Floors and Flexibility: Designing a Programming Environment for 4th-6th Grade Classrooms

Charlotte Hill†, Hilary A. Dwyer‡, Tim Martinez†, Danielle Harlow‡, Diana Franklin†

{charlottehill, franklin}@cs.ucsb.edu, {hdwyer, dharlow}@education.ucsb.edu, tmartinez@umail.ucsb.edu

†Computer Science Department
UC Santa Barbara

‡Gevirtz Graduate School of Education
UC Santa Barbara

ABSTRACT

The recent renaissance in early computer science education has provided K-12 teachers with multiple options for introducing children to computer science. However, tools for teaching programming for children with wide-scale adoption have been targeted mostly at pre-readers or middle school and higher grade-levels. This leaves a gap for 4th – 6th grade students, who differ developmentally from older and younger students.

In this paper, we investigate block-based programming languages targeted at elementary and middle school students and demonstrate a gap in existing programming languages appropriate for 4th – 6th grade classrooms. We analyze the benefits of Scratch, ScratchJr, and Blockly for students and curriculum developers. We describe the design principles we created based on our experiences using block-based programming in 4th – 6th grade classrooms, and introduce LaPlaya, a language and development environment designed specifically for children in the gap between grades K-3 and middle school students.

Categories and Subject Descriptors

D.1.7 [**Programming Techniques**]: Visual Programming; K.3.2 [**Computer and Information Science Education**]: Computer Science Education.

Keywords

Computer science education, elementary school, middle school, graphical programming. novice programming environments

1. INTRODUCTION

In an effort to engage young children in computer science, computer scientists have developed a variety of educational programming platforms, activities [23, 9, 17, 4] and outreach programs [3, 2, 12]. In the past few years, momentum has increased for elementary schools to teach computational thinking in their classrooms. Eighth graders' reported interest in pursuing a career in science and engineering areas is a strong predictor of whether or not they will later pursue a science career [28]. Further, after-school opportunities or summer camps where middle and elementary school students are likely to be introduced to computer science are less available to students from impoverished areas [6]. Adding computational thinking to earlier grade levels and integrating the subject into K-8 schooling will provide students from more backgrounds exposure to computer science at an age when they prepare to make decisions about their future. Moreover, these efforts may increase diversity in computer science fields, combat the dire shortage of computing majors, and create a higher level of the computing literacy that will be necessary for the innovators of the next century.

In order to gain deeper insight into how 4th – 6th grade students (children ages 9-12) learn computer science, we modified an existing middle-school curriculum from the Animal Tlatoque summer camp to be appropriate for 4th grade [12]. We provided programming activities in a variant of Scratch [23] with starting files that the students manipulated and modified. During the 2013-14 academic year, we revised and refined these activities based on feedback from the classrooms about what students struggled with and excelled in.

Although elementary school students are capable of programming [21, 28], while developing curricula for 4th – 6th grade, we found a gap in the programming languages available for children. Many popular block-based languages are targeted either toward pre-readers or children with math and language skills above 4th – 6th grade. Moreover, these languages are embedded in programming environments with interface features not developmentally appropriate for this age group; they contain features that are too complex, making the floor for entry too high for these students, or they do not provide students appropriate control over their projects. Additionally, existing block-based languages either require curriculum developers to create projects that fit the constraints of the environment, or to have the programming background needed to customize the environment for the project. New tools that are sufficiently flexible and have an entry floor level appropriate level for upper elementary school students are needed to make computer science successful in a regular classroom.

Here we present our design goals for a block-based language targeted towards 4th – 6th grade. Our design goals are influenced by the challenges that students faced during the pilot of our curriculum when using a programming environment not developmentally appropriate for their age. We look at Scratch, ScratchJr, and Blockly, and assess their appropriateness for curriculum development at this age group [23, 13, 10]. Finally, we introduce LaPlaya, the language we designed to fill the gap in development environments for upper elementary school classrooms.

This paper is organized as follows. We provide background on the elementary school classroom setting and a brief summary of related work in Section 2. In Section 3, we look at existing block-based languages (Scratch, ScratchJr, and Blockly). In Section 4

SIGCSE'15, March 4–7, 2015, Kansas City, MO, USA.

http://dx.doi.org/10.1145/2676723.2677275

we detail the design principles we used to create a new programming environment, LaPlaya. Finally, Section 5 contains our plans for future work.

