Bringing Computer Science into Elementary School Classrooms

David Weintrop[1], Alexandria Hansen[2], Danielle Harlow[2], Diana Franklin[3]

weintrop@umd.edu, akillian@umail.ucsb.edu,
dharlow@education.ucsb.edu, dfranklin@uchicago.edu

[1]College of Education and College of Information Studies, University of Maryland
[2]Gevirtz Graduate School of Education, UC Santa Barbara
[3]UChicago STEM Education and Computer Science, University of Chicago

Abstract

There is a growing push to bring computer science to all learners across the K-12 spectrum. While a great deal of resources and supporting research exist for high school learners, far less is known about how best to introduce younger learners to the powerful ideas of computer science in a classroom setting. This paper presents findings from a three-year study investigating approaches for supporting upper elementary learners (grades 4-6) in engaging with foundational computer science concepts. Specifically, we introduce a programming environment and curriculum designed for elementary-aged learners and present data showing both successes from this work as well as remaining challenges.

Introduction

There are growing calls to bring computer science to all learners. This can be seen in the number of national, state-level, and district-level initiatives seeking to make computer science instruction a part of every student's education. A critical component to make sure such efforts are successful is the creation of accessible, effective, and developmentally appropriate computer science educational materials. Historically, much of the effort to bring computer science into K-12 classrooms has focused on high school content and been framed as a way to prepare learners for future coursework. As the objective of computing education shifts from trying to prepare future software developers towards the larger goal of creating a computationally literate citizenry, and the focus shifts from high school to the entire K-12 spectrum, we need to rethink how we teach computer science, especially to younger learners. The challenge of how to make computer science accessible, engaging, and developmentally appropriate for youngers learners is the focus of this work. Specifically, in this work, we pursue the following two research questions:

1. What can upper elementary learners (4th-6th grade) achieve in a developmentally appropriate CS curriculum?
2. What challenges (both conceptual and pedagogical) emerge when bringing computer science into elementary classrooms?

To answer these research questions, we draw on data from a three year, design-based research study focused on the creation of a programming environment and curriculum for upper-elementary (grades 4-6, ages 9-12) learners. Given the large scope of these questions and the length constraints of this submission, we only begin to answer these two questions in this manuscript. The full version of this work will include more detail and additional analyses to further explore these questions.

Prior work

Computer Science in Elementary School Classrooms

The last decade has seen growing calls to teach the powerful ideas of computing across elementary and high school grades, with a number of framework documents, standards, and national curricula for computer science emerging to help bring computer science to K-12 classrooms (CSTA Standards Task Force, 2016; Furber, 2012; Google Inc. & Gallup Inc., 2016; "K–12 Computer Science Framework," 2016). At the same time, programming environments designed for younger learners, such as Scratch (Resnick et al., 2009), Scratch Jr. (Flannery et al., 2013), and Storytelling Alice (Kelleher, Pausch, & Kiesler, 2007) are becoming increasingly popular. A growing body of empirical research is finding that these tools and the block-based programming approach more broadly is making the powerful ideas of computing more accessible and engaging for K-12 learners (Armoni, Meerbaum-Salant, & Ben-Ari, 2015; Franklin et al., 2017; Grover, Pea, & Cooper, 2015; Kelleher et al., 2007; Lewis, 2010; Maloney, Peppler, Kafai, Resnick, & Rusk, 2008; Price & Barnes, 2015; Weintrop & Wilensky, 2017). To accompany these environments is a growing collection of curricula targeting elementary learners including Creative Computing (Brennan, 2013), Foundations for Advancing Computational Thinking (Grover et al., 2015), and Code.org's new CS Discoveries course.

Alongside these materials designed for elementary computer science classrooms is a body of research investigating how best to introduce computer science concepts across the pre-high school grades. Work investigating formal K-8 curricula is documenting both the successes and challenges of teaching computer science in elementary classrooms (Franklin et al., 2017; Grover & Basu, 2017; Seiter, 2015). Outcomes from this work are revealing developmental differences in learners by grade level as well as misconceptions some younger learners have of computer science concepts.

