Tangible Programming in the Classroom with Tern

Michael Horn
Tufts University
Department of Computer Science
Medford, MA 02155 USA
michael.horn@tufts.edu

Robert J.K. Jacob
Tufts University
Dept. of Computer Science
Medford, MA 02155 USA
jacob@cs.tufts.edu

CHI 2007, April 28 – May 3, 2007, San Jose, USA
ACM 978-1-59593-642-4/07/0004.

Abstract
This interactivity demonstrates *Tern*, a tangible programming language for middle school and late elementary school students. Tern consists of a collection of wooden blocks shaped like jigsaw puzzle pieces. Children connect these blocks to form physical computer programs, which may include action commands, loops, branches, and subroutines. With Tern we attempt to provide the ability for teachers to conduct engaging programming activities in their classrooms, even if there are only one or two computers available. In designing Tern, we focused on creating an inexpensive, durable, and practical system for classroom use.

Keywords
Tangible UIs, education, children, programming languages

ACM Classification Keywords
H5.2. Information interfaces and presentation (e.g., HCI): User Interfaces.

Introduction
Incorporating computer programming activities into classroom curriculum can be a daunting task. Students can hide behind large computer monitors where they

have easy access to digital distractions such as games, IM, and the Web. Furthermore, many classrooms have only a few working computers available. To implement a programming activity in such a setting requires shuffling small teams of students between computer and non-computer work. These student teams must crowd around a machine and work out ways to share a single mouse and keyboard. And, because learning the syntax of a computer language can be a frustrating experience for novice programmers, students may require frequent adult help.

Our project attempts to address some of these issues with an educational programming language called *Tern*. Tern employs tangible user interface technology to allow teachers to conduct in-class programming activities while avoiding many of the problems associated with desktop computers.

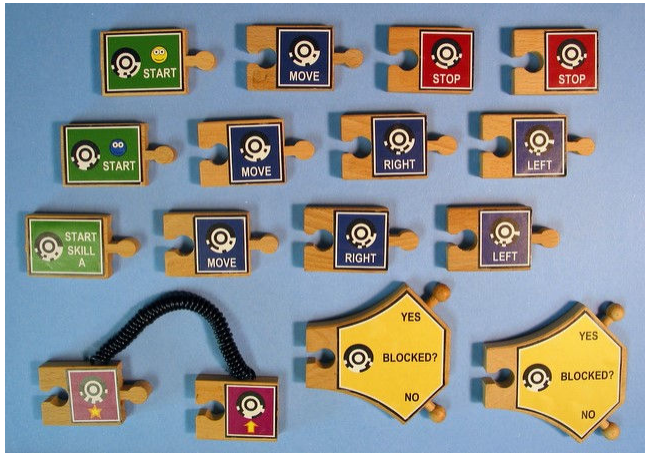

figure 1. Tern consists of a collection of wooden blocks shaped like jigsaw puzzle pieces.

We designed Tern for middle school and late elementary school students. It consists of a collection of wooden blocks shaped like jigsaw puzzle pieces. Children connect these blocks to form physical computer programs that may include action commands, loops, branches, and subroutines. Unlike other tangible programming languages, Tern's parts contain no embedded electronics or power supplies. Instead, we use a digital camera and reliable computer vision technology to *compile* Tern programs into digital code. This allows us to create inexpensive and durable parts that are practical for classroom use. Children work in offline settings, such as on their desks or on the floor, and use a portable scanning station when they are ready to compile. The scanning station consists of a standard digital camera connected to a laptop or tablet PC. Because teams of students no longer need to crowd around a computer screen to write programs, collaboration can be less formal and less constrained.

Tern is based on the text-based language outlined in the book, *Karel the Robot: A Gentle Introduction to the Art of Programming* [6]. With Karel the Robot, students write simple, Pascal-like programs to navigate a robot through a grid world. We chose this language as a starting point because of its simplicity—it has no variables, no parameter values, and a small set of primitives. Like Karel the Robot, Tern programs also control robots that inhabit a grid world on a computer screen (figure 2). We altered the Karel model to allow multiple robots to interact in the same world, thus allowing several teams of students to participate in a shared classroom experience.

