Scratch: A Sneak Preview

John Maloney
MIT Media Laboratory
jmaloney@media.mit.edu

Leo Burd
MIT Media Laboratory
leoburd@media.mit.edu

Yasmin Kafai
UCLA
kafai@gseis.ucla.edus

Natalie Rusk
MIT Media Laboratory
nrusk@media.mit.edu

Brian Silverman
MIT Media Laboratory
bss@media.mit.edu

Mitchel Resnick
MIT Media Laboratory
mres@media.mit.edu

Abstract

Scratch is a networked, media-rich programming environment designed to enhance the development of technological fluency at after-school centers in economically-disadvantaged communities. Just as the LEGO MindStorms robotics kit added programmability to an activity deeply rooted in youth culture (building with LEGO bricks), Scratch adds programmability to the media-rich and network-based activities that are most popular among youth at after-school computer centers. Taking advantage of the extraordinary processing power of current computers, Scratch supports new programming paradigms and activities that were previously infeasible, making it better positioned to succeed than previous attempts to introduce programming to youth.

Our working hypothesis is that, as kids work on personally meaningful Scratch projects such as animated stories, games, and interactive art, they will develop technological fluency, mathematical and problem solving skills, and a justifiable self-confidence that will serve them well in the wider spheres of their lives.

1. Introduction

A flurry of recent U.S. policy reports ([NRC, 1999]; [ITEA, 2000]; and [NAE, 2002]) have drawn attention to a critical societal problem: even as new technologies proliferate and play increasingly important roles in all aspects of society, most people are "poorly equipped to recognize, let alone ponder or address, the challenges technology poses or the problems it could solve" [NAE, 2002].

To address this problem, these reports call for new initiatives to help people become more fluent with technologies. The NRC report defines "fluency" with information technologies 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, according to the report, "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."

In the past, most initiatives to improve technological fluency have focused on school classrooms. But there is a growing recognition that after-school centers and other informal learning settings can play an important role, especially in economically-disadvantaged communities, where schools typically have few technological resources and many young people are alienated from the formal education system.

During the past decade, more than 2000 community technology centers (CTCs) opened in the United States, specifically to provide better access to technology in economically-disadvantaged communities. But most CTCs support only the most basic computer activities such as word processing, email, and Web browsing, so participants do not gain the type of fluency described in the NRC report. Similarly, many after-school centers (which, unlike CTCs, focus exclusively on youth) have begun to introduce computers, but they too tend to offer only introductory computer activities, sometimes augmented by educational games.

A small subset of after-school centers and CTCs, such as those in the Computer Clubhouse network [Resnick, Rusk, & Cooke, 1998], explicitly focus on the development of technological fluency, moving beyond basic computer skills and helping youth learn to design, create, and invent with new technologies. Walk into any Computer Clubhouse and you are likely to see youth creating and manipulating graphics, animations, videos, and music. The professional

image-processing tool Photoshop is particularly popular. Indeed, a "Photoshop culture" has emerged at many Clubhouses, with youth proudly displaying their Photoshop creations on bulletin boards (both physical and online), sharing Photoshop techniques and ideas with one another, and helping Clubhouse newcomers get started with the software.

The rise of the Photoshop culture indicates that after-school computer centers such as the Computer Clubhouse can make a positive difference. But there is a further step on the path towards true computer fluency that is seldom achieved, even at after-school computer centers that focus on technological fluency: youth rarely become engaged in computer programming. There is no "programming culture" analogous to the Photoshop culture. This is unfortunate, because skills associated with programming can play a central role in fluency. "The algorithmic thinking inherent in programming," writes the NRC, "is essential to comprehending how and why information technology systems work as they do." [NRC, 1999]. In addition, the NRC report argues that "the continual use of abstract thinking in programming can guide and discipline one's approach to problems in a way that has value well beyond the information technology-programming setting. In essence, programming becomes a laboratory for discussing and developing valuable life skills, as well as one element of the foundation for learning about other subjects." Many others (e.g., [Papert, 1980]; [Kay, 1991]; and [diSessa, 2000]) have made similar arguments.

Unfortunately, previous initiatives to introduce programming to youth have often failed. Computer programming has been introduced using programming languages that are difficult to use, with proposed activities that do not connect with young people's interests and in contexts where no one has enough expertise to provide guidance. As a result, many people now view computer programming as a difficult technical activity, appropriate only for a small segment of the population. But that need not be the case. The extraordinary increase in computational power over the past two decades makes possible a new generation of programming tools and activities that can help overcome the shortcomings of previous initiatives, making computer programming more accessible to everyone.

