Programming by Choice: Urban Youth Learning Programming with Scratch

John Maloney, Kylie Peppler*, Yasmin B. Kafai*, Mitchel Resnick and Natalie Rusk
MIT Media Laboratory
77 Massachusetts Ave. E15-020
Cambridge, MA 02139
001-617-253-6879
[jmaloney, mres, nrusk]@media.mit.edu

*UCLA Graduate School of Education
2331 Moore Hall
Los Angeles, CA 90095-1521
001-310-206-8150
kpeppler@ucla.edu, kafai@gseis.ucla.edu

ABSTRACT
This paper describes Scratch, a visual, block-based programming language designed to facilitate media manipulation for novice programmers. We report on the Scratch programming experiences of urban youth ages 8-18 at a Computer Clubhouse—an after school center—over an 18-month period. Our analyses of 536 Scratch projects collected during this time documents the learning of key programming concepts even in the absence of instructional interventions or experienced mentors. We discuss the motivations of urban youth who choose to program in Scratch rather than using one of the many other software packages available to them and the implications for introducing programming at after school settings in underserved communities.

Categories and Subject Descriptors
K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms
Design, Human Factors, Languages

Keywords
Novice programming environments, wider-access, Scratch.

1. INTRODUCTION
With the recognized need to broaden participation in computing, a number of different efforts have been discussed in K-12 and college education, such as mentoring, revised curricula, tool development, outreach programs, and programming courses for non-majors. One area that has received surprisingly little attention is the learning of programming in community technology centers that offer free access on a daily basis. More popular but more limited in time have been summer camps and after school programs [1; 15]. In these venues, the learning of programming is by choice meaning that what, when, and for how long programming takes place is at the discretion of the learner rather than part of a required curriculum. While considerable amount of research in computer science education has focused on the classroom or seminar level, few efforts have studied or documented activities where computer-programming opportunities are available outside of the school space for pre-college youth.

SIGCSE'08, March 12–15, 2008, Portland, Oregon, USA.

Summer camps, after-school programs, and community technology centers could play a greater role in offering opportunities for learning computer programming. For one, most schools use technology to teach content and only few offer opportunities to learn programming, especially for low-income students [3]. Another important reason that compels one to consider other places that shape learning is that children only spend 9% of their childhood in school [14]. Finally, out-of-school activities also present opportunities for youth to succeed who may not flourish in traditional school environments.

In this paper, we focus on the use of Scratch, a block-based programming language designed to facilitate media manipulation for novice programmers [11], at a Computer Clubhouse, an urban, after-school technology center. We collected of 536 Scratch programs created by youth at one particular Computer Clubhouse and analyzed their use of programming commands and concepts over time. We also interviewed Clubhouse members about their ideas of programming and perceptions of Scratch. In our discussion, we address what novice programmers can learn in an informal context that does not rely on structured instruction and what motivated youth to choose programming over other available software.

2. SCRATCH
Scratch was created by the Lifelong Kindergarten Group at the MIT Media Laboratory in collaboration with Yasmin Kafai's group at UCLA. Scratch is not the first programming environment and language aimed at novice programmers. Indeed, there is a rich history of different developments comprehensively surveyed by Kelleher and Pausch [7] and Guzdial [4]. Scratch builds on the ideas of Logo [8] but replaces typing code with a drag-and-drop approach inspired by LogoBlocks [2] and EToys [13]. Scratch emphasizes media manipulation and supports programming activities that resonate with the interests of youth, such as creating animated stories, games, and interactive presentations. A Scratch project consists of a fixed *stage* (background) and a number of movable *sprites*. Each object contains its own set of images, sounds, variables, and scripts. This organization enables easy export and exchange of sprites.

Programming is done by dragging command blocks from a palette into the scripting pane and assembling them, like puzzle pieces, to create "stacks" of blocks. An individual block or a stack of blocks can be run by double-clicking on it. Various *hat* blocks can be

placed on top of a stack of blocks to trigger that stack in response to some run-time event, such as program startup, a given key being pressed, or a mouse click on the sprite. Multiple stacks can run at the same time so, without realizing it, most Scratch users make use of multiple threads.

