhibit for the first time. Here the use of tangible blocks is advantageous because it allows the museum staff to easily customize the interface on an *ad hoc* basis to match the interest of a particular visitor. This tangible advantage transfers to the classroom as well. Teachers may decide to include only simple blocks for introductory activities. Then, over the course of a curriculum unit, teachers can gradually include other blocks as the students advance. To do the same with a graphical programming system would require teachers to customize the software in some way. Either that, or it would mean providing several versions or levels of the same programming environment, as is the case with ROBOLAB.

7.2 Future Work

My work with tangible programming languages is ongoing. In the case of science museums, I am interested in comparing tangible interfaces to a broader spectrum of input devices for use in informal science learning such as multi-touch surfaces and multi-mice systems. Our results on engagement suggest that these alternative multi-user devices might produce similar benefits to tangible interaction. I am also interested in exploring the ways in which visitors engage the exhibit and what they learn. One possible way to do this is to record visitor conversations as they interact with the exhibit. These recordings could then be analyzed for evidence of *learning talk*—for example, see the work of Sue Allen at the San Francisco Exploratorium (Allen, 2002). Through such a study I would expect to discover that visitors interact with the exhibit in unexpected ways. As an example, one boy who visited the exhibit spent over a half an hour trying to draw the TopCode symbols by hand so that they would be recognized by the computer vision system (Figure 7.1). Discovering other unexpected forms of interaction might help broaden my approach and open new avenues for research.

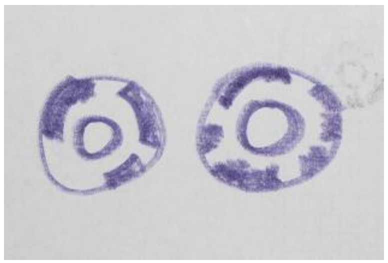


Figure 7.1: One boy who visited the museum spend over 30 minutes trying to draw the TopCode symbols by hand so that they would be recognized by the computer vision system. This photograph shows two of his attempts. He was eventually successful.

In the case of classrooms, Tangible Kindergarten is a three-year research project funded by the National Science Foundation. In the next two years we will continue to improve both our curriculum and technology. And, in the third year, we plan to conduct a formal study with the goal of documenting the capability of young children to engage with powerful ideas from computer programming and robotics and to develop their own technologically-rich projects. Part of this effort will be the development of a hybrid tangible / graphical programming system. The goal is to provide a single application that supports either tangible or graphical programming and allows for a fluid transition from one form to another.

7.3 Conclusion

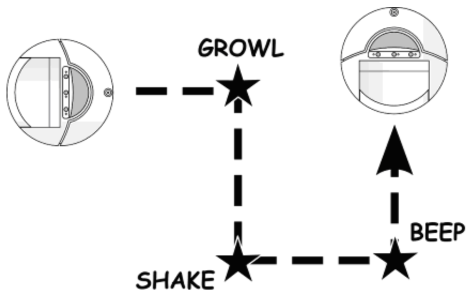
In the first chapter of this document, I stated that the goal of this project is not to suggest that tangible programming should replace existing graphical systems as general-purpose tools. Rather, I argued that tangible languages begin to make sense when one considers the specific contexts and constraints of learning environments where programming activities might be used. One of my primary goals in the development of Tern was to create a tangible interface for computer programming that was practical for use in real world educational environments. Over 18 months and 20,000 visitors later, Robot Park is still on the museum floor, seven days a week, requiring little maintenance. In this goal, I think that I have succeeded. Beyond practicality, I worked hard to make Tern an inspiring and fun learning tool that museum visitors would want to try and that teachers and students would look forward to using in the classroom. Again, in the science museum context, I think that I have succeeded. At least, evidence from our study suggests an improvement over the standard graphical interface. Children, in particular, were much more likely to try programming with the tangible blocks.

In classrooms, however, the answer is less clear. In many ways, the system is still too unreliable to hope that it will be adopted by many teachers. In addition, our pilot study suggests that while many children do struggle with the mouse-based interfaces, graphical languages can nonetheless serve a useful role in the classroom. Thus, one of the next goals for the Tangible Kindergarten project is to create a hybrid tangible / graphical programming system, designed to allow teachers greater flexibility to incorporate programming activities in the classroom. In summary, the ultimate goal of this project is to support the inclusion of computer programming activities in real life educational environments where they might otherwise be omitted. To do this I am exploring the use of novel interactive technology to overcome the limitations of existing graphical systems.

Appendix A

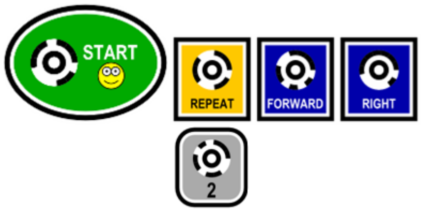
Robot Park Interview Questions

Date: ____________________
Time: ____________________
Condition: (GUI / TUI) circle one
1. How old are you?
2. What grade are you in?
3. Male / Female
4. About how long have you been at the museum today?
5. How would you describe this exhibit to a friend? What would you say you did?
6. Why did you decide to try this exhibit today?
7. Can you tell me more about how you use computers at home or at school? How often? Do you enjoy using computers?
8. Have you ever programmed a computer before? Is this programming?
9. OK. Here's a path that I want the robot to follow. Can you work together to build a program that will follow this path?



10. Great. Let's see what the robot does. Was that what you expected? Why or why not? How would you fix it?

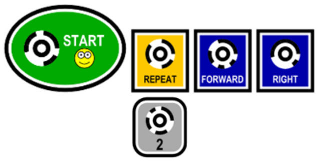
11. OK. Now I'm going to write a program. What do you think it will make the robot do?



12. Great. Let's try it. Did it do what you thought it would?

13. OK. Here's the path that I actually want the robot to follow. Can you fix the program so that it will follow this path?

14. As part of this study, we're comparing two user interfaces. We'd be interested in your opinions on them. Can I show you the other interface? Could you use this interface to have the robot follow the pattern (below)?



15. Great. I'm curious what you think of these two options?

17. Do you have any suggestions for how we could make this exhibit better? Would you be willing to answer a few last questions on this sheet of paper?

18. It was more fun to use (please circle one option):

The wooden blocks	The mouse	They were about the same	No opinion

19. It was easier for me to use:

The wooden blocks	The mouse	They were about the same	No opinion
20. If I were working alone, I would want to use:			
The wooden blocks	The mouse	They were about the same	No opinion
21. If I were working with my friends or family, I would want to use:			
The wooden blocks	The mouse	They were about the same	No opinion
22. If I had to do a programming project at school, I would want to use:			
The wooden blocks	The mouse	They were about the same	No opinion
23. If I had to do a programming project at home, I would want to use:			
The wooden blocks	The mouse	They were about the same	No opinion

Table A.1: Survey instrument used to collect qualitative data .
at the exhibit

Appendix B

Robot Park Observation Data

Date: Saturday, March 1
Researcher: Erin
Condition: TUI

Visitor	Looking	Interacting	Ah-ha	Leave	Notes
W40	11:43:42 AM	—	—	11:43:54 AM	
G8	11:52:30 AM	11:52:44 AM	—	11:53:21 AM	started by picking up robot
G8	11:52:42 AM	11:52:44 AM	—	11:53:18 AM	started by sitting down
B16	12:17:06 PM	12:17:11 PM	12:18:37 PM	12:21:09 PM	“See these are the things I like doing. Nice and simple”
M45	12:17:09 PM	12:20:26 PM	12:19:19 PM	12:21:09 PM	“It just picks up by the camera”
M35	12:23:31 PM	12:23:54 PM	12:24:49 PM	12:33:53 PM	This is cool! – That’s what you got to do, wait for bump, so you can have it wait until it bumps into wall

B8	12:23:50 PM	12:23:56 PM	12:24:51 PM	12:35:55 PM	boy and mom came back and took a pic in front of it. Others are using. He says: they won! (they got it to work)
F35	12:34:15 PM	12:34:18 PM	12:34:42 PM	12:35:55 PM	mom arrives, and boy shows mom
F26	12:29:16 PM	—	—	12:29:23 PM	
G10	12:35:55 PM	12:36:31 PM	12:38:32 PM	12:42:46 PM	both girls using completely together. Wait don't press play while I'm still working. "Let's do this! Look at booklet"
G7	12:35:58 PM	12:36:34 PM	12:38:36 PM	12:43:02 PM	
M35	12:37:16 PM	—	12:38:38 PM	12:42:27 PM	father not really paying attention, except when robot moves, make noise
W40	12:45:07 PM	12:45:10 PM	—	12:45:16 PM	
W35	12:45:37 PM	—	—	12:46:02 PM	
B9	12:46:25 PM	—	—	12:46:28 PM	
B15	12:48:26 PM	—	—	12:48:29 PM	
F17	12:51:35 PM	12:51:41 PM	12:52:34 PM	12:53:12 PM	pressed play first, and robot started going from prev program
F17	12:52:45 PM	—	—	—	Robot coming near her: Get away from me!!!
B14	12:53:07 PM	12:53:29 PM	—	12:55:21 PM	
B14	12:53:25 PM	12:53:51 PM	—	12:55:22 PM	left a really long program without running it
M27	12:55:26 PM	12:55:39 PM	1:00:41 PM	1:02:13 PM	pressed play and ran program that was left