2. The Computer Clubhouse network

Scratch is being designed with a particular group in mind: youth ages 10-18 from economically-disadvantaged and culturally diverse communities. In particular, we are focusing on the Computer Clubhouse (www.computerclubhouse.org). We will test Scratch with—and solicit feedback from—youth, mentors, and coordinators at Computer Clubhouse sites in Boston, Los Angeles, and Dublin.

The MIT Media Laboratory co-founded the first Computer Clubhouse in 1993 in collaboration with The Computer Museum (which is now part of the Boston Museum of Science). Since then, the Computer Clubhouse Network, with major financial support from Intel, has grown to more than 85 sites, including over 50 Clubhouses in the United States and international Clubhouses in India, China, Taiwan, Philippines, Australia, Mexico, Costa Rica, Panama, Colombia, Brazil, Ireland, Denmark, Netherlands, Germany, Israel, and South Africa. The Computer Clubhouse Network serves more than 20,000 youth members around the world. In 1997, the Clubhouse received the prestigious Peter Drucker Award for Nonprofit Innovation.

Youth at Computer Clubhouses work on design projects based on their own interests and the needs of their communities. Adult staff and volunteer mentors provide technical, intellectual, and emotional support for the youth. Clubhouse youth use leading-edge software to create artwork, animations, and musical compositions. In addition to the Photoshop culture mentioned earlier, many Clubhouses have also developed a thriving music-production culture. However, to date only a handful of Clubhouse youth have become deeply engaged in computer programming. We hope that Scratch will change that.