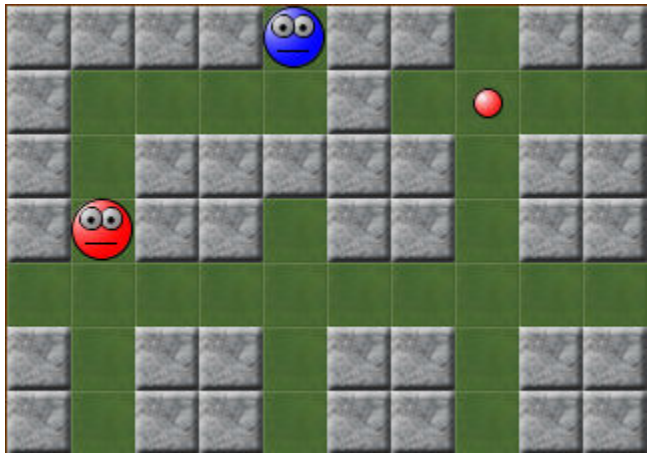

figure 2. Tern programs control virtual robots which inhabit a grid world on a computer screen. Multiple robots can interact in the same world.

Language Overview

Simple Tern programs start with a START statement and end with a STOP statement. For example, the program in figure 3 starts a robot, moves it forward one square, turns it right, and then moves it forward again.

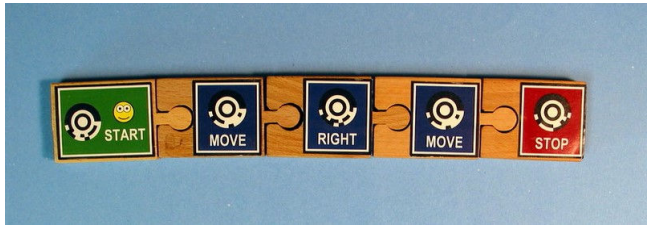

figure 3. A simple Tern program that moves a robot forward one square, turns it right, and then moves it forward again.

In the design of Tern we sought to minimize the possibility of creating programs with syntax errors. The shape of the blocks provides a physical constraint system that prevents many invalid language constructions from being assembled as physical constructions. For example, it is impossible to connect a STOP statement in the middle of a flow-of-control chain. It can only be attached at the end because it lacks an outgoing jigsaw puzzle connector. In a similar way, START statements can only be connected at the beginning of flow-of-control chains. Some syntax errors are still possible—leaving one statement disconnected from the others in a program is an example. When a syntax error occurs, the compiler displays a picture of the original program, an error message, and an arrow indicating the location of the problem.

figure 4. START statements can only be connected at the beginning of a flow-of-control chain because they have no incoming jigsaw puzzle connector.

Using a conditional block, we can program robots to respond to the state of their environment. For example, in the program in figure 5, the robot will move forward only if it is not blocked by a wall. Otherwise, it will turn

right first. Special *JUMP* and *LAND* statements allow the introduction of loops into a program's flow-of-control. These statements are connected by coiled wire that represents the flow of execution as it moves from the JUMP statement to the LAND statement. This construction is similar to a GOTO statement in a text-based language. Tern also offers structured loops that provide more controlled iteration. Like Karel the Robot, Tern includes the ability to create subroutines called *Skills*. Skills can be defined using a START SKILL block and can be invoked using a skill action block (see figure 5).

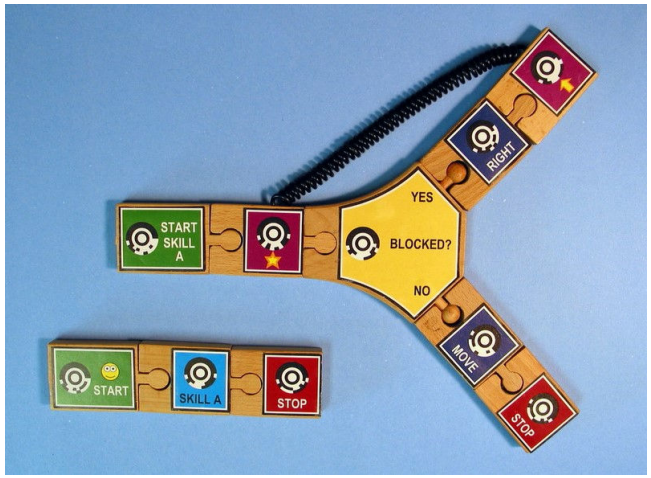

figure 5. This program includes a conditional branch, a loop (represented by coiled wire), and a subroutine.

Tern's technological predecessor is *Quetzal* [4], a tangible language we developed for controlling LEGO MINDSTORMS robots. Quetzal has the appealing property that both its application (physical robot control) and the language itself are disembodied from desktop computers. However, we found that evaluating Quetzal in educational settings was unnecessarily difficult. Students would spend the majority of time constructing LEGO robots and would typically write only one or two very simple Quetzal programs. Furthermore, much of the researchers' time was spent organizing and managing the use of the LEGO kits. With Tern, we will project the grid world onto the wall of a classroom, allowing four teams of children to participate in one shared problem-solving activity. And, because the only way to control an on-screen robot is through a Tern program, we anticipate that students will spend much more of their time discussing and writing programs.

