Growing Up Programming: Democratizing the Creation of Dynamic, Interactive Media

Mitchel Resnick
MIT Media Lab
20 Ames Street
Cambridge, MA 02139 USA
mres@media.mit.edu

Yoshiki Ohshima
Viewpoints Research Institute
1209 Grand Central Ave.
Glendale, CA 91201 USA
yoshiki@vpri.org

Mary Flanagan
Tiltfactor, Dartmouth College
304 North Fairbanks
Hanover, NH 03755 USA
mary@maryflanagan.com

Ken Perlin
Media Research Lab, NYU
719 Broadway, Room 1202
New York, NY 10003 USA
perlin@nyu.edu

Caitlin Kelleher
Washington University
1 Brookings Dr.
St Louis, MO 63130 USA
ckelleher@cse.wustl.edu

Robert Torres
New York University
designbydesign.org
354 West 37th Street, Ste 3
robert@designbydesign.org

Matthew MacLaurin
Microsoft Research
One Microsoft Way
Redmond, WA 98052 USA
mattmac@microsoft.com

CHI 2009, April 4 – April 9, 2009, Boston, MA, USA
ACM 978-1-60558-247-4/08/04.

Abstract
Young people interact with games, animations, and simulations all of the time. But few of them are able to *create* interactive media. The obstacle: traditional programming languages are too difficult to learn and understand. This panel brings together a group of researchers, developers, and educators who are aiming to democratize the activity of programming. They are developing a new generation of programming environments that enable children and teens to create their own interactive games, stories, animations, and simulations. Panelists will discuss and critique their programming environments, then set up interactive demonstration stations for focused exploration and small-group discussion. Audience members will also have the opportunity to download the environments onto their own laptops, so that they can experiment in greater depth.

Keywords
End-user programming, learning, education, children

ACM Classification Keywords
K.3.2 [Computers and Education]: Computer and Information Science Education; D.1.7 [Programming Techniques]: Visual Programming; D.2.6 [Software Engineering]: Programming Environments.

Introduction

Today's world is full of interactive objects. Walk up to a door, and it opens automatically. Play with a toy, and it flashes, beeps, and talks in response to your actions. Connect to a website, and animations react to the movements of your mouse. Log into an online game, and you can interact with fantasy creatures and other virtual objects. Boot up a science simulation, and you can explore the behavior of a bird flock or motions of the planets.

Young people *interact* with these objects everyday. But very few of them can *create* these objects. The creation of interactive objects requires the ability to program, but traditional programming languages are notoriously difficult to learn and understand. As a result, most young people are not fully fluent with digital media – they can "read" but not "write." While Web 2.0 has opened up opportunities for everyone to create and express themselves with text, audio, and video, only a small subset of the population can create and express themselves with *interactive* media.

This panel brings together a group of researchers, developers, and educators who are aiming to democratize the activity of programming. They are developing a new generation of programming environments that enable children and teens to create their own interactive games, stories, animations, and simulations. The goal is to transform programming from a specialized activity for a small sub-community of experts to an everyday activity for a diverse range of participants, so that everyone has an opportunity to become a fully fluent contributor to today's digital society.

Panelists will discuss the ideas underlying their programming environments, show sample projects, and analyze what young people learn as they program within these environments. After the opening presentations, panelists will set up demonstration stations around the room, providing audience members an opportunity to interact with the environments, see more detailed demonstrations and examples, and engage in small-group discussions with the creators of the environments. Audience members will also have the opportunity to download the environments onto their own laptops, so that they can explore and experiment in greater depth. In the final portion of the session, the panelists will reassemble on the stage to answer questions and discuss future directions

Why Programming?

In its report *Being Fluent with Information Technology*, the National Research Council (NRC) defined digital fluency as "the ability to reformulate knowledge, to express oneself creatively and appropriately, and to produce and generate information (rather than simply to comprehend it)…[Fluency] goes beyond traditional notions of computer literacy…[It] requires a deeper, more essential understanding and mastery of information technology for information processing, communication, and problem solving than does computer literacy as traditionally defined."

According to the NRC report, skills associated with programming play a "central role" in the development of fluency. The ability to program offers many important benefits:

- It expands the range of what you can create with software, so that you are no longer limited to the

features provided by standard applications – transforming your relationship with digital technology from "consumer" to "creator."