F27	12:55:28 PM	—	1:00:42 PM	1:02:15 PM	looked quickly at graphic at top that you can try. Didn't actually interact, just watched
G6	12:58:45 PM	—	—	12:58:51 PM	dad is using computer beside exhibit
B4	12:59:00 PM	—	—	12:59:28 PM	watching others use
G6	1:02:19 PM	1:02:27 PM	1:03:16 PM	1:04:08 PM	same girl as above. When people left, she came over
B4	1:06:35 PM	—	—	1:06:41 PM	
M60	1:09:03 PM	—	—	1:09:13 PM	examinging robot
B8	1:09:32 PM	1:09:48 PM	1:10:04 PM	1:10:53 PM	looking at camera. OMG
B8	1:10:00 PM	1:09:48 PM	1:10:04 PM	1:15:11 PM	you program it!
F25	1:10:42 PM	—	—	1:11:06 PM	
F25	1:10:44 PM	—	—	1:11:08 PM	
F18	1:19:13 PM	1:19:27 PM	1:19:51 PM	1:23:07 PM	
F18	1:19:16 PM	—	1:22:20 PM	1:23:09 PM	sorta looks at book (turns page, but doesn't seem to read)
G3	1:23:24 PM	1:23:26 PM	—	1:23:54 PM	
F30	1:23:39 PM	1:23:49 PM	1:24:55 PM	1:32:52 PM	
B9	1:25:11 PM	1:24:55 PM	1:29:49 PM	1:32:07 PM	it does exactly what it says. Explains to girl
F35	1:25:32 PM	1:27:42 PM	1:29:46 PM	1:32:08 PM	
G9	1:28:21 PM	1:28:31 PM	1:28:31 PM	1:32:56 PM	let's make up a new dance
F30	1:36:53 PM	1:36:55 PM	—	1:36:58 PM	
M30	1:36:53 PM	—	—	1:36:58 PM	
F17	1:39:10 PM	1:39:20 PM	1:39:35 PM	1:47:06 PM	omg that's creepy. It takes a picture of your hand
F17	1:39:18 PM	1:39:45 PM	1:39:37 PM	1:47:08 PM	

B8	1:43:04 PM	—	1:43:10 PM	1:43:24 PM	Oh! Laughing. Watching others use it
M50	1:45:10 PM	—	—	1:45:23 PM	observing others
M50	1:47:18 PM	1:47:18 PM	1:47:43 PM	2:18:40 PM	when you put these pieces together, it tells it what to do
B4	1:47:20 PM	1:47:20 PM	1:49:28 PM	2:18:39 PM	
M18	1:47:52 PM	—	—	1:49:58 PM	observing others
F18	1:49:23 PM	—	—	1:49:56 PM	
M60	1:51:52 PM	—	—	1:54:27 PM	
M60	1:51:52 PM	—	—	1:54:27 PM	
B5	1:59:19 PM	—	—	2:00:03 PM	observing others. Playing with robot. Putting self under camera
B8	1:59:50 PM	—	—	2:00:03 PM	climbing and going under camera
B16	2:22:06 PM	—	—	2:22:16 PM	
G9	2:22:26 PM	2:22:54 PM	2:22:48 PM	2:28:46 PM	
F20	2:22:28 PM	2:22:55 PM	2:22:51 PM	2:28:46 PM	
M20	2:23:48 PM	—	—	2:28:46 PM	
B8	2:29:08 PM	2:29:48 PM	—	2:30:36 PM	
B13	2:29:44 PM	2:29:48 PM	—	2:30:13 PM	
F35	2:31:20 PM	—	—	2:31:36 PM	
M35	2:31:24 PM	2:32:00 AM	—	2:33:56 PM	
M50	4:04:35 PM	12:08:03 PM	12:08:03 PM	12:14:00 PM	
G11	4:04:35 PM	12:08:05 PM	12:08:05 PM	12:14:00 PM	
B5	4:04:35 PM	12:08:07 PM	12:08:07 PM	4:04:35 PM	
W40	4:04:35 PM	12:08:09 PM	12:08:09 PM	4:04:35 PM	

Date: Sunday, March 2
Researcher: Mike
Condition: GUI

Visitor	**Looking**	**Interacting**	**Ah-ha**	**Leave**	**Notes**
B5	4:04:35 PM	4:04:35 PM	—	4:04:35 PM	
M38	4:04:35 PM	4:04:35 PM	4:04:35 PM	4:04:35 PM	Dad doing mouse. Boy asking questions. Can't hear questions
M50	4:04:35 PM	12:08:03 PM	12:08:03 PM	12:14:00 PM	Pulled out tangible blocks had used this before :)
G11	4:04:35 PM	12:08:05 PM	12:08:05 PM	12:14:00 PM	
B5	4:04:35 PM	12:08:07 PM	12:08:07 PM	4:04:35 PM	Really likes exploring. Is trying all sorts of different programs. Second time using this today, and he's still here. Boy is doing all the work. Mom is being patient.
W40	4:04:35 PM	12:08:09 PM	12:08:09 PM	4:04:35 PM	
G11	4:04:35 PM	—	—	—	just glanced
W40	4:04:35 PM	—	—	—	just glanced
M22	4:04:35 PM	4:04:35 PM	4:04:35 PM	4:04:35 PM	touching robot, turning it on off. Not connecting blocks together on screen. Just arranging them. Now he gets it.
W22	4:04:35 PM	4:04:35 PM	4:04:35 PM	4:04:35 PM	She's using the IR beam while he programs
G12	4:04:35 PM	—	—	—	glancing off to AIBO
W20	4:04:35 PM	—	—	—	
M45	4:04:35 PM	—	—	—	
W45	4:04:35 PM	—	—	—	
G16	4:04:35 PM	4:04:35 PM	4:04:35 PM	4:04:35 PM	playing with IR beam. Trying the maze.

M33	4:04:35 PM	—	—	—	
M40	4:04:35 PM	—	—	—	
M21	4:04:35 PM	—	—	—	
M35	4:04:35 PM	—	—	—	
G13	4:04:35 PM	4:04:35 PM	—	4:04:35 PM	Looked but was being used
W30	4:04:35 PM	—	—	—	
B8	4:04:35 PM	—	—	—	
M30	4:04:35 PM	—	—	—	Looking but waiting for turn
B4	4:04:35 PM	4:04:35 PM	—	4:04:35 PM	
B15	4:04:35 PM	—	—	—	just glanced
B12	4:04:35 PM	—	—	—	
M38	4:04:35 PM	4:04:35 PM	—	4:04:35 PM	Damn. Using IR beam. Confused by beam. Finally picked up mouse. Also confused by interface. How blocks fit together and come apart. Robot is also stuck agains the wall, so that's confusing for him too.
M33	4:04:35 PM	—	—	—	
M38	4:04:35 PM	4:04:35 PM	4:04:35 PM	4:04:35 PM	Pretty quickly figured it out. Same person as row 28??
W45	4:04:35 PM	—	—	—	just glanced
B13	4:04:35 PM	4:04:35 PM	4:04:35 PM	4:04:35 PM	
B15	4:04:35 PM	—	—	—	
G6	4:04:35 PM	—	—	—	
M30	4:04:35 PM	4:04:35 PM	4:04:35 PM	4:04:35 PM	
W30	4:04:35 PM	—	—	4:04:35 PM	watched a while and then left.
G10	4:04:35 PM	1:28:23 AM	—	4:04:35 PM	Just watching dad.

M30	4:04:35 PM	—	—	4:04:35 PM	Spoke to other group about Roomba for a while.
W30	4:04:35 PM	—	—	—	
G6	4:04:35 PM	—	—	—	
M38	4:04:35 PM	—	—	—	just glanced
B11	4:04:35 PM	4:04:35 PM	—	4:04:35 PM	Much more interested in Aibo. Gets connection of blocks sort of, but don't know what to do...
W30	4:04:35 PM	—	—	4:04:35 PM	
M45	4:04:35 PM	—	—	—	Son more interested in playing mancala.
B10	4:04:35 PM	—	—	—	
M42	4:04:35 PM	4:04:35 PM	—	4:04:35 PM	
G10	4:04:35 PM	—	—	4:04:35 PM	
M50	4:04:35 PM	4:04:35 PM	4:04:35 PM	4:04:35 PM	Boy just watching father.
B5	4:04:35 PM	4:04:35 PM	4:04:35 PM	4:04:35 PM	Lots of talk / questions with brother and father.
B8	4:04:35 PM	—	—	4:04:35 PM	Keeps coming back to look. Lightning show also going on.
M35	4:04:35 PM	—	—	—	
G9	4:04:35 PM	—	—	—	Other group is still using
M50	4:04:35 PM	—	—	—	Other group still using
G12	4:04:35 PM	4:04:35 PM	—	—	Touches IR box
W20	4:04:35 PM	—	—	—	Boy is still using.
M21	4:04:35 PM	—	—	—	
W40	4:04:35 PM	—	—	—	just glancing. Daughter at AIBO
M23	4:04:35 PM	4:04:35 PM	4:04:35 PM	4:04:35 PM	Confused because off edge.
G8	4:04:35 PM	—	—	—	

M30	4:04:35 PM	—	—	—	Someone else using.
W32	4:04:35 PM	—	—	—	Someone else using.

