Tangible Programming in the Classroom: A Practical Approach

Michael S. Horn
Tufts University
Computer Science
161 College Ave.
Medford, MA 02155 USA
michael.horn@tufts.edu

Robert J.K. Jacob
Tufts University
Computer Science
161 College Ave.
Medford, MA 02155 USA
jacob@cs.tufts.edu

Abstract
This paper introduces Quetzal, a tangible programming language for children to use in educational settings. Quetzal features inexpensive, durable parts with no embedded electronics or power supplies. Children create programs in offline settings—on their desks or on the floor—and carry their programs to a scanning station when they are ready to compile. We argue that a language like Quetzal could offer an appealing and practical alternative to conventional languages for introducing programming concepts in the classroom. This paper discusses the motivations for the Quetzal project and describes the design and implementation of the language. We also outline several key questions that are guiding our research with Quetzal.

Keywords
Tangible UIs, education, children, programming languages

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

Introduction
Over the past decade, research involving tangible user interfaces has created new opportunities for the productive use of technology in the K–12 classroom. Tangible interfaces seem especially applicable to

education, but the technology is often delicate, expensive, and non-standard. This can be a problem in educational settings where cost is always a factor and technology that is not dependable tends to gather dust in the corner. In this setting, we introduce *Quetzal* (pronounced *ket-zal'*), a tangible programming language for children and novice programmers to control LEGO Mindstorms^{TM} robots. Using Quetzal, children arrange and connect physical programming elements to describe algorithms that may include loops, branches, and concurrent execution.

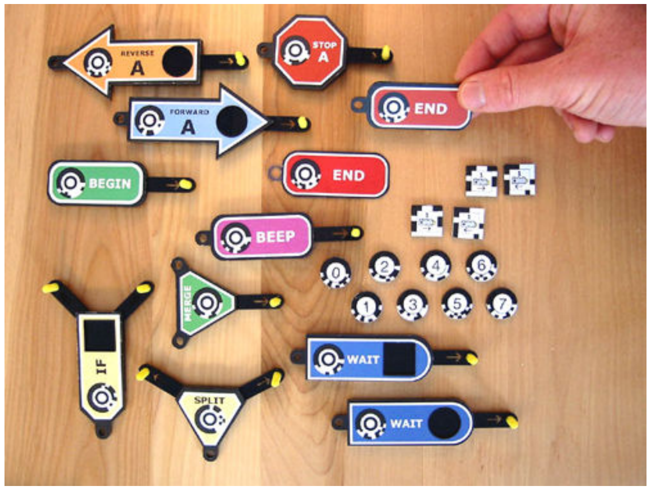

Figure 1: A collection of Quetzal statements and parameter values.

Unlike many other tangible programming languages, Quetzal features inexpensive and durable parts with no embedded electronics or power supplies. Children create programs on their desks or on the floor and carry their programs to a scanning station when they are ready to compile. In this way, teachers can introduce programming concepts to an entire classroom of children even with only one or two desktop computers available.

As a tangible user interface, Quetzal may also offer a number of advantages over conventional programming languages used in classrooms. For example, often children in schools work *on* computers during pre-scheduled "computer time" rather than *with* computers at any time that is valuable and relevant. With Quetzal, programming activities are freed from the confines of the computer desktop. Thus, teachers may be able to more fluidly integrate technology into their curriculum—a goal underscored by the National Association for the Education of Young Children in a position statement on the appropriate use of technology in the classroom [5]. In addition, Quetzal may allow students to collaborate more effectively with each other because the restriction of a single input device is removed. In other words, not only can tangible interfaces facilitate two-handed interaction, they can also support four or six-handed interaction when multiple children collaborate together on a project. Finally, much of the syntax of the language is expressed directly in the physical form of the programming elements and the way in which they interconnect. Many invalid language constructs simply cannot be assembled as physical constructs. This may reduce the confusion and frustration that many students face when learning a new text-based or visual programming language.

It is important to note that tangible programming languages are not yet commercially available, and their use has been restricted almost entirely to laboratory and research settings. Thus, the advantages outlined above are hypothetical. Indeed, one of the primary goals of the Quetzal project is to better understand how tangible languages might affect student learning in classroom environments compared to more conventional languages.