2. BACKGROUND & RELATED WORK

Students' academic knowledge and skills increase rapidly as they progress throughout elementary school and into middle school. A development environment for this age group must be flexible and expand as they learn, or the students will outgrow it. Students in upper elementary school differ in important ways from their younger counterparts. In second and third grade, "the mechanical demands of learning to read are so taxing at this point that children have few resources left over to process the content. In fourth through eighth grade, children become increasingly able to obtain new information from print" (p. 333) [26]. In fact, upper elementary school is an important transitional time for children's development. These students are developing linguistic, kinesthetic, and cognitive skills necessary to successfully interact with computers. Further, these students are learning key concepts from other fields, like math and science, needed to successfully program. As examples, key math concepts such as division, negative numbers, and percentages are taught during upper elementary school.

Fourth graders today are commonly described as "digital natives" because they have never known a world without computers or the Internet [5]. Although access to technology varies, particularly depending on students' socioeconomic levels, today's 4th graders have some familiarity with computers and ideas about how to interact with them. Students enter the computer lab already familiar with computers, but as users rather than developers. Kolikant characterizes computer science classes as encounters between intertwined cultures. Students are members of the culture of school, and are already familiar with computers as part of the "user" culture [5], but they are newcomers to the computing culture.

Previous studies have looked at the development environment's influence on students' roles as users or developers. In the creation of ScratchJr, a visual language based on Scratch, the developers removed "instant gratification" buttons, or buttons that produce an immediate effect on the state of the program, so children would spend less time playing with the software's features and more time creating [18]. However, Kazakoff found that kindergarteners using ScratchJr in their classroom still spent a significant time using the environment's paint editor instead of programming [18]. Kerr provided pre-built scenes to students for creating digital stories and found that they used more constructs and methods, and spent twice as long modifying their programs as students who created all the scenes themselves [19]. The development environment itself plays a crucial role in guiding students to be developers rather than users.

Researchers in human computer interaction have identified multiple points to consider when designing software for children, as children interact with computers differently than adults. Bruckman, Bandlow, Dimond, and Forte noted that software designers must attend to the physical and cognitive differences of children such as their limited skills with typing, reading, and manipulating keyboards and mice, as well as their more limited linguistic experiences which prevent them from understanding metaphors common among adults [7]. For example, program designers must attend to children's fine motor control. Compared to adults, children struggle holding down the mouse for extended periods or even double-clicking; therefore, young children tend to perform point-and-click movements more quickly and accurately than drag-and-drop commands [15, 27]. As another example, young children can become easily distracted by complex interfaces, and Halgren et al chose to redesign interface by hiding advanced tools, so children (ages 5-14) would not stumble upon them and get lost in their functionality [16]. These findings influenced our development of LaPlaya, which we describe in Section 5.

Studying children's learning of computer science is additionally complex because the programming environment directly impacts available opportunities for learning. Several programming languages developed for children and novice programmers such as Scratch, Alice, and AgentSheets use graphical programming environments and limit the commands available for students to use [23, 8, 25]. Compared to traditional programming environments, graphical programming lowers the cognitive barriers to programming by removing obstacles such as remembering specific vocabulary and formatting rules of computer languages. Instead of writing out programming commands, users snap commands together to form scripts and see the output of their programs immediately, thereby circumventing most required syntax or specific vocabulary. As a result, these languages offer benefits for novice programmers of all ages. Malan and Leitner advocated using Scratch even at the undergraduate level, as a gateway language to languages like Java [22]. Further, block based languages like Scratch can improve students' attitudes and increase confidence in programming [20].

3. POPULAR BLOCK-BASED ENVIRONMENTS

Block based environments have gained popularity with novice programmers because they reduce the need for typing and get rid of many syntax errors. Unlike textual programming languages, block based languages are closely tied to the development environment in which they are embedded. Many development environments are inspired by Scratch, a block-based language designed for ages 8-16 [23]. Scratch provides blocks of programming commands that users attach together like puzzle pieces to create scripts that control the actions of the sprites, 2D characters on the stage. In this section, we present our findings from using Scratch in 4th – 6th grade classrooms and then look at ScratchJr and Blockly, two other block-based environments, and their appropriateness for this age group [13, 10]. Example scripts from Scratch, ScratchJr, Blockly, and LaPlaya are shown in Figure 1.