Figure 1: Youth at a Computer Clubhouse

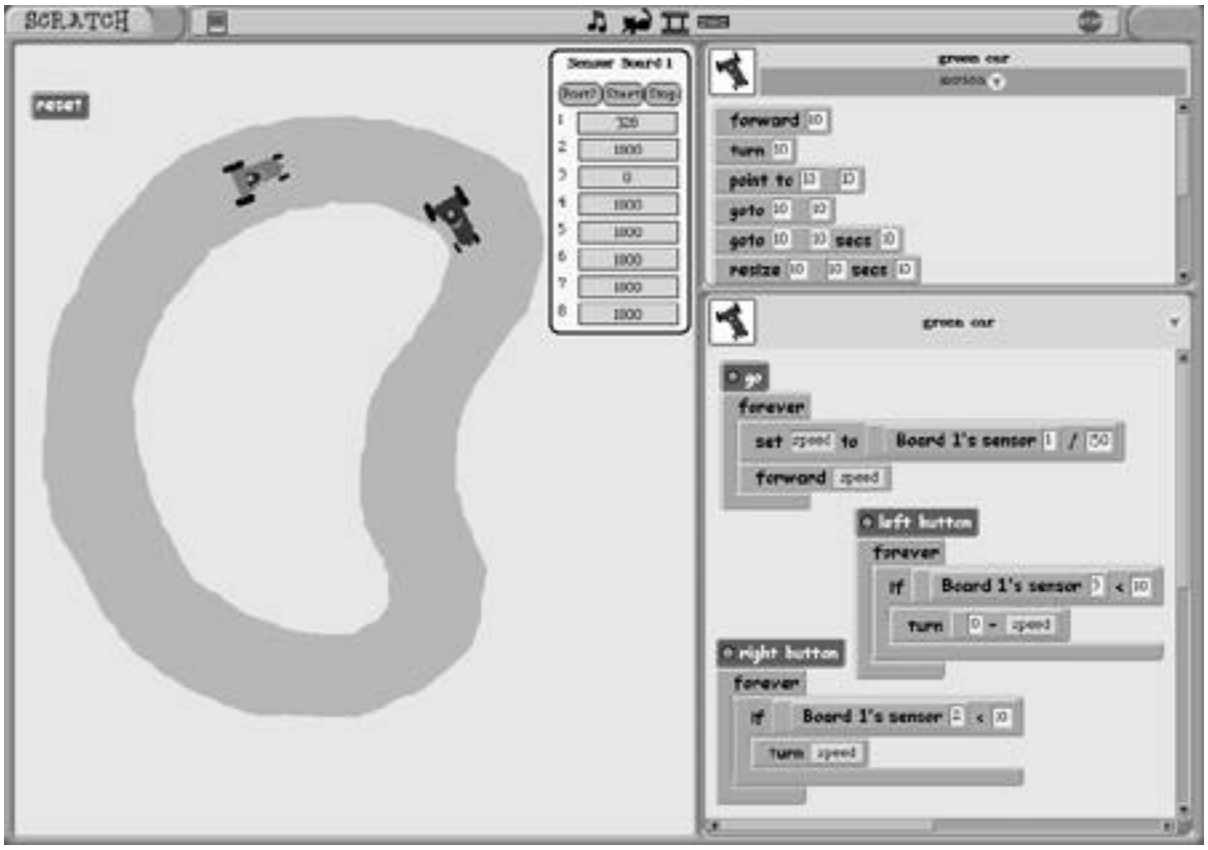

Figure 2: Using the current version of Scratch to build a two-player car racing game. The left pane is the content area containing Scratch objects (cars, road), a button, and the sensor display. The top-right pane is the palette of command blocks. The bottom-right pane is the scripting area for the selected object ("green car") containing three scripts. The variable "speed" is set from a sensor input connected to a variable resistor. Two other sensor inputs (connected to buttons) make the dark green car turn left and right.

3. Scratch design and features

3.1 Scratch design

The design and development of Scratch is being guided by the needs and constraints of Computer Clubhouses. From years of experience in Clubhouses, we have found that new software tools succeed at Clubhouses if:

- youth see the value and potential of the tool right away; it resonates with their interests and passions
- youth can create end-products that they can show off to others
- the tool supports a wide range of different types of activities that appeal to youth of different ages, genders, backgrounds, and cultures
- activities supported by the tool fit into the social dynamic of the Clubhouse; becoming proficient with that tool becomes "cool"
- youth can get started with the tool quickly
- youth can learn additional features incrementally
- youth use the tool in more complex ways over time; they can grow into it over several years

3.2 Core features

With these design criteria in mind, we are designing Scratch with the following set of core features. These features were chosen specifically to address problems that derailed many earlier efforts to introduce programming to youth.

Building-block programming. Scratch is based on a building-block metaphor, in which learners build scripts by snapping together graphical blocks much like pieces in a jigsaw puzzle. Commands and data

types are represented by blocks of different shapes, with pieces fitting together in only syntactically-correct ways. This approach eliminates syntax errors (which have proven to be a major obstacle for learning text-based programming languages), allowing youth to focus on the problems they want to solve, not the mechanics of programming. The Scratch user drags command blocks from a palette to create "stacks" (procedures) that govern behaviors of their objects. Multiprocessing is smoothly integrated into Scratch: different stacks of blocks can execute in parallel.

Programmable manipulation of rich media. Initial activities in traditional programming environments typically involve manipulation of numbers, strings, or simple graphics. In contrast, Scratch programs manipulate images, animations, movies, and sound. By giving Clubhouse youth control over rich media, Scratch supports programming activities that resonate more strongly with youth interests. For example, Scratch will include image filters similar to the ones in Photoshop, but under program control so youth can create projects in which the parameters of a filter vary over time or in response to sensor inputs.

Deep shareability. Computer Clubhouses have an important social component. Youth are constantly looking at one another's projects, trading ideas, sharing techniques. To fit into this context, the object architecture of Scratch supports what we call "deep shareability." Youth will be able to export objects at all levels, from individual animated characters to full projects, and exchange them with friends running Scratch on many types of devices—desktops, laptops, tablets, handhelds, and perhaps even mobile phones or embedded devices.

Integration with the physical world. Building on our previous research on LEGO/Logo and programmable bricks (e.g., [Resnick, 1993]; [Resnick, Berg, Eisenberg, 2000]), inputs from physical sensors (such as switches, sliders, distance sensors, motion detectors, sound sensors) can be used to control the behavior of Scratch creations. For example, a Clubhouse member could connect an accelerometer to her arm and program an animated character to change its behavior based on how she moves her arm, in the process gaining new insights into the concepts of acceleration, sensing, and feedback.

Support for multiple languages. The Computer Clubhouse is a global community, with sites in more than a dozen countries, with young people speaking many different languages. Even in a single Clubhouse in the U.S., it is not unusual to hear three or four different languages spoken. To support collaboration and sharing in this context, it is essential for Scratch to be a multi-language, multi-cultural environment. In developing Etoys [Steinmetz, 2001] and LogoBlocks [Begel, 1996], we discovered that the building-block programming approach makes it easy to handle multiple languages and character sets. Like Etoys, Scratch will allow the user to switch between languages dynamically, fostering cross-cultural collaboration both within a single Clubhouse and between youth in countries halfway around the globe.