Date: Saturday, March 15
Researcher: Mike
Condition: TUI
Notes: Plexiglass barrier added and monitor back 3 inches

Visitor	**Looking**	**Interacting**	**Ah-ha**	**Leave**	**Notes**
W37	12:14:21 PM	12:14:25 PM	12:16:03 PM	12:18:32 PM	Funny, the little boy likes playing with the robot. She's confused because she blocked the camera.
B1	12:14:42 PM	12:14:43 PM	—	12:18:34 PM	Boy comes back... very funny... likes pressing the stop button to make noise.
B14	12:21:02 PM	—	—	—	
B16	12:26:45 PM	12:26:54 PM	12:27:35 PM	12:27:39 PM	
B16	12:26:47 PM	12:26:50 PM	—	12:27:43 PM	
B8	12:28:05 PM	12:28:15 PM	—	12:28:42 PM	
W20	12:28:25 PM	12:28:31 PM	—	12:28:45 PM	
B10	12:29:45 PM	12:29:39 PM	—	12:34:42 PM	Starts with block tester, but REVERSE isn't tagged.
M30	12:29:42 PM	12:30:38 PM	12:31:06 PM	12:34:25 PM	Man moves robot out to start area. Confused by front of robot.
W30	12:32:21 PM	—	—	12:33:15 PM	
W60	12:34:09 PM	—	—	—	
W65	12:34:11 PM	—	—	—	
B12	12:34:47 PM	12:34:50 PM	12:36:59 PM	12:42:38 PM	Boy didn't want to see AIBO. Comes back. Really trying a lot of programs.
W30	12:35:21 PM	12:50:52 PM	12:51:26 PM	12:53:02 PM	
M30	12:35:23 PM	12:50:54 PM	12:51:28 PM	12:53:00 PM	
B8	12:35:10 PM	12:35:13 PM	12:36:52 PM	—	Cool. Awww! Really interested in seeing robot go.
B9	12:35:32 PM	12:35:34 PM	12:36:54 PM	12:40:11 PM	Block tester interferes with programs.

M40	12:42:54 PM	12:43:02 PM	12:44:50 PM	12:49:31 PM	Reading documentation. Flipping through challenges.
B15	12:46:07 PM	12:47:30 PM	12:45:00 PM	12:50:12 PM	
B11	12:43:16 PM	—	—	—	
W25	12:46:30 PM	—	—	—	
M25	12:46:36 PM	—	—	—	Other people using.
M20	12:47:36 PM	—	—	—	Other people using
M22	12:47:48 PM	—	—	—	Other people using
					** Really busy all of a sudden **
B13	12:49:01 PM	—	—	—	Other people using
M38	12:51:42 PM	—	—	—	Other people using
B5	12:52:40 PM	12:52:47 PM	—	1:00:58 PM	
W30	12:52:42 PM	12:54:23 PM	12:54:20 PM	12:54:27 PM	Session overlapping with other boys.
G9	12:53:15 PM	12:53:18 PM	—	12:53:46 PM	
M22	12:53:35 PM	—	—	12:54:15 PM	
M18	3/15/2008 13:09	—	—	—	
M53	12:56:09 PM	—	—	—	
W25	12:56:17 PM	—	—	—	
M35	12:54:51 PM	—	—	12:58:14 PM	
B12	12:54:52 PM	12:55:13 PM	—	12:58:12 PM	Too many boys all at once
B8	12:55:06 PM	12:55:23 PM	12:56:32 PM	12:56:28 PM	
W30	12:55:07 PM	—	—	12:58:18 PM	
M35	12:59:41 PM	—	—	—	
M28	1:00:26 PM	1:03:43 PM	1:03:45 PM	1:05:50 PM	Two groups overlapping now. Happens a lot todayy.
W28	1:03:03 PM	—	—	1:05:51 PM	
B12	12:59:04 PM	1:00:06 PM	1:01:16 PM	1:03:14 PM	

B7	1:00:18 PM	1:00:19 PM	1:01:18 PM	1:03:16 PM	
M25	1:01:00 AM	—	—	—	
B12	1:02:41 PM	1:03:08 PM	1:03:35 PM	1:04:47 PM	This group is with M28, W28 above.
W52	1:02:45 PM	1:03:09 PM	1:03:36 PM	1:05:57 PM	Boy leaves first. Funny. Mom doesn't want to go.
M28	1:06:29 PM	1:10:11 PM	1:10:12 PM	1:10:55 PM	
W28	1:06:31 PM	1:06:55 PM	1:10:19 PM	1:10:56 PM	
W25	1:06:34 PM	1:06:50 PM	1:08:35 PM	1:10:58 PM	
M22	1:11:07 PM	1:11:09 PM	1:12:33 PM	1:12:59 PM	Only gets block tester. Doesn't see button.
M38	1:13:46 PM	1:13:57 PM	1:14:28 PM	1:25:00 PM	Confused because block is too complicated. Keeps pressing and saying START¿ It's not working... hmm... Let's move all these others off. Thinks he has to clear screen. "OUCH" when robot runs into the wall. Maybe this [Lblock] is a way to make the prorgram longer.
B12	1:14:20 PM	1:14:22 PM	1:14:24 PM	—	is there an END? Let's get rid of this while. See. It's going to REPEAT it 4 times! "Oh it can zoom in if it wants." Confused by IR beam set up.
M28	1:17:26 PM	—	—	—	
B12	1:17:26 PM	—	—	—	Others using
W26	1:21:50 PM	—	—	—	Others using
M25	1:22:24 PM	—	—	—	Others using
W25	1:22:24 PM	—	—	—	

M50	1:24:14 PM	—	—	—	Others using
M45	1:24:16 PM	—	—	—	
M24	1:26:36 PM	—	—	1:29:05 PM	Others using
M50	1:26:38 PM	—	—	1:29:07 PM	
B10	1:28:23 PM	1:28:33 PM	—	1:29:02 PM	
B13	1:28:29 PM	—	—	1:29:09 PM	
W22	1:30:29 PM	—	—	1:30:50 PM	
W30	1:30:25 PM	—	—	1:30:50 PM	
M35	1:30:22 PM	—	—	1:30:00 PM	
G15	1:30:19 PM	1:30:32 PM	—	1:31:42 PM	
W17	1:32:18 PM	1:32:22 PM	1:33:33 PM	1:33:30 PM	
M70	1:33:03 PM	—	—		
W70	1:33:12 PM	—	—		
W75	1:33:16 PM	—	—		
B8	1:33:39 PM	1:33:40 PM	1:34:48 PM	1:36:50 PM	
M33	1:33:46 PM	1:33:49 PM	1:34:50 PM	1:36:44 PM	
B9	1:34:41 PM	—	—	—	Others using
B12	1:34:38 PM	—	—	—	
G16	1:35:55 PM	—	—	—	Others using
W17	1:36:40 PM	—	—	—	
W18	1:37:29 PM	1:37:31 PM	—	1:44:33 PM	
W17	1:37:38 PM	—	—	1:44:35 PM	Just using block tester so far.
W17	1:37:45 PM	1:37:57 PM	1:38:31 PM	1:45:03 PM	
W17	1:37:50 PM	1:38:21 PM	1:38:32 PM	1:45:04 PM	
W17	1:40:55 PM	1:43:42 PM	—	1:45:06 PM	
W40	39522.57	—	—	1:44:05 PM	
W16	1:41:17 PM	1:43:45 PM	—	1:44:05 PM	Others using
M40	1:41:40 PM	—	—	1:42:00 PM	
M35	1:41:58 PM	—	—	1:42:00 PM	
M22	1:42:30 PM	—	—		
W22	1:42:31 PM	—	—		
W22	1:44:17 PM	—	—	1:44:58 PM	

Date: Sunday, March 16
Researcher: Erin
Condition: GUI

Visitor	Looking	Interacting	Ah-ha	Leave	Notes
M30	12:25:27 PM	—	—	12:25:36 PM	
B6	12:27:02 PM	—	—	12:27:05 PM	
G10	12:36:28 PM	—	—	12:36:41 PM	
M28	12:36:52 PM	12:36:58 PM	12:37:59 PM	12:38:39 PM	
M28	12:36:55 PM	—	12:38:01 PM	12:38:40 PM	passive observer was helping. “No, it’s not connected”
B10	12:39:06 PM	12:39:06 PM	12:41:56 PM	12:46:52 PM	looks at manual. Tries program from before. Left with fam for a minute and came back. Mom also came back
G12	12:41:42 PM	—	—	12:43:09 PM	
F40	12:41:45 PM	12:42:10 PM	12:41:56 PM	12:43:13 PM	
M40	12:41:51 PM	—	—	12:46:49 PM	
G7	12:42:31 PM	—	—	12:43:17 PM	
G13	12:44:24 PM	—	—	12:44:40 PM	
G13	12:44:26 PM	—	—	12:44:42 PM	
B8	12:45:06 PM	—	—	12:45:29 PM	
M28	12:45:43 PM	—	—	12:46:04 PM	
B5	12:46:26 PM	—	—	12:46:41 PM	
M27	12:46:27 PM	—	—	12:46:43 PM	
M27	12:46:29 PM	—	—	12:46:43 PM	
B8	12:47:34 PM	12:47:46 PM	12:48:30 PM	12:54:59 PM	he had looked before when someone else was using. Now he came back

