Scratch: Programming for Everyone

Mitchel Resnick, John Maloney, Andrés Monroy-Hernández,
Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner,
Eric Rosenbaum, Jay Silver, Brian Silverman, Yasmin Kafai

Accepted for publication in Communications of the ACM (CACM).
Please do not circulate.

When Moshe Vardi, Editor-in-Chief of CACM, invited us to submit an article about Scratch, he shared the story of how he learned about Scratch:

> A couple of days ago, a colleague of mine (CS faculty) told me how she tried to get her 10-year-old daughter interested in programming, and the only thing that appealed to her daughter (hugely) was Scratch.

That's what we were hoping for when we set out to develop Scratch six years ago. We wanted to develop an approach to programming that would appeal to people who hadn't previously imagined themselves as programmers. We wanted to make it easy for everyone, of all ages, backgrounds, and interests, to program their own interactive stories, games, animations, and simulations – and to share their creations with one another.

Since the public launch in May 2007, the Scratch website (http://scratch.mit.edu) has become a vibrant online community, with people sharing, discussing, and remixing one another's projects. Scratch has been called "the YouTube of interactive media." Each day, Scratchers from around the world upload more than 1000 new projects to the site, with source code freely available for sharing and remixing. The collection of projects is wildly diverse: video games, interactive newsletters, science simulations, virtual tours, birthday cards, animated dance contests, interactive tutorials, and many others, all programmed in Scratch.

The core audience on the Scratch website is between the ages of 8 and 16 (with a peak at age 12), though there is a sizeable group of adult participants as well. As Scratchers program and share interactive projects, they learn important mathematical and computational concepts, while also learning to think creatively, reason systematically, and work collaboratively – essential skills for the 21st century. Indeed, our primary goal is not to prepare people for careers as professional programmers, but rather to nurture the development of a new generation of creative, systematic thinkers who are comfortable using programming to express their ideas.

In this article, we discuss the motivations underlying Scratch, the design principles that guided our development of Scratch, and future directions in our efforts to make programming accessible and engaging for everyone. But first, to give a sense of how Scratch is being used, we describe a series of projects from a 13-year-old girl with the Scratch screen name BalaBethany.

Sample Scratcher: BalaBethany

BalaBethany enjoys drawing anime characters. So when she started using Scratch, it was natural for her to program animated stories featuring anime characters. She began sharing her projects on the Scratch website, and other members of the community responded very positively, posting glowing comments under her projects (such as "Awesome!" and "OMG I LUV IT!!!!!!"), along with questions about how she achieved certain visual effects (such as "How do you make a sprite look see-through?"). Encouraged, BalaBethany started to create and share new Scratch projects on a regular basis, like episodes in a TV series.

BalaBethany periodically added new characters to her series. At one point, she got an idea: why not involve the community in the process? She created and uploaded a new Scratch project that announced a "contest," asking other community members to design a sister for one of the characters. The project listed a set of requirements for the new character, including "Must have red or blue hair, please choose" and "Has to have either cat or ram horns, or a combo of both."

The project received more than 100 comments. One comment was from a community member who wanted to enter the contest, but said that she didn't know how to draw anime characters. So BalaBethany produced another Scratch project: a step-by-step tutorial, demonstrating a 13-step process for drawing and coloring an anime character.

Over the course of a year, BalaBethany programmed and shared more than 200 Scratch projects, covering a wide range of different types of projects (stories, contests, tutorials, and others). Her programming and artistic skills progressed over time, and her projects clearly resonated with the Scratch community, receiving more than 12,000 comments.