3.1 Methods

Our results on Scratch are informed by the pilot of a 4th grade computational thinking curriculum. This curriculum consists of three types of activities: short, pre-populated projects that students can finish within a lab period; off computer activities in the classroom that tie computational thinking concepts back to every day life; and an open-ended design-thinking digital story project. This curriculum was designed specifically for Scratch [12] and introduced multiple concepts necessary for digital storytelling: sequential execution, event-based programming, initialization, message passing, costume changes, and scene changes. The initial version used the original Scratch language in a slightly modified environment; blocks that had not been introduced were hidden in order to support the short, lab-length conceptual learning tasks.

We piloted the curriculum in fifteen 4th – 6th grade classrooms at five schools across California. We refer to these schools as A, B,

C, D and E, with A being the first school trial and E being the last. In schools B and E, we collected only student projects. In schools A, C, and D, we observed instruction and interviewed students. The schools had varying numbers of classrooms, grades participating, start dates, and order of projects.

3.2 Findings

We originally intended our curriculum to use Scratch, but when we piloted it in 4th grade classrooms, we found that students struggled with some math concepts and parts of the interface. For this analysis, we focus on schools A and B, the first to use our pilot materials. For the schools with later start dates (C-E), we used a variant of Scratch and changed the interface to address difficulties students had at schools A and B.

3.2.1 The Interface: Distractions and Difficulties

In our pilot classes, we found aspects of the development environment distracted or allowed students to delete parts of the pre-populated projects without clear ways of recovering lost elements. Some students deleted the scripts or sprites that the file started with, making it difficult or even impossible in some cases to add these back later. In schools A and B (142 students, 516 projects), we found that students deleted provided sprites in at least 4.5% of projects, and deleted provided scripts in at least 9.6% of projects. These numbers might be lower than the actual number of projects where sprites or scripts were deleted. We developed these statistics from final versions of assignments that students submitted, but when students realized they had deleted an element of the project they needed, they frequently restarted with the original version of the provided file.

Additionally, students of this age group are likely to be distracted by "instant gratification" buttons [18]. For example, a "surprise sprite" button adds a new random sprite to the stage every time you click on it. These buttons can be distracting for younger age groups. As an example, one student in our study added 34 sprites to one project. Further, in schools A and B, students added unnecessary sprites in 10.1% of projects. Although it is important to allow students to explore and find different ways of solving the problem, "instant gratification" buttons can switch from being vehicles for exploration to distractions that spread through the computer lab as students observe their peers' computers. Providing the ability to limit such features is important when designing an interface for this age group.

3.2.2 The Language: Math Content

Scripts can move sprites across the screen and change the way they look, from their size to their color. However, many of the blocks needed to change the states of sprites require math concepts above the 4th grade level. Although some of these blocks are appropriate for older groups, Scratch does not provide ways to hide specific blocks when working with younger age groups. In this section, we look at the math concepts used in the Scratch language and when they are introduced in the Common Core Math Standards [24].

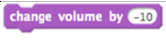
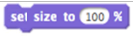
Common Core Concept	***Grade***	***Example Scratch Block***
Cartesian coordinates	5	
Negative numbers	6	
Percentages	6	
Decimal numbers	4	

Table 1. Math Concepts in Scratch and the Common Core

Initializing position and changing values (size, volume, coordinates, variables, time) require mathematical concepts that students have not learned by the start of 4th grade (e,g, x,y coordinates, negative numbers, decimals, and percentages), making the entry floor too high for 4th grade students, our target audience. Table 1 shows each concept, the grade in which this concept appears in the Common Core Math Standards, and an example block that requires this concept [24].

Cartesian Coordinates: Setting a sprite to an absolute location is done with a *go to (sprite or mouse pointer)* block or a *go to x: y: (x and y coordinates)* block. The latter requires an understanding of the Cartesian coordinate system, which is not introduced until 5th grade. Students can use an alternate approach of placing the sprite where they want it to go before selecting the *go to* block, as Scratch auto-populates the proper x and y values. The sequence of moves to memorize would require more repetition than we wanted for our projects, and we do not encourage procedures based on memorizing rather than understanding.

Negative numbers: Negative numbers are used in most motion blocks, and embed in blocks that change the appearance of the sprite (such as the size or color). Scratch's Cartesian plane places the origin, (0,0), in the center of the stage, which means that anything out of the top-right quadrant requires a negative x or y value. Negative numbers are also used by *change (something) by X* blocks to reduce the size, volume, x or y coordinate, and variable value. Negative numbers are not in the Common Core Math Standards until 6th grade, so many students in upper elementary school will not know how to make the value smaller.