Background

A tangible programming language, like any other type of computer language, is simply a tool for telling a computer what to do. With a text-based language, a programmer uses words such as BEGIN, IF, and REPEAT to instruct a computer. This *code* must be written according to strict, and often frustrating, syntactic rules. With a visual language, words are replaced by pictures, and programs are expressed by arranging and connecting icons on the computer screen with the mouse. There are still syntax rules to follow, but they can be conveyed to the programmer through a rich set of visual.

Instead of relying on pictures and words on a computer screen, tangible languages use physical objects to represent the various programming elements, data abstractions, and flow-of-control structures. Users arrange and connect these physical elements to construct programs. Rather than falling back on implied rules and conventions, tangible languages can exploit the physical properties of objects such as size, shape, and material to express and enforce syntax.

An early example of a tangible language was Suzuki and Kato's AlgoBlocks [7], a system in which interlocking aluminum blocks represented the commands of a language similar to Logo. More recently, McNerney developed Tangible Computation Bricks [4], LEGO blocks with embedded microprocessors. He also described several tangible programming languages that could be expressed with the bricks. In a similar project, Wyeth and Purchase of the University of Queensland created a language for younger children (ages four to eight) also using stackable LEGO-like blocks to describe simple programs [9]. Blackwell, Hague, and Greaves at the University of Cambridge 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 electronic components. When connected, the blocks form structures that are more than just abstract representations of algorithms. They are also working, specialized computers that can execute an algorithm through the sequential interaction of the blocks. Quetzal differs from these languages in that its programs are purely symbolic representations of algorithms—much in the way that Java or C++ programs are only collections of text files. An additional piece of technology (a compiler), must be used to translate the abstract representations of a program into a machine language that will be executed on some computer system. This approach cuts cost and allows us greater freedom in the design of the physical components of the language.

Quetzal Overview

Quetzal is a programming language for controlling the LEGO Mindstorms^{TM} RCX brick. It consists of interlocking plastic tiles that represent flow-of-control structures, actions, and data. Statements in the language are connected together to form flow-of-control chains. Simple programs start with a Begin statement and end with a single End statement. For example, figure 2 shows a program that starts a motor, waits for three seconds, and then stops the motor. Programmers can add or change parameter values to adjust the wait time and the motor's power level. The order in which the statements are connected is important, but the overall shape of a program (an arc in this picture) does not change its meaning.

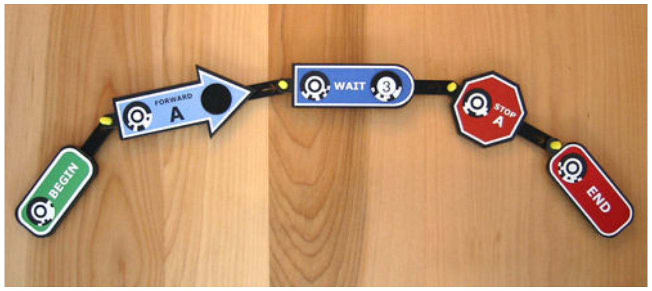

Figure 2: This figure shows a simple Quetzal program that starts a motor, waits three seconds, and then stops the motor

By inserting a Merge statement into the program, we can create an infinite loop (figure 3). Here we don't need an End statement — the robot will execute this program until turned off. With Quetzal loops in the program form physical loops in the flow-of-control chain. Using other statements, programmers can add conditional branches and concurrent tasks (e.g. figure 4). Certain statements also accept parameter values which can include constants and sensor readings. Parameter tokens are plastic tiles with specific shapes to represent their data type. These tiles can be inserted into slots in the top face of statements.

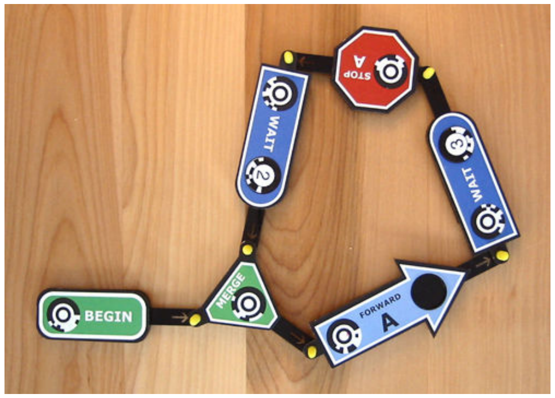

Figure 3: Using a Merge statement creates an infinite loop in the program.