3.3 Implementation

Scratch is written in Squeak, an open-source implementation of the Smalltalk-80 language (www.squeak.org). Squeak is extremely portable, with existing implementations for desktop platforms (Windows, Macintosh, Linux/Unix, Acorn, BeOS), handhelds (Windows CE, Zaurus OS, Compaq "Itsy"), and game consoles (Sony Playstation). Squeak has even been ported to hardware without any underlying operating system at all. This extreme portability will allow Scratch applications to be deployed on devices with a wide range of form factors, from desktops to pen-based tablet computers to handhelds.

Sharing and exchanging of Scratch projects and their components will be supported through a combination of standard web servers (with content viewed in a web browser) and a custom "Scratch Object Library" server. The latter will allow Scratch components from personal, Clubhouse, and Clubhouse network-wide libraries to be queried, explored, and downloaded without leaving the Scratch environment.

Scratch source code will be made freely available via periodic code releases to allow collaborators to augment the core system with their own custom features and extensions.

4. Three scenarios

In this section, we present three short scenarios of how youth might use Scratch at Computer Clubhouses.

Programmable image processing. In Clubhouses today, youth often use Photoshop filters (e.g., blur, distort, pixelate, sharpen) to manipulate and transform photographs and scanned images. Scratch adds the ability to control the image-filtering process through programming, expanding the expressive possibilities. For example, a Clubhouse member could build a Scratch program to transform an image by adding color-to-monochrome or hue-shifting effects that vary over time. Later, she might place a virtual fish-eye lens at the edge of an image and program the motion of that lens so that it gradually spirals in. By combining these programs with sensor input, she could create an intriguing piece of interactive video art.

Sensor-controlled music. Many Clubhouse members enjoy the arcade game "Dance Dance Revolution" (DDR), in which players dance on a floor

pad with embedded sensors, aiming to synchronize their movements with music and images on the screen. With Scratch, Clubhouse youth could create their own version of DDR. A Clubhouse member might download MIDI files of songs from the Web (or compose and mix new songs in the Clubhouse music studio), design a floor pad with four touch sensors, connect the sensors to the computer, then create programs that check how well the player's dance steps synchronize with the music. Later, another Clubhouse member might decide to use the floor pad to generate music (rather than to follow it). She would write programs to map the floor pad sensor inputs to different sound and music clips.

Networked animations. Making animations (with tools such as Macromedia Flash) is an increasingly popular activity at Clubhouses. With Scratch, Clubhouse members will be able to create an animation, upload it to a Scratch library server, and then track how it is used or modified by others. A future version of Scratch may even allow youth to download animations to handheld devices and exchange them via IR or Bluetooth. Youth can modify animations that they receive (since all Scratch "program blocks" are accessible), or they could even program an object to behave differently depending on the age, gender, or location of the person receiving it. The Scratch server will automatically keep track of all transactions, so youth can view tree-like graphs representing the spread of their animations, with indicators of how and where the animations have been modified. Through these activities, we hope that an *ecosystem of Scratch creations* will develop, with Clubhouse youth trading and modifying one another's creations.

5. Related programming environments

The Scratch design builds on a number of earlier programming environments designed for young people or novice programmers. The building-block approach grew out of previous research on LogoBlocks, which served as the basis for the programming language used in LEGO MindStorms, and Squeak's Etoys, which has been successfully used in classroom settings for over five years. Alice2 [Pausch, 1995]—itself inspired by Etoys—also employs a building-block approach to make programming easier for novices.

The Scratch user interface was partly inspired by Logo Microworlds. Like AgentSheets [Repenning & Ambach, 1996], Scratch encourages sharing of projects and components on the web and, like Boxer [diSessa, 2000], Scratch makes program elements such as variables into visible, manipulable objects on the screen.

6. Project status and next steps

The Scratch project is a multiple year effort. A series of working prototypes will be built, tested, and revised. Implementation of the first prototype began in January of 2003, based on design discussions over the previous year. This prototype, Scratch 0.1, was tested in early October of 2003 by Harvard and MIT students taking a seminar course on the use of technology for learning in informal educational settings. This test lead to a redesign of Scratch (currently in progress) with several important changes:

• *A place for global scripts*. Many of the student projects included scripts to setup and start a number of other objects. While a few students created a "master of ceremonies" object to hold these global setup and start scripts, quite a few others felt that there should be a centralized place for such global scripts. The next Scratch prototype will have provide one.