- It helps you develop a deeper understanding of how computers work, enabling you to use computer applications more effectively and analyze them more critically.
- It offers a meaningful context for learning important mathematical concepts, including some concepts that are already taught in pre-college curricula (such as "variables") and others that are typically seen as too advanced for pre-college students (such as "feedback" and "emergence").
- It supports the development of "computational thinking," providing experience with important problem-solving and design strategies (such as modularization and iterative design) that carry over to non-programming domains. By providing an external representation of your problem-solving processes, it also offers opportunities to reflect on your own thinking – and to think about thinking itself.

Previous Research

When personal computers were first introduced in the late 1970s and 1980s, there was initial enthusiasm for teaching all children how to program. Thousands of schools taught millions of students to write simple programs in the Logo or Basic programming languages. Seymour Papert's book *Mindstorms* presented Logo as a cornerstone for rethinking approaches to education and learning. Although some children and teachers were energized and transformed by these new possibilities, most schools soon shifted to other uses of computers. In the past 20 years, computers have become a pervasive presence in children's lives, but few children learn to program. Today, most people view computer programming as a narrow, technical activity, appropriate only for a small segment of the population.

What happened to the initial enthusiasm for introducing programming to children? Why did Logo and other initiatives not live up to their promise? There were many factors:

- Early programming languages were too difficult to use. Many children had difficulty mastering the syntax of programming languages.
- Programming was often introduced with activities (generating lists of prime numbers, or making simple line drawings) that were not connected to children's interests or experiences.
- Children did not have access to a "literature" of interesting computer programs. Whereas young writers are often inspired by reading great works of literature, there was no analogous literature of programming projects to inspire new programmers.
- Programming was often introduced in contexts where no one had the expertise needed to provide guidance when things went wrong – or encourage deeper explorations when things went right.

Featured Projects

The panel will feature six programming environments, each designed to make the activity of programming more intuitive and the core concepts of programming more understandable. The projects overcome limitations of earlier initiatives by building upon recent HCI research in the areas of end-user programming, graphical interface design, collaborative learning, and

interaction design for children. The panelists are all core members of the design teams for their respective projects.

Alice 2 and ***Storytelling Alice*** are designed to enable novice programmers to learn basic programming constructs while creating their own 3D animations and games. Alice 2 targets college and high school students learning computer programming in a formal classroom setting. Storytelling Alice is designed for middle school students, particularly girls. *(Caitlin Kelleher)*

Boku is an exploratory programming environment situated within a modern, high-quality 3D video game running on either a PC or an XBox 360 game console. In Boku, kids can interactively edit the world, place characters, and give those characters autonomous behaviors using a purpose-built iconic programming language. Typing is optional: all programming is done with a standard game controller. *(Matthew MacLaurin)*

Etoys (http://squeakland.org) is an interactive multimedia authoring tool designed for children (around 5^{th}-6^{th} graders) which draws upon the ideas of Logo, Smalltalk, Hypercard, and StarLogo. Built on top of the Squeak programming language, Etoys pioneered a tile-scripting interface in which a user can drag and drop graphical tiles to construct "scripts" for the multimedia objects. The creators of Etoys envisioned the use of computers in mathematics, science, and computing education. Etoys offers features that make it easy to describe the discrete form of differential equations, and support materials for Etoys demonstrate how to model physical phenomenon in Etoys. *(Yoshiki Ohshima)*

Gamestar Mechanic is an RPG (Role-Playing Game) style online game where middle and high school-age players learn the fundamentals of game design by playing roles as “game mechanics” charged with the making and “modding” (modifying) of games. The game’s online social networking feature allows player-designers to play and comment on each other’s games. *Gamestar Mechanic* is a collaborative research and development project between Gamelab, a game company in New York, and the Games, Learning, and Society Program at the University of Wisconsin, Madison. *(Robert Torres)*

RAPUNSEL is the name of the research project that developed the PEEPS game for "real-time, applied programming for underrepresented students' early literacy (RAPUNSEL)." The team's design goal was to develop an entertaining venue for programming education among middle school girls in informal settings. Our goal of addressing girls in particular through the design is related to gender equity and the digital divide. *(Mary Flanagan and Ken Perlin)*

Scratch enables young people (ages 8 and up) to create their own interactive stories, games, and animations – and share their creations on the web. Scratch is designed to make programming more *tinkerable*, more *meaningful*, and more *social*. Since Scratch was launched in May 2007, more than 300,000 projects have been shared on the Scratch website (http://scratch.mit.edu), which has been called “the YouTube of interactive media.” As young people create and share Scratch projects, they learn to think creatively, reason systematically, and work collaboratively. *(Mitchel Resnick)*