Implementation

The Tern compiler uses a collection of reliable image processing techniques to convert physical programs into digital instructions. Each block in the language is imprinted with a circular symbol called a SpotCode [2, 3]. These codes allow the position, orientation, relative size, and type of each statement to be quickly determined from a digital image. The image processing routines use an adaptive thresholding algorithm [8] and work under a variety of lighting conditions without the need for human calibration. Our prototype uses a digital camera attached to a tablet or laptop PC. The camera has an image resolution set to 1024 x 768. A programming surface approximately 26 inches wide and 20 inches high can be reliably compiled as long as the programming surface is white or light-colored. A Java application controls the flash, optical zoom, and image resolution. Captured images are transferred to the host computer through a USB connection and saved as JPEG images on the file system. With this image, the compiler converts a program directly into digital

instructions. Students initiate a compile by pressing a button on the scanning station, and the entire process takes a few seconds to complete. Any error messages are reported to the user with a picture of the original program and an arrow pointing to the cause of the problem.

Related Work

Several tangible programming languages inspired and influenced Tern. The earliest and most directly relevant language is Suzuki and Kato's AlgoBlocks [7], which represents the commands of a language similar to Logo using interlocking aluminum blocks. More recently, McNerney [5], and Wyeth and Purchase [9] created tangible programming languages consisting of LEGO-like bricks with embedded electronics. Students stack these bricks to describe simple programs. Zuckerman and Resnick's System Blocks project [10] provides an interface for simulating dynamic systems. Wood blocks with embedded electronics express six simple behaviors in a system. Blackwell, Hague, and Greaves developed Media Cubes [1], tangible programming elements for controlling consumer 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.

In each of these examples, the blocks that make up the programming languages contain some form of embedded electronic components. When connected, these blocks provide real-time feedback, typically executing some algorithms through the sequential interaction of the blocks. Our model differs from these languages in that Tern programs are purely symbolic representations of algorithms—much in the way that Java or C++ programs are only collections of text files. A compiler must be used to translate physical programs into digital instructions for controlling robots. This approach cuts cost, increases reliability, and allows us greater freedom in the design of the physical components of the language.

Conclusions and Future Work

Our work with Tern is ongoing. Our next step is to evaluate Tern's effectiveness as a teaching tool in real-life classroom settings. In addition, we hope to better understand the effect of tangible programming on student learning and collaboration. Part of this study might involve a direct comparison of Tern to a comparable visual or text-based programming language designed for classroom use.

In this paper we introduced Tern, a tangible programming language for middle school and late elementary school students to use in classroom settings. We described the design and implementation of Tern, and we described how it differs from other tangible programming languages. Specifically, our languages consist of parts with no embedded electronics or power supplies. Instead of real-time interaction, we use a compiler to convert physical Tern programs into digital instructions. This allows us to create durable and inexpensive parts that are practical classroom use.

Acknowledgements

We thank the National Science Foundation for support of this research (NSF Grant No. IIS-0414389). Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Blackwell, A.F. and Hague, R. Autohan: An architecture for programming in the home. In Proc. IEEE Symposia on Human-Centric Computing Languages and Environments 2001, 150-157.

[2] de Ipina, D.L., Mendonca, P.R.S. and Hopper, A. TRIP: A low-cost vision-based location system for ubiquitous computing. Personal and Ubiquitous Computing, 6 (2002), 206–219.

[3] High Energy Magic. http://www.highenergymagic.com

[4] Horn, M. and Jacob, R.J.K. Tangible Programming in the Classroom: A Practical Approach. Extended Abstracts CHI 2006, ACM Press (2006).

[5] McNerney, T.S. From turtles to Tangible Programming Bricks: explorations in physical language design. Personal Ubiquitous Computing, 8(5), Springer-Verlag (2004), 326–337.

[6] Pattis, R.E., Roberts J., Stehlik, M. Karel the Robot: a Gentle Introduction to the Art of Programming, 2nd edition. John Wiley and Sons, Inc. 1995.

[7] Suzuki, H. and Kato, H. Interaction-level support for collaborative learning: Algoblock–an open programming language. In Proc. CSCL '95, Lawrence Erlbaum (1995).

[8] Wellner, P.D. Adaptive thresholding for the DigitalDesk. Technical Report EPC-93-110, EuroPARC (1993).

[9] Wyeth, P. and Purchase, H.C. Tangible programming elements for young children. *Extended Abstracts CHI 2002*, ACM Press (2002), 774–775.

[10] Zuckerman, O. and Resnick, M. A physical interface for system dynamics simulation. *Extended Abstracts CHI 2003*, ACM Press (2003), 810-811.