Percentages: Size and volume are set with percentages, which are not introduced until 6th grade. Not only must students understand percentage parts of a whole, they also need to understand what 100% means for that variable; for example, the size percentages are of the original picture size, which the students are unlikely to know.

3.3 Other block-based environments

In this section, we look at ScratchJr and Blockly for their appropriateness on 4th – 6th grade classrooms and curriculum. ScratchJr, recently released for the iPad, is aimed toward ages 5-7 [13]. Blockly is used by Code.org in its curriculum and the Hour of Code, a programming initiative that has been massively successful in encouraging students and teachers to try programming [10].

3.3.1 ScratchJr

ScratchJr fixes many of the math and interface issues present when using Scratch with younger age groups. Most the "instant gratification" buttons have been removed to encourage students to focus on programming. The blocks in ScratchJr require less math knowledge, and an optional grid overlay on the stage adds extra support for using x,y coordinates.

Because it is aimed at 5-7 year olds, ScratchJr's ceiling is not as high as is desired for 4th – 6th grades. The programming language uses symbols instead of words and has a smaller block selection than Scratch.

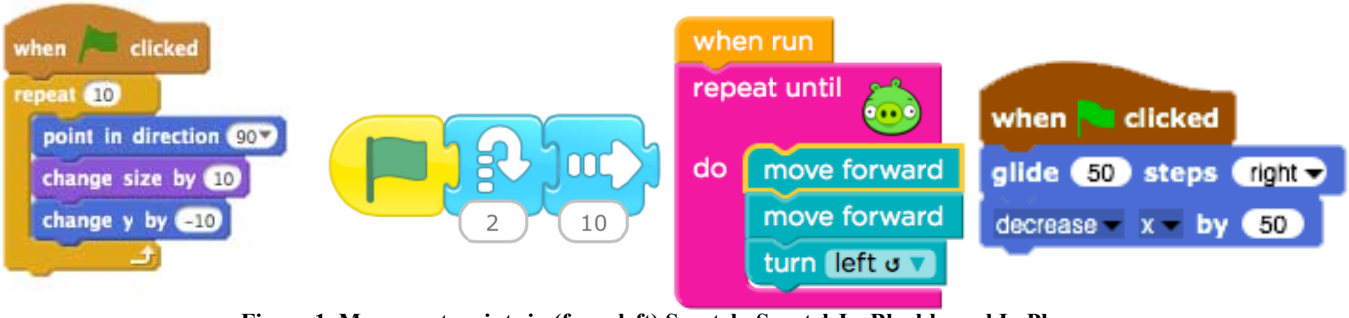


Figure 1. Movement scripts in (from left) Scratch, ScratchJr, Blockly and LaPlaya

It does not have variables, lists, or many of the control structures present in Scratch. Additionally, ScratchJr is more structured than Scratch; users pick scenes for their sprites to act in. ScratchJr lacks the complexity needed to appropriately challenge 4^{th} – 6^{th} graders.

Like Scratch, ScratchJr works best for open-ended projects, but teachers can use pre-populated files for assignments. The danger is that students can see and modify any given sprites and scripts so they may delete sprites and find they need them later on.

3.3.2 Blockly

Blockly is the block-based language used in the popular Hour of Code. Unlike Scratch and ScratchJr, users cannot simply open up Blockly and begin coding; each project is created in Javascript, limiting users to specific projects designed by developers. Projects in Blockly are similar to Scratch and ScratchJr; users create scripts out of blocks to control characters.

Developers create each project in Javascript, which allows them to create unique blocks and interface elements for each project. This added control for developers means that Blockly can be used for any age group, but curriculum developers need to know Javascript to create projects. Additionally, students cannot engage in open-ended, exploratory projects like in Scratch or ScratchJr. Although Blockly can be used to create scaffolded projects for 4^{th} – 6^{th} grade, it lacks the open-ended options present in Scratch and ScratchJr and it has a high overhead for project development. It is unlikely that classroom elementary school teachers will have the background necessary to develop projects using Blockly.