Finally, it is important to note that the notion of programming being a productive activity for elementary aged learners begins with the work of Papert and colleagues. Through their work with the Logo programming language, they found that programming was accessible to younger learners and could serve as a powerful learning practice (Harel & Papert, 1990; Papert, 1980; Papert, Watt, diSessa, & Weir, 1979). In many ways, the current push to bring computing into classrooms has its roots in this line of work which started decades earlier.

Methods and Materials

Study Design

This project employs an ecological, design-based research approach to create and evaluate a programming environment and curriculum designed to fit into existing upper elementary school classrooms (Barab & Squire, 2004; DBR Collective, 2003; Gaver, 1991; Harlow, Dwyer, Hansen, Iveland, & Franklin, Accepted; Norman, 1999). The data for this paper are drawn from a three-year study involving nearly 1,400 students in grades 4-6 (ages 9-12) across the state of California. The focus of the analysis presented in this paper comes from the second two years of the project

during which 4th and 5th grade students worked through the first module of our curriculum focused on digital story telling.

Participants and Data Collection

During the 15-week curriculum, students spent roughly one hour per week in class working on the curriculum. In the second year of the study, 450 students across 15 classrooms in 6 schools worked through the curriculum's digital story telling module. The primary data source for the analysis presented in this work are the programs authored by students while they worked through the curriculum. To record this information, we developed a system of tracking student progress in a way that preserved the paths they took (productive or not) toward accomplishing the computer task by taking sequential snapshots of their projects. Additionally, each class session was observed by researchers, with a subset being video recorded for later analysis.

The LaPlaya Programming Environment & KELP-CS Curriculum

Participants in this study programmed using the LaPlaya environment while they worked through the KELP-CS curriculum, both of which were designed for elementary-aged learners. LaPlaya (Figure 1) is a visual, block-based programming environment designed to support both guided and open-ended exploration for upper elementary school students. The environment is built on top of the Snap! programming environment (Harvey & Mönig, 2010) and presents learners with a block-based programming interface and a stage where they can create programs that control on-screen characters, called sprites, as they move around on a two-dimensional stage.

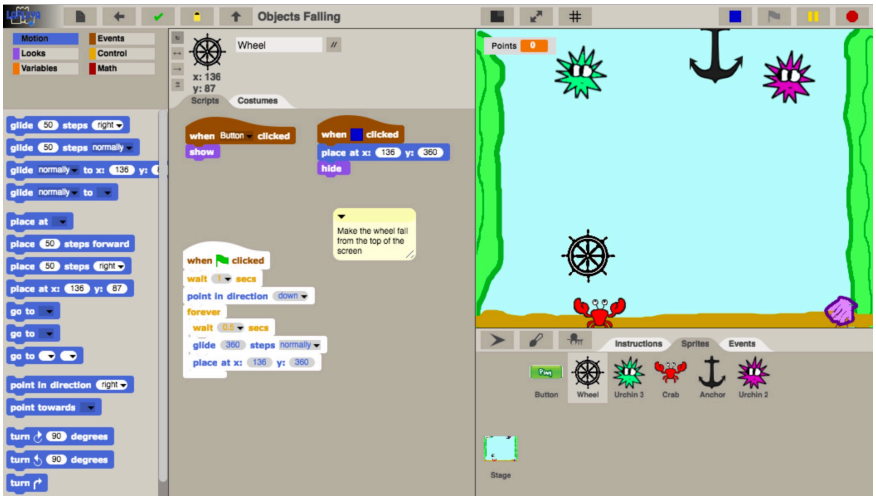

Figure 1. The LaPlaya programming environment.

To bring LaPlaya into the classroom, we created the KELP-CS curriculum. KELP-CS is designed to provide a developmentally-appropriate introduction to foundational computer science concepts. The curriculum gradually introduces computer science concepts to students as they progress. For example, to learn how to create event-driven programs that use the keyboard, the students start by following a template to program a rocket ship to move up when the up key is pressed. The second activity asks them to program a sprite to move left and right based on key

presses without a template to follow, and then finally, in the third activity, students program that allows the user to move an on-screen car in all four directions by using the keyboard. Module 1, which is the focus of this paper, covers a number of topics (outlined in Table 1) and culminates with students designing and implementing a digital story.