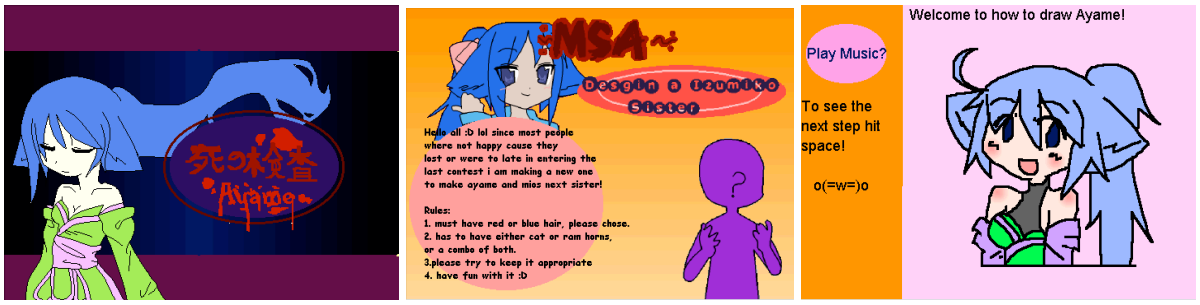

Figure 1: Screen shots from BalaBethany's anime series, contest, and tutorial

Why Programming?

It has become commonplace to refer to young people as "digital natives," because of their apparent fluency with digital technologies[15]. And, indeed, many young people are very comfortable sending text messages, playing online games, and browsing the web. But does that really make them *fluent* with new technologies? Although young people interact with digital media all of the time, few of them can create their own games, animations, or simulations. It's as if they can "read" but not "write."

As we see it, digital fluency requires not just the ability to chat, browse, and interact, but also the ability to design, create, and invent with new media[17], as BalaBethany did in her projects. To do that, you need to learn some type of programming. The ability to program offers many important benefits: it greatly expands the range of what you can create (and how you can express yourself) with the computer, while also expanding the range of what you can learn. In particular, programming supports the development of "computational thinking," helping you learn important problem-solving and design strategies (such as modularization and iterative design) that carry over to non-programming domains[18]. And since programming involves the creation of external representations of your problem-solving processes, programming provides you with opportunities to reflect on your own thinking – and even to think about thinking itself.[2]

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 Logo or Basic. Seymour Papert's book *Mindstorms*[13] 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 several factors:

- Early programming languages were too difficult to use. Many children had difficulty mastering the syntax of programming.

- Programming was often introduced with activities (generating lists of prime numbers, or making simple line drawings) that were not connected to young people's interests or experiences.

- 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.

Papert argued that programming languages should have a *low floor* (easy to get started with) and a *high ceiling* (opportunities for increasingly complex projects over time). In addition, we believe that languages need *wide walls* (supporting many different types of projects, so that people with different interests and learning styles can all become engaged). Satisfying the triplet of low-floor/high-ceiling/wide-walls hasn't been easy.[3]

In recent years, there have been new attempts to introduce programming to children and teens.[6] Some use professional programming languages like Flash/ActionScript; others use

new languages developed specifically for younger programmers, such as Alice[7] and Squeak Etoys[5]. These efforts have inspired and informed our work on Scratch. But we weren't fully satisfied with the existing options. In particular, we felt that it was important to make the floor even lower and the walls even wider – but still supporting the development of computational thinking.

To achieve these goals, we established three core design principles for Scratch: we wanted to make it ***more tinkerable***, ***more meaningful***, and ***more social*** than other programming environments. The next three sections discuss each of these design principles, and how they guided the design of Scratch.

More Tinkerable

Our research group has worked closely with the LEGO Company for many years, helping in the development of LEGO MINDSTORMS and other robotics kits[16]. We have always been intrigued and inspired by the way children play and build with LEGO bricks. Given a box full of LEGO bricks, children will start tinkering. They'll snap together a few bricks, and the emerging structure will give them new ideas. As children play and build with LEGO bricks, plans and goals organically evolve along with the structures.

We wanted the process of programming in Scratch to have a similar feeling. The Scratch grammar is based on a collection of graphical "programming blocks" that children snap together to create programs. As with LEGO bricks, connectors on the blocks suggest how they should be put together. Children can start by tinkering with the blocks, snapping them together in different sequences and combinations to see what happens. There is none of the obscure syntax or punctuation of traditional programming languages. The floor is low and the experience is playful.