3.4 Bridging the gap with lower floors and greater flexibility

When creating our curriculum, we wanted a language that combined aspects of Scratch, ScratchJr, and Blockly. Scratch gives students low floors, wide walls, and high ceilings, but in some cases (such as the math) the floors are not low enough for our target age group. ScratchJr provides a lower floor and ceiling, but they are too low to be challenging for upper elementary school students. Although Blockly provides more content flexibility, it has too high an overhead for project creation and lacks the open-ended, "playground" environment option present in Scratch and ScratchJr. We wanted a combination of Scratch's "playground" programming environment and low entry point for curriculum developers, ScratchJr's attention to developmental requirements of the target age group, and Blockly's flexibility. We used our experiences with these languages and the feedback from our pilot classrooms to construct a set of design principals that guided our development of a new programming environment, LaPlaya.

4. LAPLAYA

LaPlaya is a block-based language based on Snap!, a development environment inspired by Scratch for supporting higher grade levels [14]. We created a set of design principles for LaPlaya based on existing block-based environments and our experiences in the pilot classrooms. A beta version of LaPlaya was used in some of our pilot classes in the 2013-14 school year, and we plan to use an updated version in the 2014-15 school year.

Our design principles, shown in Table 2, were informed by several criteria. First, we wanted to support different kinds of curricula for 4^{th} – 6^{th} grade, rather than privilege either open-ended or scaffolded projects over each other. Our second criterion was that our interface could support use in formal learning environments with elementary teachers with or without a background in computer science. Practical challenges that teachers face in a traditional classroom create different constraints and opportunities than informal learning environments. Our third criterion was that we wanted the designer interface to support creating student projects and tasks to be accessible to curriculum developers without extensive programming backgrounds. The design principles are informed by existing block-based languages and our own findings during the pilot implementation of our curriculum and while running computer science summer camps for 6^{th} – 8^{th} grade students for three summers [13].

4.1 Support multiple types of tasks.

Our goal was to create an environment that did not limit curriculum designers when developing assignments. This led to the first design principal: support multiple types of tasks. We wanted to support a range of curricular types, from purely open-ended projects that students design and create over long periods of time, to scaffolded curricula with targeted projects of pre-populated sprites and pre-determined goals completed in as little as 10 minutes. The programming environment should offer a wide range of blocks and functionality to support open-ended projects; and developers may want to constrain available blocks and functions to only those necessary to complete the task.

The LaPlaya environment allows developers to create prepopulated projects with options for the scripts and sprites that will be visible to students. Developers can hide sprites or scripts that work in the background; for example, a project with a distinct end goal may include an analysis script that says "Great job!" when the student completes the project. Scripts can be shown as inert examples that do not run but show students how to create the script. Developers can lock scripts or sprites available to students, but that students do not need to manipulate. For example, in Figure 2, students program the car sprite to move to the different cities. Curriculum developers already programmed the other sprites so they are locked. Some

LaPlaya blocks and block categories, such as "Sounds", are hidden since they are not needed for this project. Alternatively, developers may also choose to leave the entire environment visible to students for open-ended projects.

4.2 Require only grade- and age-appropriate content.

The computer science content in the environment should not rely on non-computer science content above grade level. Programming requires both math and literacy skills as programmers must be able to read and understand commands. Some commands require mathematical understandings like addition, which 4th graders do understand, and percentages, which they may not. While we contend that the interface and programming language should support the development of new math and literacy skills, we do not want math and literacy requirements to preclude students from coding. Nor do we want to require teachers to teach content before they would do so in their normal curricular plans.

In LaPlaya, the origin, (0,0), is in the lower left-hand corner of the stage, removing the need for negative numbers in the *go to x: y: (x and y coordinates)* block. All *change (state) by X* blocks were replaced by *increase/decrease (state) by X* blocks so students would not need negative numbers to decrease the value. Finally, LaPlaya has *set size to X* blocks with absolute amounts (small, medium, large) that correspond to absolute sizes on the screen rather than percentages relative to the original picture size. Removing the need for the Cartesian coordinate system is much more challenging and is the subject of ongoing research.

4.3 Include an age-appropriate interface.

The development environment should encourage novice programmers to write code. Those comfortable in the role of "user" might be inclined to avoid coding in favor of activities they are more familiar with, such as manipulating settings or non-programming aspects of the environment, like an image editor. Students today enter the computer lab already familiar with computers as computer users. Using the computer as a developer is a new experience for these students. Kolikant argues that computer science education is a cultural encounter where students have multiple viewpoints; they are newcomers to computer science, students in a classroom, and also computer users [5]. A development environment that introduces students to computer science should encourage students to be developers creating their own programs for an audience rather than users playing with unrelated features of the interface (such as a paint editor).