Table 1. The story telling module of the KELP-CS curriculum including data showing the prevalence of each concept in students' final projects.

Unit #	Concept	Number of Activities	Percent of students who incorporated this concept into their final project
1	Sequence & Interface	3	67.3%
2	Breaking Down Actions	4	35.5%
3	Event 1: On Sprite Clicked	3	72.0%
4	Event 2: Other Sprite Clicked	4	
5	Event 3: On Key Pressed	3	
6	Initialization	3	79.4%
7	X/Y Coordinates	5	69.2%
8	Costume Changes	3	28.0%
9	Scene Changes	3	64.5%

Findings

Outcomes from Learning to Program with LaPlaya and KELP-CS

In this first findings section, we focus on the first six the KELP-CS curriculum. Using an automated evaluation system, every program for each unit was evaluated to identify if the submitted work demonstrated an understanding of the concept at hand. Figure 2 shows the percent of students who successfully completed each activity for each unit, broken down by grade.

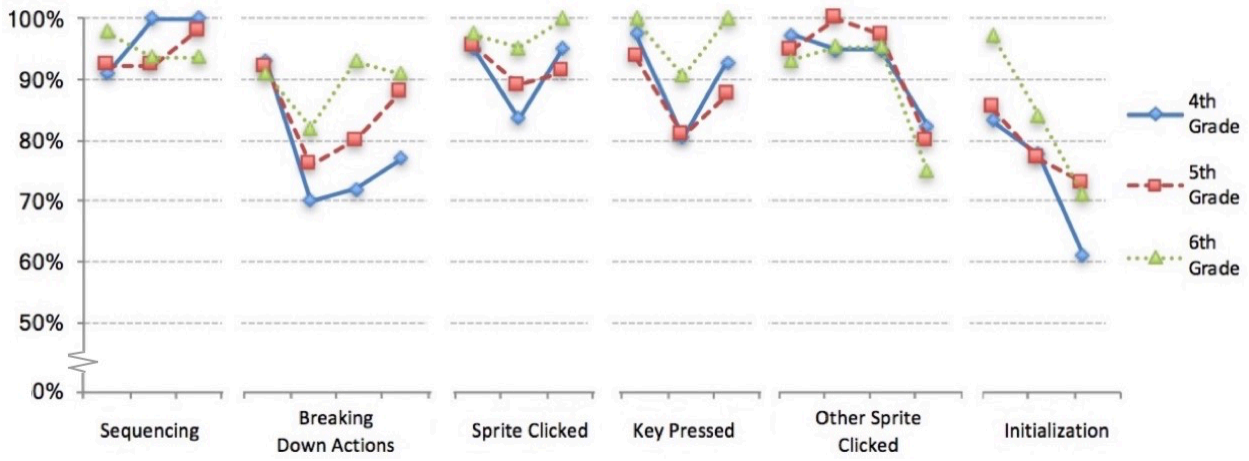

Figure 2. Average student scores on the first module of KELP-CS, grouped by grade.

The first thing to note in Figure 2 is that the y-axis is not continuous, jumping from 0% to 50%, this was necessary because the student scores were so high throughout the curriculum. The lowest average score for any single activity was 4th grade learners on the very last structured activity, where the mean score was 61.1%. The second thing to note about Figure 2 is how similar the scores are across grades, suggesting the curriculum was successful as supporting learners with differing incoming computer science knowledge. A more detailed comparative analysis of these outcomes revealing differences between the grades can be found in (blinded).

Outcomes from Open-ended LaPlaya Final Projects

The KELP-CS curriculum culminates with an open-ended final project that asks learners to create a project of their own design. To see if and how students incorporated the concepts from the curriculum into their own-ended final projects, we hand-coded all 135 culminating projects created by the students whose data was presented in the previous section. The far-right column in Table 1 shows the percentage of students that included the concept of that unit in their final project. It is important to note, students were not required to incorporate these concepts, so in some cases, projects did not include the concept listed because it did not make sense in the project design. The main thing to note about these results is the relatively high frequency that students used the concepts from the structured activities in their final projects. Sixty-seven percent of projects included sequencing logic, 79.4 % initialized their program, and 72.0% of projects included user-initiated events like mouse-clicks or key presses.