M35	12:47:39 PM	12:48:55 PM	12:48:25 PM	12:55:01 PM	"It's doing whatever the program says" . Wait. I'm gonna put it right here on start. Tries to get son to do maze. He's definitely instructing: put forward. Okay, another forward. How about left. Son is using mouse and doing whatever dad says. Dad corrects him. tells son to move whole thing here. see if you can move start all the way over there. Most of the time son was interacting and dad instructing. but dad did move robot and eventually did use the mouse a bit
F35	12:48:09 PM	—	—	12:48:18 PM	
B2	12:48:11 PM	—	—	12:48:20 PM	
M35	12:50:54 PM	—	—	12:51:59 PM	
B9	12:50:55 PM	—	—	12:52:06 PM	
B8	12:51:03 PM	—	—	12:52:07 PM	
B8	12:54:19 PM	—	—	12:54:26 PM	
B8	12:54:51 PM	12:55:05 PM	12:55:30 PM	12:58:54 PM	boy interacting, dad instructing, mom watching
M35	12:55:18 PM	—	12:55:32 PM	12:58:56 PM	trying to get robot back to start without picking up
F35	12:55:43 PM	—	—	12:58:57 PM	
B8	12:56:29 PM	—	—	12:57:29 PM	same as earlier
B9	12:56:34 PM	—	—	12:57:29 PM	
M35	12:56:36 PM	—	—	12:57:29 PM	
M35	12:59:43 PM	12:59:45 PM	1:00:43 PM	1:02:38 PM	
F35	1:01:52 PM	—	—	1:02:28 PM	
B8	1:01:53 PM	—	—	1:02:30 PM	

M30	1:09:32 PM	1:09:42 PM	1:10:47 PM	1:11:04 PM	
F30	1:09:33 PM	—	1:10:50 PM	1:11:05 PM	
F27	1:13:37 PM	1:13:48 PM	—	1:14:53 PM	speaking spanish
M30	1:13:39 PM	—	—	1:14:46 PM	
G9	1:14:14 PM	1:14:14 PM	—	1:15:10 PM	
F30	1:14:16 PM	—	—	1:14:39 PM	
M30	1:14:25 PM	—	—	1:14:40 PM	
B8	1:14:36 PM	—	—	1:14:42 PM	
M60	1:15:54 PM	1:16:04 PM	1:18:08 PM	1:23:06 PM	
F19	1:28:05 PM	1:28:05 PM	1:28:46 PM	1:29:21 PM	
M19	1:28:06 PM	—	1:28:47 PM	1:29:23 PM	giving passive instruction.
B16	1:30:40 PM	1:30:55 PM	1:30:55 PM	1:34:50 PM	
B16	1:49:58 PM	—	—	1:50:04 PM	
F19	2:15:15 PM	—	—	2:15:18 PM	
G16	2:26:28 PM	2:26:30 PM	2:27:40 PM	2:32:01 PM	
M40	2:26:36 PM	—	2:30:33 PM	2:37:13 PM	dad left when girl was using, but when boy was using came back and used it with him
B8	2:29:19 PM	2:31:45 PM	2:30:35 PM	2:37:15 PM	can I try. Keeps asking this. Let me do it. Finally she lets him. Got to use it for a while: “Isn't that cool”. I'm going to do one of everything. Can you move it to the start. I think he looked at book and was trying something
B7	2:37:20 PM	2:37:22 PM	—	2:37:44 PM	

Appendix C

PCODE Assembly Language Specification

Command	**Opcode**	**Args**	**Description**
noop	0x00	0	This instruction is ignored.
frame	0x01	0	Sets up a new stack frame for a function call. This includes the following actions: (1) pushes the current frame pointer; (2) sets the frame pointer to the top of the stack; and (3) pushes a zero, which acts as a placeholder for the return address.
yield	0x02	0	Suspends the current process or behavior.
stop	0x03	0	Terminates the current process or behavior. After stopping, behaviors may be restarted at a later time by the interpreter. Processes, on the other hand will not be restarted.
exit	0x04	0	Terminates the interpreter, stopping all processes and behaviors.
checkpoint	0x05	1	Checkpoints are used to implement the start-when clauses of behaviors: • Pops the stack • Sets the checkpoint property to the popped stack value • Yields the process • If the value of the checkpoint property is true, then the behavior has requested to run.

goto	0x06	1	Pops the stack and sets the instruction pointer to the popped stack value.
call	0x07	1	Invokes a subroutine. • Pushes the instruction-pointer to (fp + 1) • Pops the stack for subroutine address • Sets the instruction-pointer to the popped stack value
return	0x08	1	Returns from a function call: • Pops and saves the return value • Pops all arguments and local variables off the stack • Pops and restores the instruction pointer • Pops and restores the frame pointer • Pushes the return value back onto the stack
pop	0x09	0	Pops the stack and discards the value
store-global	0x0A	2	Pops the stack twice and stores the value at the given address location. The first argument is the assignment value and the second argument is the global variable's address.
load-literal	0x0B	0	Pushes a two-byte immediate value onto the stack. This value is included in the assembly code immediately following the instruction.
load-global	0x0C	1	Pops the stack and loads the global variable value stored at that address onto the stack.
and	0x0D	2	Pops the stack twice and then pushes the logical AND of the popped values.
or	0x0E	2	Pops the stack twice and then pushes the logical OR of the popped values.
not	0x0F	1	Pops the stack once and then pushes the logical NOT of the popped value.
if-true	0x10	2	Pops the stack twice. If the second popped value is not zero, jump to the destination address given by the first popped value.
if-false	0x11	2	Pops the stack twice. If the second popped value is zero, jump to the destination address given by the first popped value.
if-timer	0x12	1	Pops the stack once. Then, if the timer property is zero, jump to the destination address given by the popped stack value.

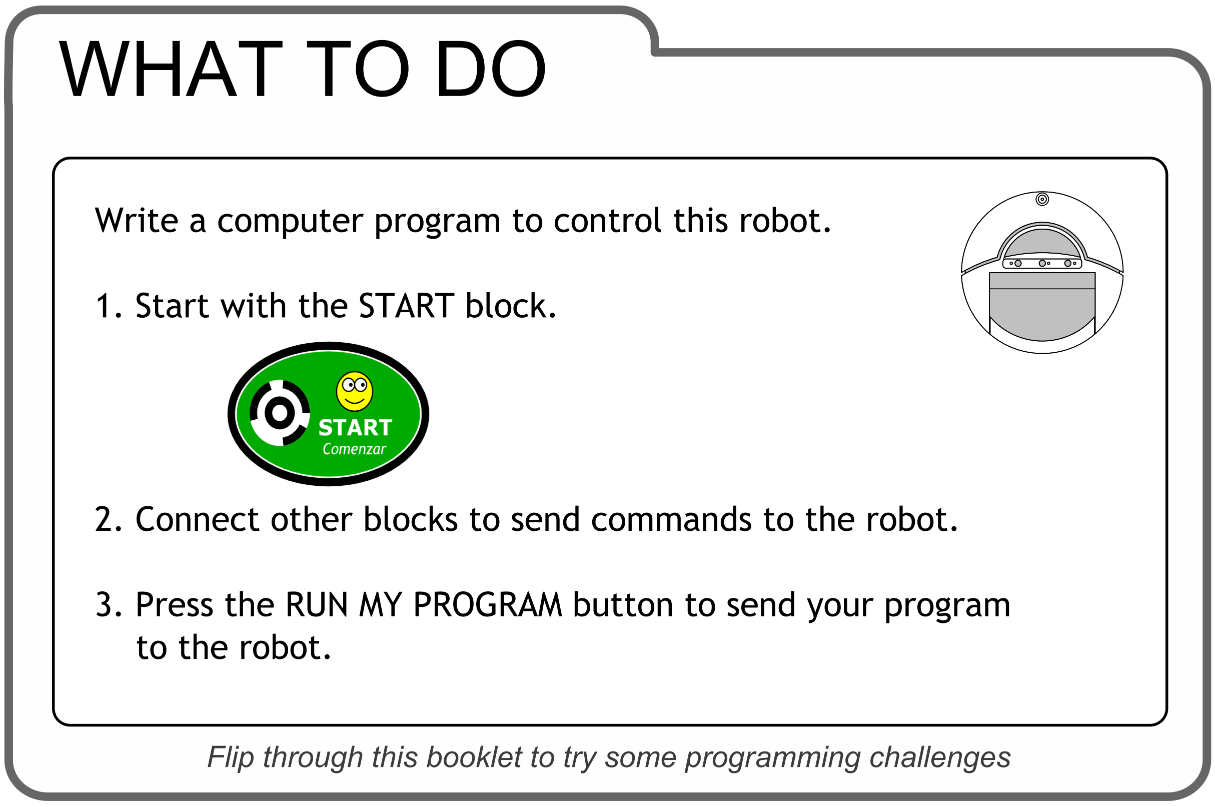
timer	0x13	1	Pops the stack once. Sets the timer property to the number of milliseconds given by the popped stack value. The interpreter will eventually decrement this value to zero as time passes.
random	0x14	2	Pops the stack twice for the low and high range arguments. Generates a random number from [low - high] and pushes it onto the stack.
store-frame	0x16	2	Replaces the given stack entry with a new value. The stack entry is specified by its position relative to the frame pointer—that is, position 0 is the bottom element of the current frame; position 1 is the second element of the frame; and so on. The first argument popped of the stack is the new value and the second argument popped is the frame offset.
load-frame	0x17	1	Copies the given stack entry to the top of the stack. The stack entry is specified by its position relative to the frame pointer.
=	0x18	2	Pops the stack twice. Pushes 1 if the two stack values are equal; pushes 0 otherwise.
>	0x19	2	Pops the stack twice. Pushes 1 if the first stack value is greater than the second; pushes 0 otherwise.
<	0x1A	2	Pops the stack twice. Pushes 1 if the first stack value is less than the second; pushes 0 otherwise
≥	0x1B	2	Pops the stack twice. Pushes 1 if the first stack value is greater than or equal to the second; pushes 0 otherwise.
≤	0x1C	2	Pops the stack twice. Pushes 1 if the first stack value is less than or equal to the second; pushes 0 otherwise.
<>	0x1D	2	Pops the stack twice. Pushes 1 if the two stack values are not equal; pushes 0 otherwise.
sensor	0x1E	1	Pops the stack to get a sensor ID number and then pushes current value of that sensor onto the stack.
led-on-off	0x1F	2	Pops the stack twice to get the LED ID number and the on/off value. Then turns the binary LED on or off.
led-color	0x20	3	Pops the stack three times to get the LED ID number, brightness, and color (brightness and color values may be between 0 and 255). Then sets the intensity and color of the given RGB LED.
song	0x21	1	Pops the stack once. Plays a predefined song given by the popped ID number.
beep	0x22	2	Pops the stack twice to get the pitch and duration. Plays a note with the given pitch and duration. Duration is specified in units of 1/64 seconds. The pitch is defined by MIDI note numbering scheme.