Figure 2: Sample Scratch scripts

Scratch blocks are shaped to fit together only in ways that make syntactic sense. Control structures like **`forever`** and **`repeat`** are C-shaped to suggest that blocks should be placed inside – and to indicate scoping. Blocks that output values are shaped according to the types of values they return: ovals for numbers and hexagons for booleans. Conditional blocks (like **`if`** and **`repeat-until`**) have a hexagon-shaped voids, indicating a boolean is required.

The name "Scratch" itself highlights the idea of tinkering. The name comes from the scratching technique used by hip-hop disc jockeys, who tinker with music by spinning vinyl records back and forth with their hands, mixing music clips together in creative ways. In Scratch programming, the activity is similar: mixing together graphics, animations, photos, music, and sound.

Scratch is designed to be highly interactive. Double-click on a stack of blocks and it starts to run immediately. You can even make changes to a stack as it is running, so it is easy to experiment with new ideas incrementally and iteratively. Want to create parallel threads? Simply create multiple stacks of blocks. Our goal is to make parallel execution as intuitive as sequential execution.

Figure 3: Scratch User Interface

The scripting area in the Scratch interface is intended to be used like a physical desktop. You can leave extra blocks or stacks lying around, in case you need them later. The underlying message: It's OK to be a little messy and experimental. Most programming languages (and computer-science courses) privilege top-down planning over bottom-up tinkering. With Scratch, we want tinkerers to feel just as comfortable as planners.

The emphasis on iterative, incremental design is aligned with our own development style in creating Scratch. We selected Squeak as an implementation language since it is well-suited for rapid prototyping and iterative design. Before we launched Scratch, we continually field-tested prototypes in real-world settings, revising over and over based on feedback and suggestions from the field.[4]

More Meaningful

We know that people learn best, and enjoy it most, when they are working on personally-meaningful projects. So in developing Scratch, we put a high priority on:

- ***diversity*** – supporting many different types of projects (stories, games, animations, simulations), so that people with widely varying interests can all work on projects that they care deeply about.
- ***personalization*** – making it easy for people to personalize their Scratch projects by importing photos and music clips, recording voices, creating graphics[14].

These priorities influenced many of our design decisions. We decided to focus on 2D images, rather than 3D, since it is much easier for people to create, import, and personalize 2D artwork. While some people might see the 2D style of Scratch projects as somewhat outdated, Scratch projects exhibit a visual diversity and personalization that is missing from 3D authoring environments.

The value of personalization is captured nicely in this blog post from a computer scientist who introduced Scratch to his two children:

> I have to admit that I initially didn't get why a kids' programming language should be so media centric, but after seeing my kids interact with Scratch it became much more clearer to me. One of the nicest things I saw with Scratch was that it personalized the development experience in new ways by making it easy for my kids to add personalized content and actively participate in the development process. Not only could they develop abstract programs to do mindless things with a cat or a box, etc… but they could add THEIR own pictures and THEIR own voices to the Scratch environment which has given them hours of fun and driven them to learn.

We continue to be amazed by the diversity of projects that appear on the Scratch website. As expected, there are lots of games on the site, ranging from painstakingly recreated versions of favorite video games (such as Donkey Kong) to totally original games. But there are many other genres too. Some Scratch projects document life experiences, such as a family vacation in Florida, while others document imaginary wished-for experiences, such as a trip to meet other Scratchers. Some Scratch projects, such as birthday cards and messages of appreciation, are intended to cultivate relationships. Others are designed to raise awareness on social issues such as global warming or animal abuse. During the 2008 U.S. Presidential election, there was a flurry of projects featuring Barack Obama and John McCain – and then a series of projects by Scratchers self-organizing an election for the not-quite-defined position of "President of Scratch."

Figure 4: Screen shots from sample Scratch projects

Some Scratch projects grow out of school activities. For an earth-science class, a 13-year-old in India created a project in which an animated character travels to the center of the earth, with a voice-over describing the different layers along the way. As part of a social-studies class, a 14-year-old from New Jersey created a simulation of life on the island of Rapa Nui, designed to help others learn about the local culture and economy.