The final critical piece of these culminating projects is the creative and expressivity shown by the students. All but 9 of the projects (91.6% in total) included either custom art or imported images as ways to personalize the project. Additionally, the content of the projects ranged from underwater adventures to gameshows, and superhero rescues to animated stories about shopping, space travel, and magic tricks. Figure 3 shows three images of student final projects.

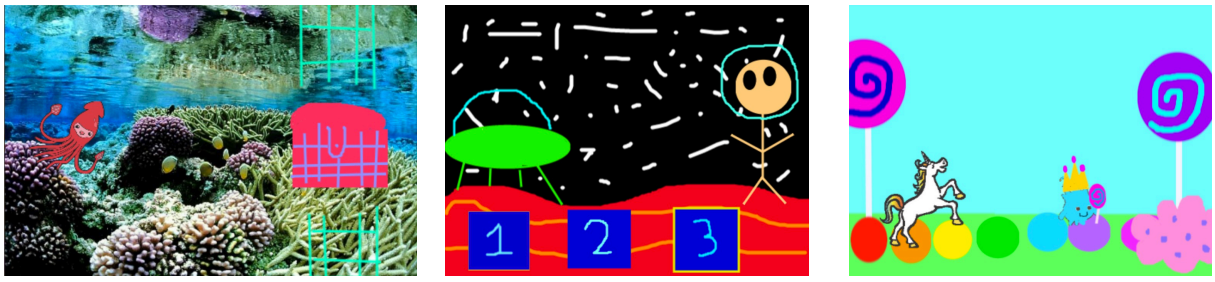

Figure 3. Three example final projects created by students at the end of the KELP-CS curriculum.

Challenges of Teach Computer Science in Elementary Classrooms

While this project revealed many successes related to teaching computer science concepts in elementary classrooms, there were also a number of challenges identified. These challenges related to the concepts, curriculum, and tools. In this section, we discuss one such challenge associated with the choice of programming environment and discuss possible ways that educators might respond. In the full version of this paper, we will more fully explore other challenges identified in this work.

Coordinating events with Event-based programming

In LaPlaya and other Scratch-style programming environments, to run a program you associate an event with it, be it clicking a start button, waiting for an in-program event, or binding a script to a key press. This is different from many general-purpose programming languages where there is a single main function that is explicitly called that begins the serial execution of the program. Events are an intuitive and accessible way to engage novices and younger learners with programming and were very common in student final projects. Students used an average of 13.5 events (SD = 20.8) per project, with 27 students using more than 20 events in their projects, and 6 students using more than 50 event blocks.

Another feature of event-driven programming is that it makes it easy to create interactive programs by giving the programmer direct control over how and when behaviors are run. In this way, it helps achieve the low-threshold to programming sought by the Scratch designers and contributes to the engagement and enjoyment of the environment (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010). One potential outcome from learning to program in event-driven programming is developing programming practices that are unique to the event-based paradigm and do not have natural analogs in conventional programming languages. For example, students can bind multiple scripts to the same event for the same sprite *even though the events were not intended to execute in parallel*. Figure 4 depicts an example of this found in one student's final project where four scripts are defined for the `when left arrow key pressed` event of a single sprite (the project had a total of 12 `when left arrow key pressed` blocks). In composing these commands, the learner directly mapped a single event to multiple actions. Conceptually, this both makes sense and is an intuitive approach for achieving a behavior such as making multiple things happen after a single key stroke. However, this also circumvents the need to define commands in a sequential manner, i.e. the learner does not need to define the steps one after the other in a single script. While this is a functional solution, it is not how the same outcome would be achieved in a non-event-based context. This distinction is meaningful because if all the commands shown in Figure 4 were moved into a single script, the numerical values in the `wait` commands would need to change, meaning the shift is not just reorganizing commands, but the underlying logic would need to change as well.