throttle	0x23	1	Pops the stack once. Sets the forward velocity (in mm/s) of the robot to the popped value. Negative values drive the robot backwards.
radius	0x24	1	Pops the stack once. Sets the robot's turn radius (in mm) to the popped stack value. 1 and -1 are turn in place; 0x8000 is drive straight.
rudder	0x25	1	Pops the stack once. Sets the robot's turn *rudder* to the popped stack value. The arguments specifies a rudder angle from -180 to 180 degrees. The equivalent radius is given by the formula $R = k/\sin(a/2)$.
+	0x26	2	Pops the stack twice and pushes the sum of the two stack values.
-	0x27	2	Pops the stack twice and pushes the difference of the two stack values.
*	0x28	2	Pops the stack twice and pushes the product of the two stack values.
/	0x29	2	Pops the stack twice and pushes the integer division result of the two stack values.
function	0x2A	0	Marks the start address of a helper function.
behavior	0x2B	0	Marks the start address of a behavior with the given priority level. The 8-bit priority level is specified in the assembly code immediately following the instruction.
process	0x2C	0	Marks the start address of a process.
data	0x2D	0	Marks the location of a global variable address and initializes the value. The initial 16-bit value is specified in the assembly code immediately following the instruction.
print-decimal	0x2E	0	Prints the top element of the stack through debug serial output. Leaves the stack unchanged.
actuator	0x31	2	Pops the stack twice. Applies the value of the first argument to the actuator specified by the ID number given by the second argument.
dup	0x32	0	Pushes a duplicate copy the top element of the stack.

Table C.1: Description of PCODE assembly instructions.

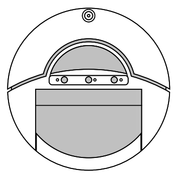
Appendix D

RobotPark Sinage and Documentation



WHAT TO DO

Write a computer program to control this robot.



1. Start with the START block.

START
Comenzar

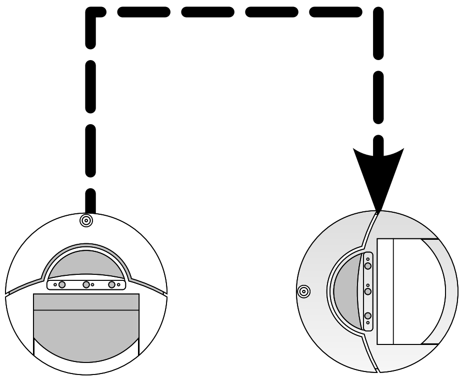
2. Connect other blocks to send commands to the robot.

3. Press the RUN MY PROGRAM button to send your program to the robot.

Flip through this booklet to try some programming challenges

TRY THIS...

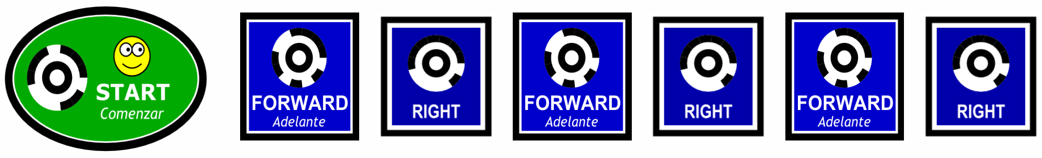
Program the robot to follow this path.



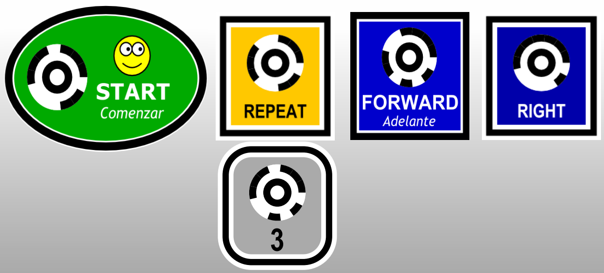
Challenge Level

1

Flip this card over to see possible solutions.

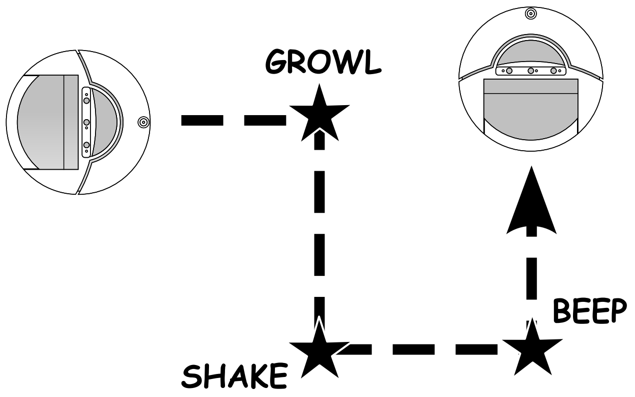


OR



TRY THIS...

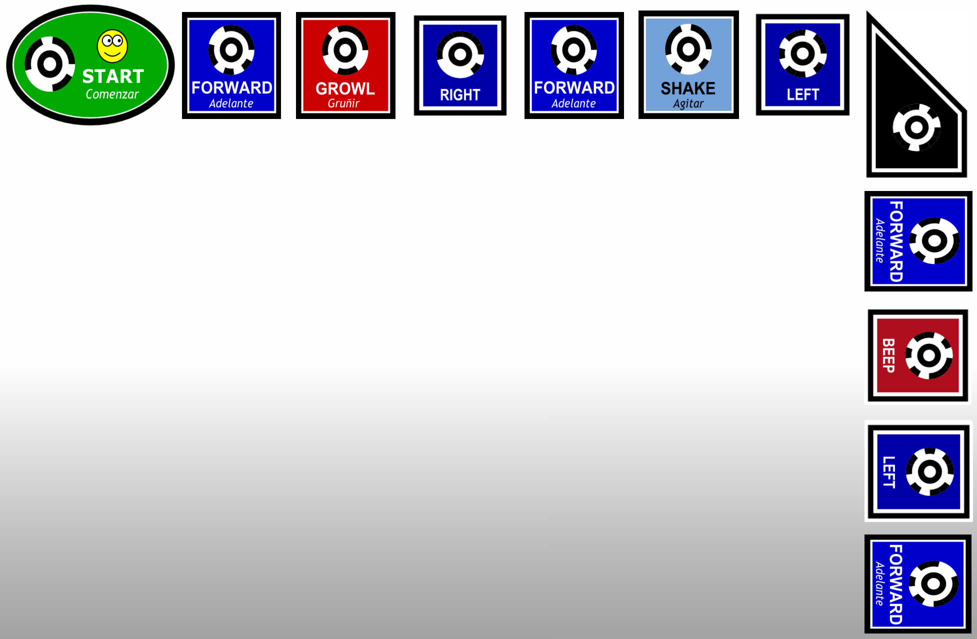
Program the robot to follow this path.



Challenge Level

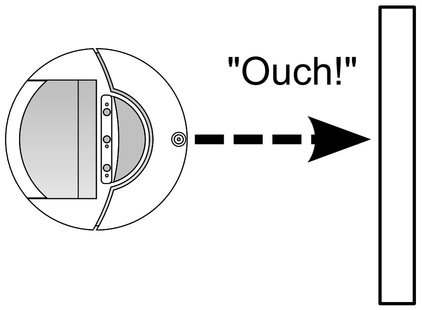
1

Flip this card over to see possible solutions.



TRY THIS...

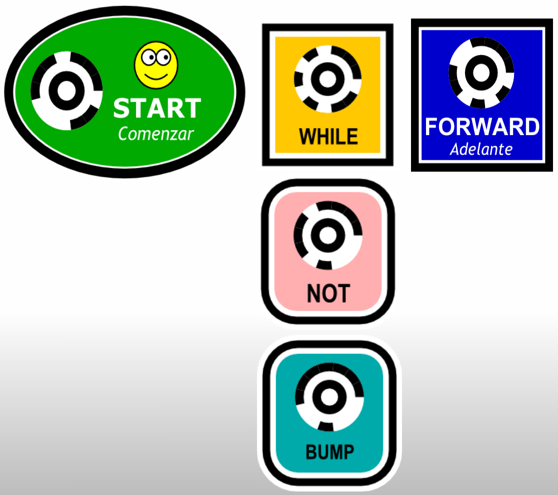
Program the robot to drive forward until it hits a wall.