As Scratchers work on personally meaningful projects, we find that they are ready and eager to learn important mathematical and computational concepts associated with their projects. Consider the case of Raul, a 13-year-old boy who was using Scratch to program an interactive game at his after-school center[9]. Raul had created the graphics and basic actions for the game, but he didn't know how to keep score. So when a researcher on our team visited the center, Raul asked him for help. The researcher showed Raul how to create a variable in Scratch, and Raul immediately saw how he could use a variable for keeping score. He began playing with the blocks for incrementing variables, then reached out and shook the researcher's hand, saying "Thank you, thank you, thank you." The researcher wondered: How many 8th grade algebra teachers get thanked by their students for teaching them about variables?

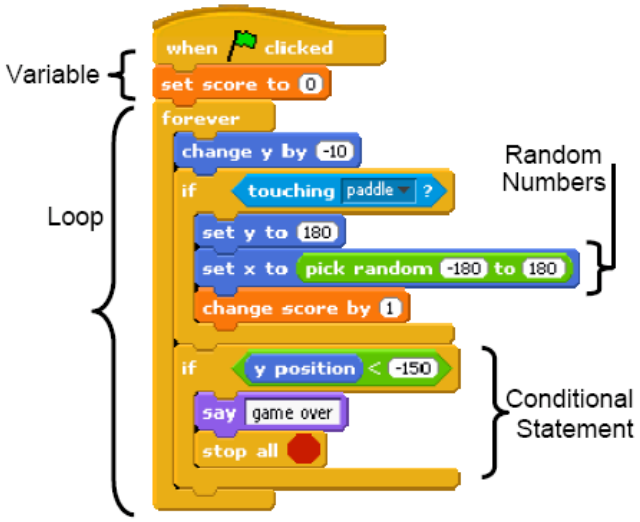

Figure 5: Sample Scratch script (from Pong-like paddle game), highlighting computational and mathematical concepts

More Social

The development of the Scratch programming language has been tightly coupled with the development of the Scratch website.[12] For Scratch to succeed as we hoped, we felt the language needed to be linked to a community, where people could support one another, collaborate with one another, critique one another, and build on one another's work[1].

The concept of sharing is built right into the Scratch user interface, with a prominent Share menu and icon at the top of the screen. Click the Share icon and your project is uploaded to the Scratch website, where it is displayed at the top of the page, along with the Newest Projects. Once a project is on the website, anyone can run the project within a

browser (using a Java-based player), comment on the project, vote for the project (by clicking the "Love it?" button), or download the project to view and revise the scripts. (All projects shared on the website are covered by Creative Commons license.)

In the first 18 months after the Scratch launch, more than 250,000 projects were shared on the Scratch website. For many Scratchers, the opportunity to put their projects in front of a large audience – and receive feedback and advice from other Scratchers – serves as a strong motivation. The large library of projects on the website also serves as an inspiration. By exploring projects on the site, Scratchers can get ideas for new projects and learn new programming techniques. Marvin Minsky once noted that Logo had a great grammar but not much literature.[11] Whereas young writers are often inspired by reading great works of literature, there was no analogous library of great Logo projects to inspire young programmers. The Scratch website is the beginning of a "literature" for Scratch.

Figure 6: Scratch website

The website has served as fertile ground for collaboration. Community members are constantly borrowing, adapting, and building upon one another's ideas, images, and programs. More than 15% of the projects on the website are remixes of other projects on the site. For example, there are dozens of versions of the game Tetris, as Scratchers continue to add new features and try to improve the gameplay. There are also dozens of dress-up doll projects, petitions, and contests, all adapted from previous Scratch projects.

At first, some Scratchers were upset when their projects were remixed, complaining that others were "stealing" their projects. That led to discussions on the website forums about the value of sharing and open-source communities. Our goal is to create a culture in

which Scratchers feel proud, not upset, when their projects are adapted and remixed by others. We have continually added new features to the website to support and encourage this mindset. Now, when someone remixes a project, the website automatically adds a link back to the original project, so that the original author gets credit. Also, each project includes links to its "derivatives" (projects that were remixed from it), and the Top Remixed projects are featured prominently on the homepage of the Scratch website.