The Scratch screen (see Figure 1) is divided into four areas. On the right is the stage. A button on the bar below the stage allows the stage to be displayed in full screen mode to show off a finished project. Below the stage is an area that shows thumbnails of all sprites in the project. Clicking on one of these thumbnails selects the corresponding sprite. The middle pane allows the user to view and change the scripts, costumes (images), or sounds of the selected sprite. The left-most pane is the palette of command blocks that can be dragged into the scripting area. The palette is divided into eight color-coded categories.

This user interface design grew out of a desire to make the key concepts of Scratch as tangible and manifest as possible. Having the command palette visible at all times invites exploration. A user who notices an interesting command can double-click it right in the palette to see what it does. A user can watch stacks in the scripting area highlight as the action unfolds on the stage. These explorations are supported by having the palette, scripting area, and stage simultaneously visible, providing the user with a process model of how their scripts are interpreted by the computer.

Figure 1: Screenshot of Scratch Interface

The Scratch vocabulary of roughly ninety commands includes commands for relative motion (like the Logo turtle), absolute positioning using Cartesian coordinates, image transformations (rotation, scaling, and effects such as Fisheye), cell animation (switching between images), recorded-sound playback, musical note and drum sounds, and a programmable pen. Since many of these commands take numbers as parameters, the Scratch user has meaningful context for improving their understanding of numbers. For example, using a negative argument with the *move* command makes the sprite move backwards. Arithmetic, comparison and simple Boolean operations are currently supported, and more advanced scientific functions (e.g. *sine*) will be added soon. There are sensing blocks to detect when a sprite is touching the edge, another sprite, or a particular color, as well as sensing blocks that report the mouse location or the up-down state of keyboard keys.

Scratch has a number of control structures, including conditionals (*if*, *if-else*), loops (*repeat*, *forever*, *repeat-until*), and event triggers (*when-clicked*, *when-key-pressed*). Communication is done via named broadcasts. For example, one sprite might broadcast "you won!" causing another sprite to appear on the stage and play a victory song. One broadcast can trigger multiple scripts. A variant of the broadcast command waits for all triggered scripts to complete before going on, thus providing a simple form of synchronization. In addition, Scratch supports two kinds of variables. *Sprite variables* are visible only to the scripts within that sprite, while *global variables* are visible to all objects. Global variables are sometimes used in conjunction with *broadcast* as a way to pass data between sprites.

3. SCRATCH IN THE CLUBHOUSE

We introduced Scratch in January 2005 to a Computer Clubhouse located at a storefront location in South Central Los Angeles. The Clubhouse serves African American and Latino youth ages 8-18 from one of the city's most impoverished areas. Youth become members of the Computer Clubhouse at no cost to them or their families and gather in the after-school hours to engage in a variety of gaming and design activities [12]. These can include playing Microsoft Xbox, downloading images online, recording music in the studio, playing board games, manipulating images in Adobe Photoshop, making roller coaster games in RPG maker or designing 3D backgrounds in Bryce 5.

At the time that Scratch was first introduced, programming activities were not a part of the Clubhouse portfolio of activities [6], despite the wide availability of various types of programming software. Over the course of the first two years of the project, Scratch grew to be the most widely used design software available at the Clubhouse and local programming experts emerged. We did very little to explicitly "teach" programming concepts; rather, youth worked on projects of their own choosing and requested assistance from mentors when needed. Every two or three months, we organized a Scratch-a-thon during which all clubhouse members would work on Scratch for 3-4 hours and share publicly the projects they had created. Clubhouse members used Scratch to implement various media applications ranging from video games to music videos, greetings card and animations [9]. For instance, the dance video "k2b" was created by Kaylee, a thirteen-year-old female software designer, who modeled the piece after a Gwen Stefani music video called "Hollaback Girl" (see Figure 2).

Figure 2: "k2b" Scratch program

Another example is the videogame called "Metal Slug Hell Zone X" created by Jorge (see Figure 3). The middle screen is a screenshot of an avatar while the game is in play mode. On the left is a partial screen shot of the scripts that control one of the avatars. On the right is a partial screen shot of the costumes area,

illustrating a short sequence of still frames used to animate a shooting sequence.