Challenge Level

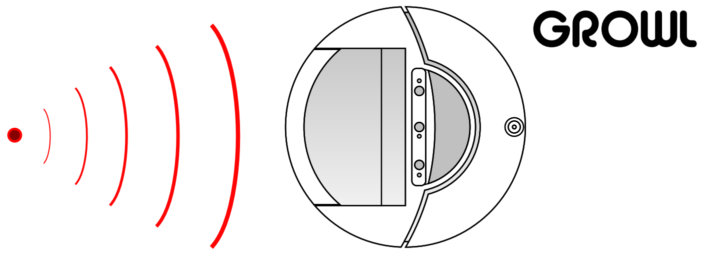
2

Flip this card over to see possible solutions.



TRY THIS...

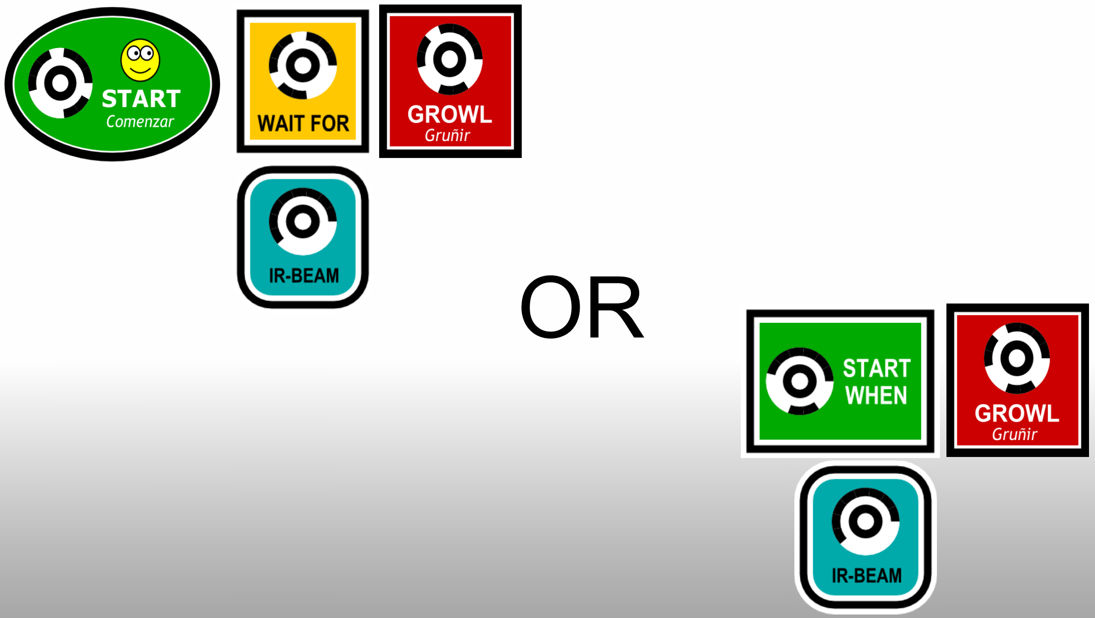
Program the robot to growl when it sees an infrared (IR) beam.



Challenge Level

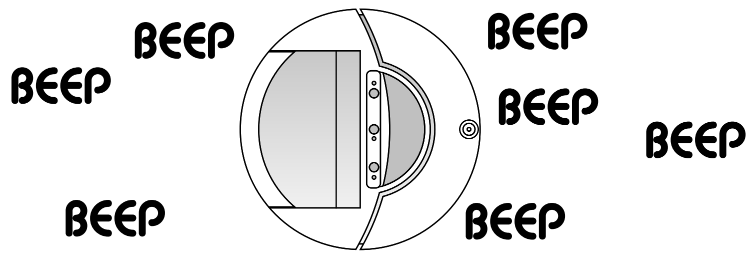
2

Flip this card over to see possible solutions.



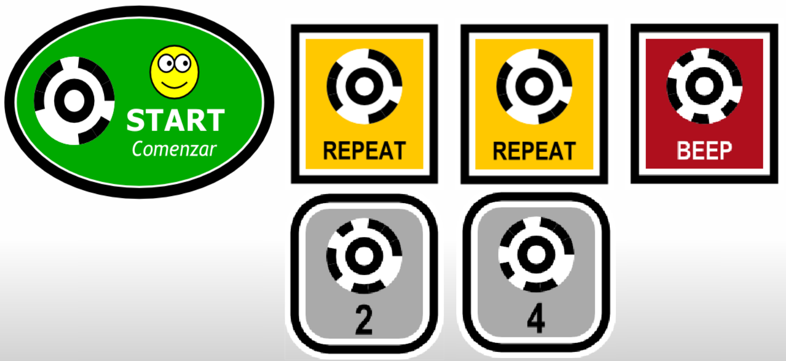
TRY THIS...

Program the robot to beep 8 times in a row. How about 9 times?



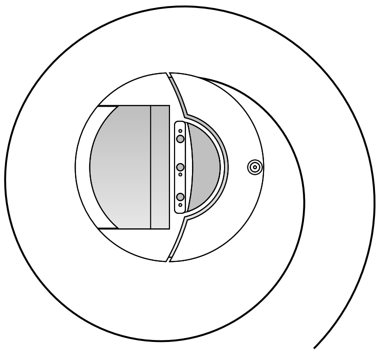
Challenge Level 3

Flip this card over to see possible solutions.



TRY THIS...

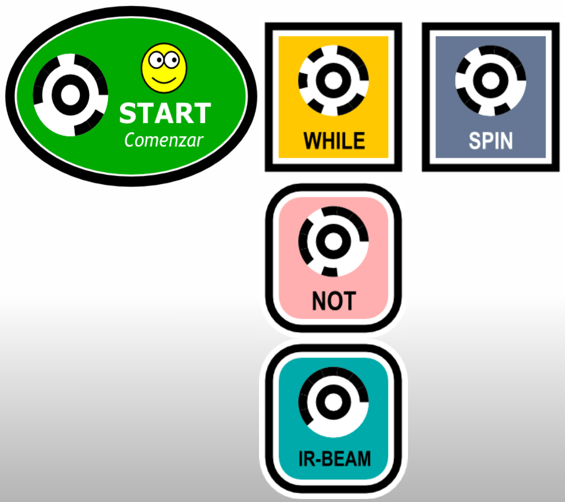
Program the robot to spin until it sees an infrared (IR) beam?



Challenge Level

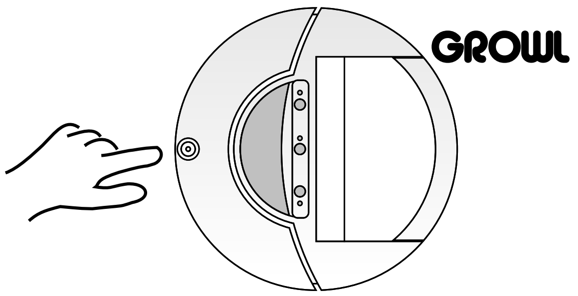
2

Flip this card over to see possible solutions.



TRY THIS...

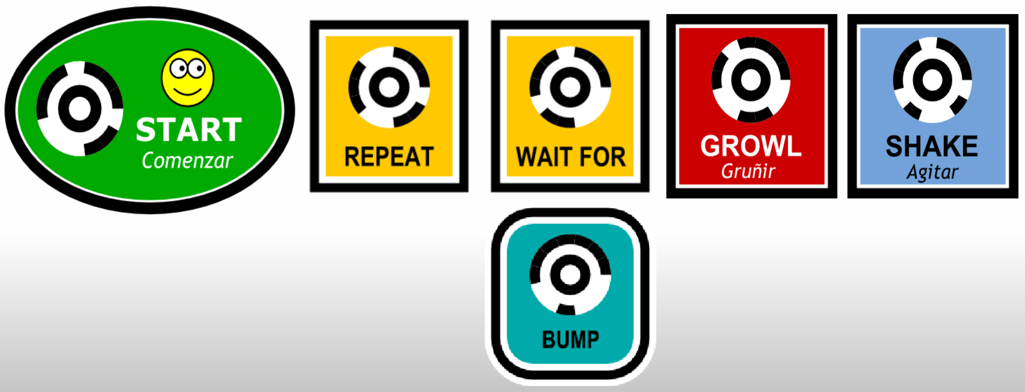
Program a grumpy robot. Every time it gets bumped it should growl and shake!



Challenge Level

3

Flip this card over to see possible solutions.