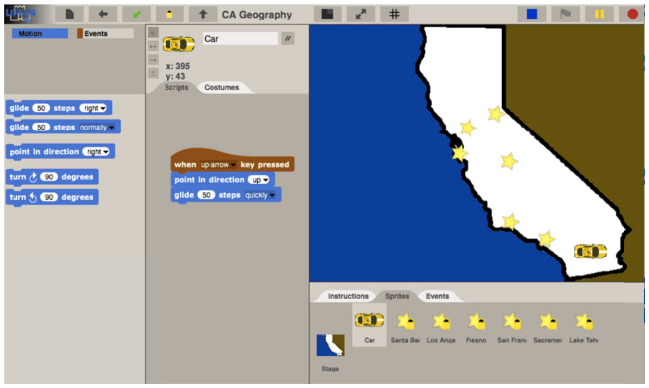


Figure 2. A KELP-CS project in LaPlaya, as seen in student mode

Table 2. Comparison of Scratch, ScratchJr, Blockly and LaPlaya for 4th – 6th grade

Design Principle	*Scratch*	*ScratchJr*	*Blockly*	*LaPlaya*
Support multiple types of tasks				✓
Require only grade- and age-appropriate content			✓	✓
Include an age-appropriate interface			✓	✓
Support project developers	✓	✓		✓

As mentioned earlier, in our pilot classes, we found that "instant gratification" buttons distracted students and led to unrecoverable errors because it was easy to delete sprites and scripts in the pre-populated projects, but hard to get them back. In LaPlaya, developers can hide aspects of the interface that might be distracting for students, such as the "add sprite" button. To encourage students to experiment with the programming blocks (instead of staying in the more familiar role of "user" and playing with non-programming areas), curriculum developers could hide these "non-programming" elements in early projects.

4.4 Support project developers.

Finally, creating a starting file for a project should not require a degree in computer science. Although most computer science teachers at the elementary school level may be more comfortable using existing projects, we believe in lowering the threshold for creating content so someone without a strong computer science background could create new material. In our scenario, any person with facility in Scratch could make a new project in less than a day. As teachers become more comfortable with the environment, they may want to adjust existing material or map projects to their other classroom activities. As such, we contend that the project creation environment (developer mode) should be similar to the environment used by students to finish the projects.

LaPlaya has two modes for viewing and editing a project: a developer mode for project developers and teachers, and a student mode for teachers and students. The developer mode allows developers to manage available content in a project when it is opened in the student version. Developers create projects using the same blocks that could be used in student mode, so creating starting files for assignments does not require more advanced computer science knowledge.

4.5 Summary

Scratch, ScratchJr, and Blockly were not designed to be consistent with our stated design principals, so identifying inconsistencies between the languages and our design principles is not a criticism of these languages, only an indication that they were not ideal for the learning and development context that we are working with.

We found the Scratch interface features and math content were too advanced for 4th – 6th grade. ScratchJr remedied many of these issues but is designed for a younger age group, making it also not developmentally appropriate for 4th – 6th grade. Scratch and ScratchJr can be used with open-ended or scaffolded projects, but their interface designs are problematic when working with pre-populated projects. Blockly provides developers greater control

over block options when creating projects and can be used to make grade level-appropriate assignments for 4^{th} – 6^{th} grade. However, experienced programmers must design these projects. Further, Blockly does not have an option for a "playground" environment where students can create their own projects without a starting file. In developing LaPlaya we built on the strengths of each of these interfaces to create a programming environment tailored to 4^{th} – 6^{th} grade students in a classroom setting.

5. FUTURE WORK

We plan to continue studying students' experiences learning computational thinking and how development environments shape their learning. In particular, we will look at the word choice of blocks in visual programming languages, and whether younger students benefit from a language with more blocks that produce visual effects.

In addition, we plan to extend our work to looking at teachers' learning. In our pilot, we found that many teachers did not have prior experience teaching computational thinking. We plan to research how teachers learn computational thinking, and map the terrain of teacher education in science and technology.

6. ACKNOWLEDGMENTS

This work is supported by the National Science Foundation BPC Award CNS-0940491 and CE21 Award CNS-1240985.

7. REFERENCES

[1] Achieve, I. The Next Generation Science Standards. The National Academies Press, 2013.