During the first 18 months of the introduction, we collected youths' Scratch projects on a weekly basis in order to track the extent to which programming concepts were taking root in the Clubhouse culture over time. We used three different data sources for our analyses: (1) the exported project summary files, which contained text-based information such as the date, file name, and author of the project as well as information about the number and types of commands that were used and the total number of stacks, sounds, and costumes used in the project; (2) weekly participant field notes that were written by a team of Undergraduate and Graduate field researchers that visited the Computer Clubhouse and supported Scratch activities at the field site; and (3) interviews with Clubhouse members about their impressions of Scratch, what it compared to, and what they knew about programming. It's important to note that the primary role of the Undergraduate and Graduate support team was to model how to learn and they were not computer scientists. The mentors had little or no experience programming and were new to Scratch [5]. In our view, this was empowered youth, allowing them to sometimes switch roles and teach a mentor something new in Scratch.

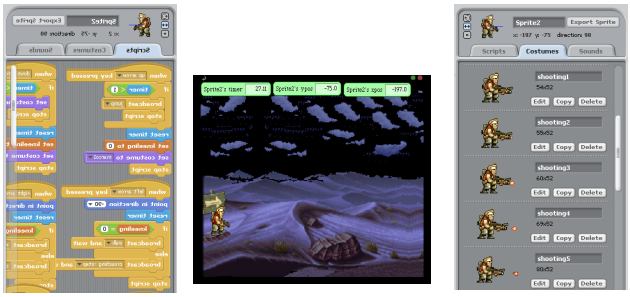

Figure 3: "Metal Slug Hell Zone X" Scratch program.

4. PROGRAMMING CONCEPTS

A total of 536 projects were collected for analysis, which constituted 34% of all the projects created at the Computer Clubhouse during the course of this study. Scratch was more heavily used than any other media-creation tool, including Microsoft Word. Overall, an even mix of over 80 boys and girls used Scratch to create programming projects and most programmers engaged in working on a single project over long periods of time – sometimes over the course of a year. These findings demonstrate that Scratch became a successful part of the local Computer Clubhouse culture. It's also one of the few programming initiatives that successfully engaged both boys and girls – all of them youth of color.

To get an idea of what programming concepts were learned by these youth, we analyzed the use of Scratch commands across the set of projects that we collected. We took the use of certain blocks to indicate that a concept was being used in a given project. For example, Figure 4 shows a script from a simple paddle game in which a ball falls from a random place along the top of the stage and must be caught by a paddle controlled by the mouse. This script uses the concepts of sequential control flow, a loop, conditional statements, variables, and random numbers. The overall game also uses the concepts of user interaction (the paddle tracks the x position of the mouse) and threads (the paddle has its own script that runs in parallel with the ball script).

Of the 536 projects, 111 of them contained no scripts at all. These "pre-scripting" projects illustrate the use of Scratch simply as a media manipulation and composition tool. Beginning Scratch users often spend time importing or drawing images and recording sounds before moving on to scripting. Of the remaining 425 projects, all of them make use of sequential execution (i.e. a stack with more than one block) and most (374 projects, 88%) show the use of threads (i.e. multiple scripts running in parallel). These are core programming concepts that confront every Scratch user when they begin writing scripts. We also looked at a number of other programming concepts: User Interaction (use of keyboard or mouse input), Loops, Conditional Statements, Communication and Synchronization (*broadcast* and *when-receive*), Boolean Logic (*and*, *or*, and *not*), Variables, and Random Numbers. Unlike sequential execution, these concepts are not needed in every project. For example, one can make a simple program to move a sprite using the arrow keys without using a loop or conditional.