About this Exhibit

TERN
TANGIBLE
PROGRAMMING

This exhibit was created in collaboration with the Tufts University Human Computer Interaction (HCI) Lab. Tern is a tangible programming language designed to provide a painless introduction to computer programming for children in classrooms, afterschool programs, and other learning settings. For more information, contact:

Michael Horn
michael.horn@tufts.edu
http://hci.cs.tufts.edu/tern

Tangible Programming Exhibit

Block Checklist

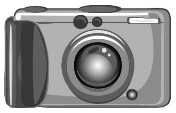
TERN
TANGIBLE
PROGRAMMING
hci.cs.tufts.edu/tern

START	x 1	1	x 2	START SKILL ▲	x 1
WHISTLE	x 2	2	x 2	START SKILL ★	x 1
GROWL	x 2	3	x 2	START WHEN	x 2
SING	x 2	4	x 2	REPEAT	x 2
BEEP	x 2	AND	x 2	WHILE	x 2
SPIN	x 2	OR	x 2	WAIT FOR	x 2
SHAKE	x 2	NOT	x 2	◣	x 2
WIGGLE	x 2	BUMP	x 2	SKILL ▲	x 2
FORWARD	x 4	BUMP-LEFT	x 2	SKILL ★	x 2
REVERSE	x 4	BUMP-RIGHT	x 2	IF? YES NO	x 2
LEFT	x 3	IR-BEAM	x 2		
RIGHT	x 3	CLIFF	x 2		

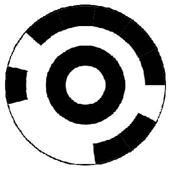
checklist.pdf

How Does This Exhibit Work?

This exhibit uses **computer vision** technology. That means that it uses a digital camera to take a picture of your program and turn it into commands for the robot.

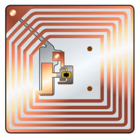


The exhibit computer can *see* the black and white symbols on the wooden blocks. It uses these symbols to turn your program into digital instructions.



These digital instructions are sent to the robot using a wireless **Bluetooth** connection.

Each block also has an **RFID** (Radio Frequency Identification) tag inside it. The Block Tester uses the RFID tags to know which block you are testing.



Bibliography

AAUW (2000). Tech-savvy: Educating girls in the new computer age.

Allen, S. (2002). Looking for learning in visitor talk: A methodological exploration. In G. Leinhardt, K. Crowley, & K. Knutson (Eds.) *Learning Conversations in Museums*, (pp. 259–303). Lawrence Erlbaum.

Allen, S. (2004). Designs for learning: Studying science museum exhibits that do more than entertain. *Science Education*, *88*(S1), S17–S33.

Ananny, M. (2002). Supporting childrens collaborative authoring: Practicing written literacy while composing oral texts. In *Computer-Supported Collaborative Learning CSCL'02*.

Ansel, J. (2003). Real, simple and new. *Informal Learning Review*, *November–December*(63).

Beals, L., & Bers, M. (2006). Robotic technologies: When parents put their learning ahead of their child's. *Journal of Interactive Learning Research*, *17*(4), 341–366.

Ben-Ari, M. (1998). Constructivism in computer science education. In *ACM Conference on Computer Science Education SIGCSE'98*, (pp. 257–261). ACM Press.

Bers, M., Rogers, C., Beals, L., Portsmore, M., Staszowski, K., Cejka, E., Carberry, A., Gravel, B., Anderson, J., & Barnett, M. (2006). Early childhood robotics for learning. In *Symposium at the International Conference on the Learning Sciences*.

Bers, M. U. (2008). *Blocks to Robots: Learning with Technology in the Early Childhood Classroom*. Teachers College Press.

Blackwell, A., Hague, R., & Greaves (2001). Autohan: An architecture for programming the home. In *Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments*, (pp. 150–157).

Boston Museum of Science (2001). Science is an activity.
URL `http://www.mos.org/exhibitdevelopment/pdf/ScienceIsAnActivity.pdf`

Bredekamp, S., & Copple, C. (1997). *Developmentally Appropriate Practice in Early Childhood Programs*. National Association for the Education of Young Children.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, *2*(1), 14–23.

Buechley, L., Eisenberg, M., Catchen, J., & Crockett, A. (2008). The lilypad arduino: Using computational textiles to investigate engagement, aesthetics, and diversity in computer science education. In *Proc. ACM Conference on Human Factors in Computing Systems CHI'08*. ACM Press.

CanonSDK (2004). *Canon Digital Camera Development Kit (version 7.1)*. Canon, Inc. Http://consumer.usa.canon.com.

Cejka, E., Rogers, C., & Portsmore, M. (2006). Kindergarten robotics: Using robotics to motivate math, science, and engineering literacy in elementary school. *International Journal of Engineering Education*, *22*(4), 711–722.

Chipman, G., Druin, A., Beer, D., Fails, J., Guha, M., & Simms, S. (2006). A case study of tangible flags: a collaborative technology to enhance field trips. In *Interaction Design and Children IDC'06*. ACM Press.

Clements, D. (1999a). Concrete manipulatives, concrete ideas. *Contemporary Issues in Early Childhood*, *1*(1), 45–60.

Clements, D. (1999b). The future of educational computing research: the case of computer programming. *Information Technology in Childhood Education*, (pp. 147–179).

Clements, D. (1999c). Young children and technology. In *Dialogue on Early Childhood Science, Math, and Technology Education*. American Association for the Advancement of Science.

Clements, D., & Gullo, D. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology*, *76*(6), 1051–1058.

Clements, D., & Sarama, J. (2002). The role of technology in early childhood learning. *Teaching Children Mathematics*, *8*(6), 340–343.

Conway, M., Pausch, R., Gossweiler, R., & Burnette, T. (1994). Alice: a rapid prototyping system for building virtual environments. In *Proc. ACM Conference on Human Factors in Computing Systems CHI'94*, (pp. 295–296). ACM Press.

Cuban, L. (1984). *Teachers and Machines: The Classroom Use of Technology Since 1920*. Teachers College Press.

Cuban, L. (2001). *Oversold and Underused: Computers in the Classroom*. Harvard University Press.

de Ipina, D., Mendonca, P., & Hopper, A. (2002). Trip: A low-cost vision-based location system for ubiquitous computing. *Personal and Ubiquitous Computing*, *6*, 206–219.

Dede, C., Nelson, B., Ketelhut, J., Clarke, J., & Bowman, C. (2004). Design-based research strategies for studying situated learning in a multi-user virtual environment. In *International Conference on Learning Sciences ICLS'04*. LEA Publishing.

Druin, A. (1999). Cooperative inquiry: Developing new technologies for children with children. In *ACM Conference on Human Factors in Computing Systems CHI'99*, (pp. 592–599). ACM Press.

Fails, J., Druin, A., Guha, M., Chipman, G., Simms, S., & Churaman, W. (2005). Child's play: A comparison of desktop and physical interactive environments. In *Interaction Design and Children IDC'05*, (pp. 48–55). ACM Press.

Fernaeus, Y., & Tholander, J. (2006). Finding design qualities in a tangible programming space. In *Proc. ACM Conference on Human Factors in Computing Systems CHI'06*, (pp. 447–456). ACM Press.

for the Education of Young Children, N. A. (2009). Position statement: Developmentally appropriate practice in early childhood education programs serving children from birth through age 8. *Young Children*.

Frei, P., Su, V., Mikhak, B., & Ishii, H. (2000). Curlybot: designing a new class of computational toys. In *Proc. ACM Conference on Human Factors in Computing Systems CHI'00*. ACM Press.

Haugland, S. (1992). The effect of computer software on preschool children's developmental gains. *Journal of Computing in Childhood Education*, *3*(1), 15–30.

Haugland, S., & Shade, D. (1994). Software evaluation for young children. In J. Wright, & D. Shade (Eds.) *Young children: Active learners in a technological age*. Washington, DC: National Association for the Education of Young Children.

Heath, C., vom Lehn, D., & Osborne, J. (2005). Interaction and interactives: collaboration and participation with computer-based exhibits. *Public Understanding of Science*, *14*, 19–101.

Horn, M. (2006). *Tangible Programming with Quetzal: Opportunities for Education*. Master's thesis, Tufts University.

Horn, M. S., & Jacob, R. J. (2007). Designing tangible programming languages for classroom use. In *First International Conference on Tangible and Embedded Interaction TEI'07*. ACM Press.

Horn, M. S., Solovey, E. T., Crouser, R. J., & Jacob, R. J. (2009). Comparing the use of tangible and graphical programming interfaces for informal science education. In *Proc. ACM Conference on Human Factors in Computing Systems CHI'09*. ACM Press.

Horn, M. S., Solovey, E. T., & Jacob, R. J. (2008). Tangible programming and informal science learning: Making tuis work for museums. In *IDC'08 Interaction Design and Children*. ACM Press.

Hornecker, E., & Buur, J. (2006). Getting a grip on tangible interaction: A framework on physical space and social interaction. In *Proc. ACM Conference on Human Factors in Computing Systems CHI'06*, (pp. 437–446). ACM Press.

Hornecker, E., Marshall, P., & Rogers, Y. (2007). From entry to access — how sharability comes about. In *Designing Pleasurable Products and Interfaces DPPI'07*. ACM Press.

Hornecker, E., & Stifter, M. (2006). Learning from interactive museum installations about interaction design for public settings. In *Australian Computer-Human Interaction Conference OZCHI'06*.

Hourcade, J., Bederson, B., Druin, A., & Guimbretire, F. (2004). Differences in pointing task performance between preschool children and adults using mice. *ACM Transactions on Computer-Human Interaction*, *11*(4), 357–386.

Humphrey, T., & Gutwill, J. (2005). *Fostering Active Prolonged Engagement: The art of creating APE exhibits*. Exploratorium.

Inkpen, K., Booth, K. S., Klawe, M., & Upitis, R. (1995a). Playing together beats playing apart, especially for girls. In *First International Conference on Computer Support for Collaborative Learning (CSCL 1995)*, (pp. 177–181). Lawrence Erlbaum.