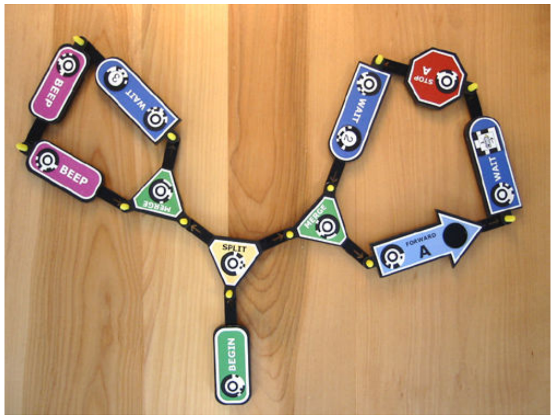

Figure 4: This program features two concurrent tasks. The left branch causes the robot to emit two beeps every three seconds. The right branch starts a motor, waits for a touch sensor to be released, stops the motor, and then repeats.

Implementation

The current implementation of Quetzal uses a collection of image processing techniques to convert physical

Quetzal programs into RCX machine code. Each statement in our prototype 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. Parameter tokens are also imprinted with similar visual codes. The image processing routines use an adaptive thresholding algorithm[8] and work under a variety of lighting conditions without the need for human calibration.

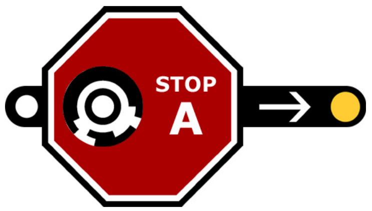

Figure 5: A SpotCode is printed on every statement of the language. This code allows us to determine the position, size, orientation, and type of individual statement in a program.

Our prototype uses a Canon PowerShot S200 digital camera with the image resolution set to 1024 × 768. We suspend the camera approximately 25 inches over a horizontal programming surface by mounting it on the arm of a standard spring-loaded desk lamp. At this height, a programming surface approximately 26 inches wide and 20 inches high is visible to the camera. The programming surface can be any white or light-colored desk or table top. We typically place a white poster board with the correct dimensions on the table where the camera is mounted so that programmers know the exact region visible to the camera. 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 the Quetzal program into a text-based language for the RCX brick called Not Quite C (NQC) [6]. Any error messages are reported to the user. Error messages include a picture of the program with an arrow pointing to the statements that caused the problem.

Initial Usability Testing

We conducted initial usability testing with nine first and second grade children in a weeklong day camp called "Dinosaurs and Robots" conducted at the Eliot Pearson School at Tufts University. As part of the camp, the children used a Quetzal prototype to program robots that they had constructed. This testing helped us resolve some usability problems but also provided encouraging evidence that Quetzal can be viable and appropriate language for use with children in educational environments.

Figure 6: A student constructs a program with Quetzal during usability testing.

Next Steps

Our work with Quetzal is ongoing. We would like to expand the capabilities of the language, improve the existing prototypes, and conducting more formal evaluations of Quetzal's effectiveness in classroom settings. Of these, we feel that the last is most important to discuss in this article. Based on prior research involving other tangible programming languages and on our initial usability sessions with Quetzal, we have formulated several high-level questions that we hope to explore:

1. Are tangible programming languages a viable alternative to more traditional programming languages used in classrooms?
2. Do tangible programming languages offer increased opportunities for learning—especially for students with diverse learning styles?
3. Can tangible languages improve the style and tone of collaboration occurring between students?
4. Can tangible languages improve classroom management when teaching programming concepts?

We believe that it is important to explore these questions in real-life educational settings such as after school programs or classrooms. Ideally, the studies would involve comparisons of entire classrooms of students working together on a programming activities with a tangible language and a similar visual language.

Acknowledgements

We would like to thank the Tufts University Center for Children (TUCC) and the University College of Citizenship and Public Service (UCCPS) for their generous financial support. We would also like to acknowledge the Center for Engineering Education Outreach (CEEO) at Tufts University for materials used in this project. Finally we would like to thank Kevin Joseph Staszowski, Principal Investigator for the Dinosaurs and Robots project.

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 Ltd. http://www.highenergymagic.com

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

[5] National Association for the Education of Young Children. Position statement: Technology and young children—ages three through eight. *Young Children* (1996).

[6] Not Quite C http://bricxcc.sourceforge.net/nqc/

[7] Suzuki, H. and Kato, H. Interaction-level support for collaborative learning: Algoblock–an open programming language. In *Proc. of 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. *Ext. Abstracts CHI 2002*, ACM Press (2002), 774–775.