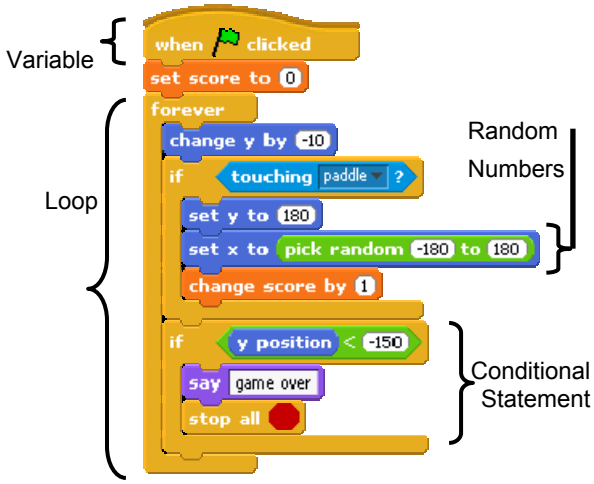

Figure 4: Scratch Script for Ball in a Simple Paddle Game

The goal of this analysis was to study the extent to which youth touched upon these concepts as well as to gauge if the community as a whole increased their knowledge of computer programming over time. Table 1 summarizes the use programming concepts. Given the popularity of games and animation, it is not surprising that projects showing the use of User Interaction and Loops were common. It was a pleasant surprise to find that the Communication and Synchronization commands were also fairly heavily used; inter-object communications is one of the most complex ideas in Scratch, but it answers a critical need when building more complex projects. On the other hand, Boolean operations, variables, and random numbers are concepts that are not easily discovered on one's own. In fact, one user had a desperate need for the variables in his project. When Mitchel Resnick, on a visit to the Clubhouse, showed him how to use variables, he immediately saw how they could be used to solve his problems and thanked Mitchel repeatedly for the advice.

We also examined trends over time. In general, the number of projects produced in the second school year doubled the number of projects produced during same period the first year of the project. (We completed this part of the data collection partway through the second school year.) When we compared the

Table 1. Programming concepts in Scratch projects containing scripts, ordered from most to least heavily used

PROGRAMMING CONCEPT	Number of Projects Containing Concept	Percentage of 425 Scripted Projects
User Interaction	228	53.6%
Loops	220	51.8%
Conditional Statements	111	26.1%
Communications and Synch.	105	24.7%
Boolean Logic	46	10.8%
Variables	41	9.6%
Random Numbers	20	4.7%

percentage of projects containing the various programming concepts over time, we found that five out of the seven concepts that we targeted for our analyses demonstrated significant gains ($p < .05$) during the second school year. Among these were the less obvious concepts of variables, Boolean logic, and random numbers. Chi-Square tests were used to analyze differences in the percentages of projects containing targeted programming concepts from Year 1 to Year 2 (see Figure 5). Overall, four of the seven programming concepts (i.e., Loops, Boolean Logic, Variables, and Random Numbers) demonstrated significant gains in the number of projects utilizing the targeted concepts ($p < .001$). One of the remaining concepts (i.e., Conditional Statements) had marginal gains ($p = .051$) and one concept (i.e., Communication/ Synchronization) demonstrated a significant reduction in the number of projects utilizing this particular concept.

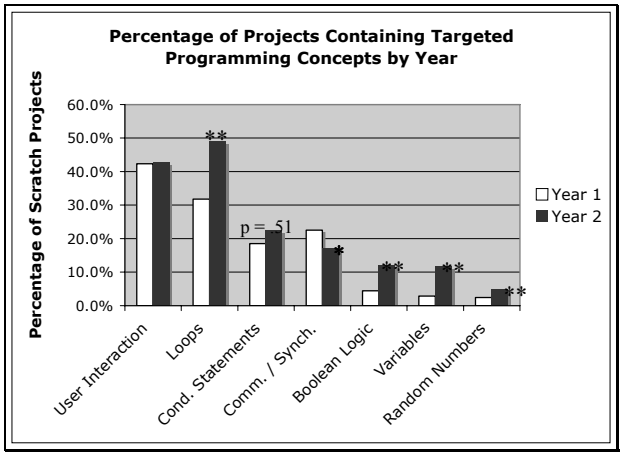

Figure 5. Graph demonstrating the change in the percentage of projects that used various programming concepts over time **p < .001 *p < .05

5. YOUTH IDEAS OF PROGRAMMING

What did the youth in this study have to say about their experiences? We interviewed 30 Clubhouse members and asked them about their ideas of programming and Scratch. When pressed to answer the question, "If Scratch had to be something not on the computer, what would it be?" the most common response was "paper" or "a sketchbook" because Scratch allows you to "do anything that you want with it, just like paper" (n = 8). Others responded in a similar fashion, saying that Scratch was like "everything" because it can be "your own creative world" (n = 6). The remaining responses varied but were along the lines of "something cool," "something fun," or "school" because it "gives you opportunities." Only one youth said that Scratch wouldn't allow him to do what he wanted. He was accustomed to working in Flash and he missed the timeline feature for animation.