Some projects focus on the website itself, providing reviews and analyses of other projects on the site. One early example was called SNN, for Scratch News Network. The project featured the Scratch cat (the default character in Scratch) delivering news about the Scratch community, much like a CNN anchor. At first, we saw it as a "simulated newscast." But then we realized it was a *real* newscast, providing news of interest to a real community – the Scratch online community. The SNN project inspired others, leading to a proliferation of online newsletters, magazines, and TV shows, all programmed in Scratch, reporting on the Scratch community.

Other Scratchers have formed online "companies," working together to create projects that none of the individual members could have produced on their own. One company got its start when a 15-year-old girl from England, with screen name BeeBop, created a project full of animated sprites, and encouraged others to use the sprites in their projects – or to place special requests for custom-made sprites. In short, she was setting up a no-fee consulting business. A 10-year-old girl, with screen name MusicalMoon, liked BeeBop's animations and asked if she'd be willing to create a background for one of her projects. This collaboration gave rise to Mesh Inc., a self-proclaimed "miniature company" to produce "top quality games" in Scratch. A few days later, a 14-year-old boy from New Jersey, screen name Hobbit, discovered the Mesh Inc. gallery and offered his services: "I'm a fairly good programmer, and I could help with de-bugging and stuff." Later, a 11-year-old boy from Ireland, with screen name Marty, was added to the Mesh staff because of his expertise in scrolling backgrounds.

Such collaborations open opportunities for many different types of learning. Here's how a 13-year-old girl, who started a Scratch company called Blue Elk Productions, describes her experience:

> What is fun about Scratch and about organizing a company to write games together is that I've made a lot of friends and learned lots of new things. I've learned a lot about different kinds of programming by looking at other games with interesting effects, downloading them, and looking at and modifying the scripts and sprites. I really like programming! Also, when I started with Scratch I didn't think I was a very good artist. But since then, just by looking at other people's art projects, asking them questions, and practicing drawing using programs like Photoshop and the Scratch paint editor, I've gotten a lot better at art...Another thing I've learned while organizing Blue Elk is how to help keep a group of people motivated and working together…I like Scratch better than blogs or social networking sites like Facebook because we're creating interesting games and projects that are fun to play, watch, and download. I don't like to just talk to other people online, I like to talk about something creative and new.

To encourage international sharing and collaboration, we've placed a high priority on translating Scratch into multiple languages. We created an infrastructure that allows the Scratch programming blocks to be translated into any language with any character set. A global network of volunteers has provided translations for more than 40 languages. So children around the world can now share Scratch projects with one another, each viewing the Scratch programming blocks in their own language.

Future Directions

A growing number of K-12 schools, and even some universities[8], are using Scratch as a first step into programming. A natural question is what comes next – and, indeed, there are ongoing debates in the Scratch discussion forums about what programming language to use after Scratch. We receive many requests to add more advanced features to Scratch (such as object inheritance or recursive list structures), so that Scratch itself could be the "next step."

But we plan to keep our primary focus on lowering the floor and widening the walls, not raising the ceiling. For some Scratchers, especially those wanting to pursue careers in programming or computer science, it is important to move on to other languages. But for many other Scratchers, who see programming as a medium for expression, not a path toward career, Scratch is sufficient for their needs. With Scratch, they can continue to experiment with new forms of self-expression, producing a diverse range of projects, while also deepening their understanding of a core set of computational ideas. A little bit of programming can go a long way.

As we develop future versions of Scratch, our goal is to make Scratch even more tinkerable, meaningful, and social. With our Scratch Sensor Board, people can create Scratch projects that sense and react to events in the physical world. We are also developing a version of Scratch that runs on mobile devices, and a web-based version that enables people to access online data and program online activities.