Inkpen, K., Gribble, S., Booth, K., & Klawe, M. (1995b). Give and take: Children collaborating on one computer. In *Proc. ACM Conference on Human Factors in Computing Systems (CHI'95)*, (pp. 258–259). ACM Press.

Ishii, H., & Ullmer, B. (1997). Tangible bits: Towards seamless interfaces between people, bits, and atoms. In *Proc. ACM Conference on Human Factors in Computing Systems CHI'97*, (pp. 234–241). ACM Press.

Jacob, R., Girouard, A., Hirshfield, L., Horn, M., Shaer, O., Solovey, E., & Zigelbaum, J. (2008). Reality–based interaction: A framework for post-wimp interfaces. In *Proc. ACM Conference on Human Factors in Computing Systems CHI'08*. ACM Press.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.*, *37*(2), 83–137.

Kelleher, C., Pausch, R., & Kiesler, S. (2007). Storytelling alice motivates middle school girls to learn computer programming. In *Proc. ACM Conference on Human Factors in Computing Systems CHI'07*. ACM Press.

Levy, S., & Mioduser, D. (2008). Does it “want” or “was it programmed to...”? kindergarten children’s explanation of an autonomous robot’s adaptive functioning. *International Journal of Technology Des Education*, *18*, 337–359.

Maeda, J., & McGee, K. (1993). Dynamic form. *Tokyo: International Media Research Foundation*.

Marshall, P. (2007). Do tangible interfaces enhance learning? In *First International Conference on Tangible and Embedded Interaction TEI’07*, (pp. 163–170). ACM Press.

Marshall, P., Price, S., & Rogers, Y. (2003). Conceptualising tangibles to support learning. In *Proceeding of Interaction Design and Children*, (pp. 101–109). ACM Press.

Martin, F., Mikhak, B., & Silverman, B. (2000). Metacricket: A designers’ kit for making computational devices. *IBM Systems Journal*, *39*(3, 4).

Mayer, R. (1981). The psychology of how novices learn computer programming. *Computing Surveys*, *13*(1).

Mazalek, A., Davenport, G., & Ishii, H. (2002). Tangible viewpoints: A physical approach to multimedia stories. In *Multimedia’02*. ACM Press.

McKeithen, K., Reitman, J., Rueter, H., & Hirtle, S. (1981). Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, *13*, 307–325.

McNerney, T. (2000). *Tangible programming bricks: An approach to making programming accessible to everyone*. Master’s thesis, MIT Media Lab.

McNerney, T. (2004). From turtles to tangible programming bricks: explorations in physical language design. *Personal and Ubiquitous Computing*, *8*, 326–337.

Montemayor, J., Druin, A., Farber, A., Simms, S., Churaman, W., & D’Amour, A. (2002). Physical programming: designing tools for children to create physical interactive environments. In *Proc. ACM Conference on Human Factors in Computing Systems CHI’02*, (pp. 299–306). ACM Press.

Norman, D. (1986). Cognitive engineering. In D. Norman, & S. Draper (Eds.) *User centered system design, new perspectives on human-computer interaction*, (pp. 31–61). Lawrence Erlbaum.

Norman, D. A. (1988). *The Design of Everyday Things*. Doubleday.

O’Malley, C., & Fraser, D. S. (2004). Literature review in learning with tangible technologies. Tech. Rep. Report 12, NESTA Futurelab.

Oppenheimer, T. (2003). *The Flickering Mind: The False Promise of Technology in the Classroom and How Learning Can Be Saved*. Random House.

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

Papert, S. (1991). What's the big idea: Towards a pedagogy of idea power. *IBM Systems Journal*, *39*(3–4).

Parkes, A., Raffle, H., & Ishii, H. (2008). Topobo in the wild: longitudinal evaluations of educators appropriating a tangible interface. In *Proc. ACM Conference on Human Factors in Computing Systems CHI'08*. ACM Press.

Pattis, R., Roberts, J., & Stehlik, M. (1995). *Karel the Robot: A Gentle Introduction to the Art of Programming*. John Wiley and Sons, Inc., second ed.

Perlman, R. (1976). Using computer technology to provide a creative learning environment for preschool children. *Logo memo no 24*.

Rader, C., Brand, C., & Lewis, C. (1999). Degrees of comprehension: Children's understanding of a visual programming environment. In *ACM Conference on Human Factors in Computing Systems CHI'99*. ACM Press.

Raffle, H., Parkes, A., & Ishii, H. (2004). Topobo: A constructive assembly system with kinetic memory. In *Proc. ACM Conference on Human Factors in Computing Systems (CHI'04)*. ACM Press.

Resnick, M. (2007). Sowing the seeds for a more creative society. *Learning and Leading with Technology*, (pp. 18–22).

Resnick, M., Bruckman, A., & Martin, F. (1996). Pianos not stereos: creating computational construction kits. *Interactions*, *3*(6).

Resnick, M., Martin, F., Berg, R., Borovoy, R., Colella, V., Kramer, K., & Silverman, B. (1998). Digital manipulatives: new toys to think with. In *Proc. ACM Conference on Human Factors in Computing Systems CHI'98*, (pp. 281–287). ACM Press.

Richardson, K. (1998). *Models of Cognitive Development*. Psychology Press.

Rizzo, F., & Garzotto, F. (2007). The fire and the mountain: Tangible and social interaction in a museum exhibition for children. In *Interaction Design and Children IDC'07*. ACM Press.

Rogers, Y., Scaife, M., Gabrielli, S., Smith, H., & Harris, E. (2002). A conceptual framework for mixed reality environments: designing novel learning activities for young children. *Presence*, *11*(6), 677–686.

Scharf, F., Winkler, T., & Herczeg, M. (2008). Tangicons: Algorithmic reasoning in a collaborative game for children in kindergarten and first class. In *Interaction Design and Children IDC'08*.

Schweikardt, E., & Gross, M. (2008). The robot is the program: interacting with roblocks. In *Conference on Tangible and Embedded Interaction TEI'08*. ACM Press.

Scott, S., Mandryk, K., & Inkpen, K. (2003). Understanding childrens collaborative interactions in shared environments. *Journal of Computer Assisted Learning*, *19*, 220–228.

Scott, S., Shoemaker, G., & Inkpen, K. (2000). Towards seamless support of natural collaborative interactions. In *Graphics Interface*, (pp. 103–110).

Serrell, B. (1996). *Exhibit Labels: An Interpretive Approach*. AltaMira Press.

Smith, A. (2007). Using magnets in physical blocks that behave as programming objects. In *First International Conference on Tangible and Embedded Interaction TEI'07*. ACM Press.

Smith, A. (2008). Handcrafted physical syntax elements for illitterate children: Initial concepts. In *Interaction Design and Children*.

Sowell, E. J. (1989). Effects of manipulative materials in mathematics instruction. *Journal for Research in Mathematics Education*, *20*(5), 498–505.

Stewart, J., Raybourn, E., Bederson, B., & Druin, A. (1998). When two hands are better than one: Enhancing collaboration using single display groupware. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, (pp. 287–288). ACM Press.

Suzuki, H., & Kato, H. (1995). Interaction-level support for collaborative learning: Algoblock–an open programming language. In *Proceedings of Computer Supported Collaborative Learning '95*. Lawrence Erlbaum.

Ullmer, B. A. (2002). *Tangible Interfaces for Manipulating Aggregates of Digital Information*. Ph.D. thesis, Massachusetts Institute of Technology.

Uttal, D. H., Scudder, K. V., & DeLoache, J. S. (1997). Manipulatives as symbols: a new perspective on the use of concrete objects to teach mathematics. *Journal of Applied Developmental Psychology*, *18*, 37–54.

Vegso, J. (2006). Drop in cs bachelor's degree production. *Computing Research News*, *18*(2).

Vygotsky, L. (1978). *Mind in Society: Development of Higher Psychological Processes*. Cambridge, MA: Harvard University Press.

Wang, X., & Ching, C. (2003). Social construction of computer experience in a first-grade classroom: Social processes and mediating artifacts. *Early Education and Development*, *14*(3), 335–361.

Wellner, P. D. (1993). Adaptive thresholding for the digitaldesk. Tech. Rep. EPC-93-110, EuroPARC.

Wing, J. (2006). Computational thinking. *Communications of the ACM*, (pp. 33–35).

Wyeth, P. (2008). How young children learn to program with sensor, action, and logic blocks. *Journal of the Learning Sciences*, *17*(4), 517–550.

Wyeth, P., & Purchase, H. (2002). Tangible programming elements for young children. In *Proceedings of the ACM Conference on Human Factors in Computing Systems CHI'02 (extended abstracts)*, (pp. 774–775). ACM Press.

Zheng, S., Bromage, A., Adam, M., & Scrivener, S. (2007). Surprising creativity: A cognitive framework for interactive exhibits designed for children. In *Creativity and Cognition*, (pp. 17–26). ACM Press.

Zuckerman, O., Arida, S., & Resnick, M. (2005). Extending tangible interfaces for education: Digital montessori inspired manipulatives. In *Proceedings of SIGCHI Conference on Human Factors in Computing Systems*, (pp. 859–868). ACM Press.

Zuckerman, O., Grotzer, T., & Leahy, K. (2006). Flow blocks as a conceptual bridge between understanding the structure and behavior of a complex causal system. In *Proceedings of the International Conference of the Learning Sciences*.