We asked a series of open-ended questions to better understand how youth situated Scratch among a number of tools at home, at school, and at the Clubhouse. When asked whether Scratch reminded the youth of anything at school, all of the youth said Scratch was at least like one school subject, and most cited several subjects that they thought connected to their experiences in Scratch. The most frequent response was generally to the arts (n = 20), then to language arts, particularly to reading (n =10), followed by math (n = 8), science (n =5), history or social studies (n =3), and computer class (n =2). When probed further about Scratch's similarities to art, the youth cited drawing or sculpture (n = 11), drama (n = 6), music (n =4), and dance (n = 3). From these responses, we learned that youth felt that Scratch was most similar to schooling activities that support creative, personal expression, such as art and language arts.

Most youth didn't identify scripting in Scratch as a form of programming. In general, when youth were asked, "What is computer programming to you?" they responded: "Computer programming? I do not have a clue [what that is]!" At first we were concerned that youth didn't make the connection between Scratch and programming. But on reflection, *not* seeing Scratch as "programming" may have helped Scratch catch on, allowing youth to see Scratch as being in line with their identities as kids, as something "cool", and as a central part of the Computer Clubhouse culture. After all, the point of engaging youth in computer programming is not to turn them all into hackers or programmers, but because being engaged in the full range of technology fluencies–including programming–is an educational right of the 21st Century. This point becomes even more important when over 90% of the youth that come to the Clubhouse have never been in a computer class during their entire K-12 schooling experience. The Clubhouse then becomes an important space for access to computer programming tools.

6. DISCUSSION

Our findings show a sustained engagement with programming among urban youth at a Computer Clubhouse. We found that, on their own, Clubhouse youth discovered and used commands demonstrating the concepts of user interaction, loops, conditionals, and communication and synchronization. The use of less easily discovered concepts such as variables, Boolean logic, and random numbers was less common but increased over time. These findings are especially surprising given the lack of formal instruction and the fact that the mentors had no prior programming experience.

A few commands, such as *absolute value* and *square root* never appeared in projects. This is not surprising, since the need for such computations is rare in the types of projects created. However, other concepts such as variables and random numbers are very useful, but caught on slowly. We speculate that these are concepts that are not easily discovered without guidance. In some cases, the concept may have been appropriated from one of the sample projects that come bundled with Scratch. But in the case of variables, a concept that also appears in sample projects, it

seems that a timely visit by a knowledgeable mentor was needed before the community began to use the idea.

A more pressing question is, of course, why did Clubhouse youth choose to get involved in Scratch programming given that they had many other software options? The best answer might have been provided by Kelleher and Pausch [7] who noted how systems can make programming more accessible for novices "by simplifying the mechanics of programming, by providing support for learners, and by providing students with motivation to learn to program" (p. 131). We think that Scratch addresses all three of these areas. For one, the design of the Scratch blocks simplifies the mechanics of programming by eliminating syntax errors, providing feedback about placement of command blocks, and giving immediate feedback for experiments.

Furthermore, we think that the social infrastructure of the Computer Clubhouse is important in providing support for novice programmers. While the mentors did not have any prior programming experiences – all of them were liberal arts majors – they were willing to listen and encourage youth in pursuing their programming projects. Often we could observe youth recruiting mentors to be collaborators or sounding boards for their project ideas. At times, we saw clubhouse youth teach mentors a few things they had learned about Scratch. While mentors are often associated with being more knowledgeable than their mentees, here we found a more equitable relationship that turned both mentees and mentors into learners [5]. This need for an audience and resources may also explain the success of the recently opened Scratch website (scratch.mit.edu), which allows programmers to upload their projects and share them with others.