[2] Alliance, S. The stars alliance: A southeastern partnership for diverse participation in computing. NSF STARS Alliance Proposal. http://www.itstars.org/.

[3] Arroyo, I. et al. Effects of web-based tutoring software on students' math achievement. In AERA, 2004.

[4] Bell, T., Witten, I. H. & Fellows, M. Computer Science Unplugged. 2006.

[5] Ben-David Kolikant, Y. (Some) Grand Challenges of Computer Science Education in the Digital Age: A Socio-Cultural Perspective. In WiPSCE, 2012.

[6] Bradley, R. H., & Corwyn, R. F. (2002, February). Socioeconomic status and child development. Annual Review of Psychology, 53, 371–399.

[7] Bruckman, A., Bandlow, A., & Forte, A. 2012. HCI for kids. In J. A. Jacko (Ed.) *The human-computer interaction handbook: fundamentals, evolving technologies, and emerging applications* (841 – 862). Boca Raton, FL: Taylor & Francis Group, LLC.

[8] Cooper, S., Dann, W., and Pausch, R. 2000. Alice: a 3-D tool for introductory programming concepts. *J. Comput. Sci. Coll.* 15, 5 (April 2000), 107-116.

[9] Dann, W., Cooper, S., & Pausch, R. Making the connection: programming with animated small world. ITiCSE, 2000.

[10] Empson, R. 2 Weeks and 600+ Lines of Code Later, 20M Students Have Learned An "Hour Of Code", TechCrunch. http://techcrunch.com/2013/12/26/code-org-2-weeks-and-600m-lines-of-code-later-20m-students-have-learned-an-hour-of-code/

[11] Flannery, L. P., et al. Designing ScratchJr: Support for Early Childhood Learning Through Computer Programming. In IDC '13.

[12] Franklin, D., Conrad, P., Aldana, G., et. al. Animal Tlatoque: Attracting Middle-School Students to Computing through Culturally-Relevant Themes. In SIGCSE '11.

[13] Franklin, D., Conrad, P., Boe, B., Nilsen, K., Hill, C., Len, M., Dreschler, G., Aldana, G., et al. Assessment of Computer Science Learning in a Scratch-Based Outreach Program. In SIGCSE '13.

[14] Garcia, D., Segars, L. & Paley, J. 2012. Snap! (build your own blocks): tutorial presentation. *J. Comput. Sci. Coll.* 27, 4 (April 2012), 120-121.

[15] Gelderblom, H., & Kotze, P. 2009. Ten design lessons from literature on child development and children's use of technology. In IDC 2009.

[16] Halgren, S., et al. 1995. Amazing Animation™: Movie making for kids design briefing. In SIGCHI '95.

[17] Hood, C. S. & Hood, D. J. Teaching programming and language concepts using legos. In ITiCSE, June 2005.

[18] Kazakoff, E. R. Cats in Space, Pigs that Race: Does self-regulation play a role when kindergartners learn to code? (Unpublished doctoral dissertation). Tufts University, Massachusetts, 2014.

[19] Kerr, J., Kelleher, C., Ellis, R. & Chou, M. 2013. Setting the scene: scaffolding stories to benefit middle school students learning to program. In *IEEE VL/HCC*, 95-98.

[20] Lewis, C. M. 2010. How programming environment shapes perception, learning and goals: Logo vs. scratch. In SIGCSE '10.

[21] Lewis, C. M. 2011. Is pair programming more effective than other forms of collaboration for young students?, Computer Science Education, 22, June 2011, DOI=10.1080/08993408.2011.579805

[22] Malan, D. J. & Leitner, H. Scratch for Budding Computer Scientists. In SIGCSE'07.

[23] Maloney, J., et al. The scratch programming language and environment. Trans. Comput. Educ., 10(4): 1–15, Nov. 2010.

[24] National Governors Association Center for Best Practices, Council of Chief State School Officers. Common Core State Standards (Math), National Governors Association Center for Best Practices, Council of Chief State School Officers, Washington D.C., 2010.

[25] Repenning, A. 1993. Agentsheets: a tool for building domain-oriented visual programming environments. In CHI '93.

[26] Santrock, J. W. *Child Development.* McGraw Hill, Boston, 2004.

[27] Strommen, E. 1994. Children's use of mouse-based interfaces to control virtual travel. In CHI '94.

[28] Tai, R., Liu, C., Maltese, A., & Fan, X. 2006. Planning early for careers in science, Science, 312, 5777, 1143–11.44.