Probably the biggest challenges for Scratch are not technological but cultural and educational.[10] Scratch has been a success among early adopters, but we need to provide better educational support for it to spread more broadly. We are starting a new online community, called Scratch-Ed, where educators can share their ideas, experiences, and lesson plans for Scratch. More broadly, there needs to be a shift in how people think about programming – and about computers in general. We need to expand the notion of "digital fluency" to include designing and creating, not just browsing and interacting. Only then will initiatives like Scratch have a chance to live up to their full potential.

Acknowledgements

Many people have contributed to the development of Scratch – and even more to the ideas underlying Scratch. We'd like to thank friends and former members of the Lifelong Kindergarten group who have worked on Scratch, especially Tammy Stern, Dave Feinberg, Han Xu, Margarita Dekoli, Leo Burd, Oren Zuckerman, Nick Bushak, and Paula Bonta. We are grateful to Kylie Peppler, Grace Chui, and other members of Yasmin Kafai's research team, who conducted and

participated in field studies in the early development of Scratch. Scratch was deeply influenced and inspired by the work of Seymour Papert and Alan Kay. We appreciate financial support from the National Science Foundation (grant ITR-0325828), Microsoft, Intel Foundation, Nokia, and MIT Media Lab research consortia. For more information, see http://scratch.mit.edu (All children's names in this article are pseudonyms.)

References

1. Bransford, J., Brown, A., and Cocking, R. *How People Learn: Mind, Brain, Experience, and School*. National Academy Press, 2000.
2. diSessa, A. *Changing Minds: Computers, Learning, and Literacy*. MIT Press, 2000.
3. Guzdial, M. Programming environments for novices. In S. Fincher and M. Petre (Eds.), *Computer Science Education Research*, 127-154. Taylor & Francis, 2004.
4. Kafai, Y., Peppler, K., and Chiu, G. High Tech Programmers in Low Income Communities: Seeding Reform in a Community Technology Center. In C. Steinfield, B. Pentland, M. Ackerman, & N. Contractor (Eds.), *Communities and Technologies*, 545-564. Springer, 2007.
5. Kay, A. Squeak Etoys, Children, and Learning. http://www.squeakland.org/resources/articles
6. Kelleher, C., and Pausch, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, 37, 2, 83-137.
7. Kelleher, C., and Pausch, R. Using Storytelling to Motivate Programming. *Communications of the ACM*, 50, 7, 58-64 (July 2007).
8. Malan, D., and Leitner, H. Scratch for budding computer scientists. *Proceedings of SIGCSE conference*, 2007
9. Maloney, J., Peppler, K., Kafai, Y., Resnick, M., and Rusk, N. Programming by choice: Urban youth learning programming with Scratch. *Proceedings of SIGCSE conference*, 2008.
10. Margolis, J. *Stuck in the Shallow End: Education, Race, and Computing*. The MIT Press, 2008.
11. Minsky, M. Introduction to LogoWorks. In C. Solomon, M. Minsky, and B. Harvey (Eds.), *LogoWorks: Challenging Programs in Logo*. McGraw-Hill, 1986
12. Monroy Hernandez, A., and Resnick, M. Empowering Kids to Create and Share Programmable Media. *Interactions*, 15, 2, 50-53 (March-April 2008).
13. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, 1980.
14. Peppler, K., and Kafai, Y. From SuperGoo to Scratch: Exploring Creative Media Production in Informal Learning. *Journal on Learning, Media, and Technology*, 32, 7, 149-166 (2007).
15. Prensky, M. Digital Natives, Digital Immigrants. *On the Horizon*, 9, 5, 1-6 (October 2001).
16. Resnick, M. Behavior Construction Kits. *Communications of the ACM*, 36, 7, 64-71 (July 1993).
17. Resnick, M. Sowing the Seeds for a More Creative Society. *Learning and Leading with Technology* (Dec. 2007), 18-22.
18. Wing, J. Computational Thinking. *Communications of the ACM,* 49, 3, 33-35 (March 2006).