Finally, we think that the multimedia aspect of Scratch facilitated urban youth's engagement in programming. The project archive provided ample evidence that Clubhouse members were savvy about various media genres and interested in not only using them but also producing their own versions. Many Scratch programs started with images pulled from the web and centered on popular characters. In fact, we have evidence from other analyses that Scratch projects focused on generic characters were more often abandoned than those that used popular characters [10]. Youth interest in technology starts with digital media and might thus serve as a more promising pathway into programming. The broad spectrum of media designs – from video games to music videos and greeting cards – is a true indicator of youth's interest in not only being user of digital media (as they do on a regular and personal basis) but in going beyond mere consumption to become content creators themselves, a role often denied to urban youth.

7. ACKNOWLEDGMENTS

The work reported in this paper was supported by a grant from the National Science Foundation (NSF-0325828) to Mitchel Resnick and Yasmin Kafai and by a dissertation fellowship from the Spencer Foundation to Kylie Peppler. The views expressed are those of the authors and do not represent the views of the supporting funding agencies or universities. We wish to thank Zrinka Bilusic for her preparation and initial analysis of the Scratch archive.

8. REFERENCES

[1] Adams, J. C. (2007). Alice, middle schoolers & the imaginary worlds camps. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (pp. 307-311). New York, NY: ACM Press.

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

[3] Goode, J., Estrella, R., & Margolis, J. (2006). Lost in translation: Gender and high school computer science. In J. M. Cohoon & W. Aspray (Eds.) *Women in IT: Reasons on the Underrepresentation* (pp. 89-114). Cambridge, MA: The MIT Press.

[4] Guzdial, M. (2004). Programming environments for novices. In S. Fincher and M. Petre (Eds.), *Computer Science Education Research* (pp. 127-154). Lisse, The Netherlands: Taylor & Francis.

[5] Kafai, Y. B., Desai, S., Peppler, K., Chiu, G. & Moya, J. (in press). Mentoring Partnerships in a Community Technology Center: A Constructionist Approach for Fostering Equitable Service Learning. *Mentoring & Tutoring*.

[6] Kafai, Y., Peppler, K., & Chiu, G. (2007). High Tech Programmers in Low Income Communities: Seeding Reform in a Community Technology Center. In C. Steinfield, B. Pentland, M. Ackerman, &. N. Contractor (Eds.), *Proceedings of Communities and Technologies 2007* (pp. 545-564). New York: Springer.

[7] Kelleher, C. & Pausch, R. (2005). Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys, 37*(2), 88-137.

[8] Papert, S. (1980). *Mindstorms*. New York: Basic Books.

[9] Peppler, K. & Kafai, Y. B. (2007). From SuperGoo to Scratch: exploring creative digital media production in informal learning. *Learning, Media, and Technology*, 32(2), pp. 149–166.

[10] Peppler, K. & Kafai, Y. B. (under review). Creative Bytes: The Technical, Creative, and Critical Practices of Media Arts Production. *Journal of the Learning Sciences.*

[11] Resnick, M., Kafai, Y., & Maeda, J. (2003). ITR: A Networked, Media-Rich Programming Environment to Enhance Technological Fluency at After-School Centers. Proposal [funded] to the National Science Foundation, Washington, DC.

[12] Steinmetz, J. (2001). Computers and Squeak as Environments for Learning. In Rose, K. and Guzdial, M. (eds.), Squeak: Open Personal Computing and Multimedia, pp. 453-482. Prentice Hall: New York.

[13] Resnick, M., Rusk, N., & Cooke, S. (1998). Computer Clubhouse: Technological fluency in the inner city. In D. Schon, B. Sanyal and W. Mitchell (Eds.), High technology and low-income communities. Cambridge, MA: MIT Press.

[14] Sosniak, L. (2001). The 9% Challenge: Education in School and Society. Teachers College Record, 103.

[15] Werner, L. L., Campe, S., and Denner, J. (2005). Middle school girls + games programming = information technology fluency. *Proceedings of the 6th Conference on Information Technology Education SIGITE '05* (pp. 301-305). New York, NY: ACM Press.