• *Only one kind of object.* Scratch 0.1 has a different kind of programmable object for each different type of media (still images, movies, and sounds), and the set of command blocks available in the palette changes as different objects are selected. While the notion of different kinds of objects having different sets of operations (i.e., the object-oriented programming model) has become common among professional programmers, it proved confusing for the students in the seminar. Also, some students wanted to create objects with a combination of functions that existed in separate object types (for example, a movie that rotated as it moved around the screen), and Scratch 0.1 did not provide a way to do that. We concluded that it would be better to have only a single type of object in the next prototype. This means that the palette of command blocks will be larger—since it must subsume the functionality of the three object types of Scratch 0.1—but the fact that the palette won't depend on which object is selected should prove less confusing.

• *No inter-object script invocation.* In Scratch 0.1, one object can invoke another object's command blocks and scripts. For example, a "cat" object could invoke a "dog" object's "chase" script. This feature was heavily used. As a result, very few of the student-created objects were self-contained; most objects had scripts that invoked scripts in other objects. Objects with such inter-object references cannot be easily exported and used in other projects (which would lack the object being referred to), making it difficult to support the free-form object exchange that we envision for Scratch. In fact, we realized that Scratch 0.1 actually encourages the construction of non-exchangeable objects. Scratch 0.2 will therefore disallow directly inter-object script invocation, and we

are in the process of designing more loosely-coupled ways for objects to interact.

7. Conclusions

It is too early in the development of Scratch to draw any conclusions. We have not yet begun formal testing at Computer Clubhouses, although initial informal interactions with Computer Clubhouse members have gotten a positive response. We will spend the spring of 2004 creating the Scratch 0.2 prototype which we hope to begin testing at Computer Clubhouses by summer 2004.

Acknowledgements

We are grateful to generous support from the National Science Foundation (award EIA-0325828), Intel Corp., Intel Foundation, and the LEGO Company. Portions of this paper previously appeared in our proposal to the National Science Foundation.

References

[Begel, 1996] Begel, A. (1996). LogoBlocks: A Graphical Programming Language for Interacting with the World. Unpublished Advanced Undergraduate Project, MIT Media Lab.

[diSessa, 2000] diSessa, A. (2000). *Changing Minds: Computers, Learning, and Literacy*. MIT Press: Cambridge, MA.

[ITEA, 2000] International Technology Education Association (2000). *Standards for Technological Literacy*.

[Kay, 1991] Kay, A. (1991). Computers, Networks and Education, *Scientific American*, September, 1991, pp. 138-48.

[NAE, 2002] National Academy of Engineering and National Research Council (2002). *Technically Speaking: Why All Americans Need to Know More About Technology*.

[NRC, 1999] National Research Council (1999). *Being Fluent with Information Technology*. National Academy Press: Washington DC.

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

[Pausch, 1995] Pausch, R., et al. (1995). Alice: Rapid Prototyping System for Virtual Reality. IEEE Computer Graphics and Applications.

[Repenning & Ambach, 1996] Repenning, A. and Ambach, J. (1996). Tactile programming: A unified manipulation paradigm supporting program comprehension, composition, and sharing. IEEE Symposium on Visual Languages, Boulder, CA.

[Resnick, 1993] Resnick, M. (1993). Behavior Construction Kits. *Communications of the ACM*, vol. 36, no. 7, pp. 64-71.

[Resnick, Berg, Eisenberg, 2000] Resnick, M., Berg, R., and Eisenberg, M. (2000). Beyond Black Boxes: Bringing Transparency and Aesthetics Back to Scientific Investigation. *Journal of the Learning Sciences*, vol. 9, no. 1, pp. 7-30.

[Resnick, Rusk, & Cooke, 1998] Resnick, M., Rusk, N., and Cooke, S. (1998). The Computer Clubhouse: Technological Fluency in the Inner City. In Schon, D., Sanyal, B., and Mitchell, W. (eds.), *High Technology and Low-Income Communities*, pp. 266-286. MIT Press: Cambridge, MA.

[Steinmetz, 2001] Steinmetz, J. (2001). Computers and Squeak as Environments for Learning. In Rose, K. and Guzdial, M. (eds.), *Squeak: Open Personal Computing and Multimedia*. Prentice Hall: New York.