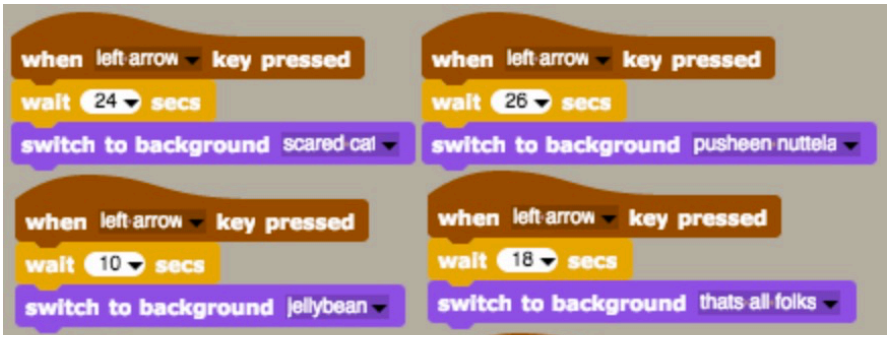

Figure 4. Four of the 12 `when left arrow key pressed` scripts defined in one students' final project.

The program shown in Figure 4 is one example of the more general outcome of learners developing programming practices that leverage features of Scratch-style programming environments. This is to be expected of novices with little prior experience and shows how they take advantage of affordances provided by such introductory programming tools. This finding suggests that teachers of more advanced courses need to be aware of these practices and be prepared to help students move from the parallel thinking supported by events toward the linear, sequential ordering of commands imposed by the programming languages used in later instruction.

Conclusion

The current excitement around bringing computer science to learners across the K-12 spectrum poses a challenge to those in the computer science education community. While we have decades of work to draw upon when thinking about how to make the powerful ideas of computing accessible and meaningful to high school-aged learners, it is less clear how to bring computer science into elementary school classrooms. In the paper, we present findings from one project designed to do just that, reporting both success of the project as well as one example of an unexpected outcome. With this work, we contribute to the growing body of literature on how best to introduce computer science into elementary classrooms, bringing us one step closer to realizing the goal of computer science for all.

Reference

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "Real" Programming. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 25:1-15.

Barab, S., & Squire, K. (2004). Design-based research: Putting a stake in the ground. *The Journal of the Learning Sciences*, *13*(1), 1–14.

Brennan, K. (2013). Learning computing through creating and connecting. *Computer*, *46*(9), 52–59.

CSTA Standards Task Force. (2016). *K–12 Computer Science Standards*. ACM. New York, NY. Retrieved from https://csta.acm.org/Curriculum/sub/K12Standards.html

DBR Collective. (2003). Design-based research: An emerging paradigm for educational inquiry. *Educational Researcher*, 5–8.

Flannery, L. P., Silverman, B., Kazakoff, E. R., Bers, M. U., Bontá, P., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. In *Proceedings of the 12th International Conference on Interaction Design and Children* (pp. 1–10). ACM.

Franklin, D., Skifstad, G., Rolock, R., Mehrotra, I., Ding, V., Hansen, A., … Harlow, D. (2017). Using Upper-Elementary Student Performance to Understand Conceptual Sequencing in a Blocks-based Curriculum. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 231–236). New York, NY, USA: ACM. https://doi.org/10.1145/3017680.3017760

Furber, S. (2012). Shut down or restart? The way forward for computing in UK schools. *The Royal Society, London*.

Gaver, W. W. (1991). Technology affordances. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 79–84). ACM. Retrieved from http://dl.acm.org/citation.cfm?id=108856

Google Inc., & Gallup Inc. (2016). *Trends in the State of Computer Science in U.S. K-12 Schools*. Retrieved from http://goo.gl/j291E0

Grover, S., & Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 267–272). New York, NY: ACM Press. https://doi.org/10.1145/3017680.3017723

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237. https://doi.org/10.1080/08993408.2015.1033142

Harel, I., & Papert, S. (1990). Software design as a learning environment. *Interactive Learning Environments*, *1*(1), 1–32.

Harlow, D., Dwyer, H., Hansen, A., Iveland, A., & Franklin, D. (Accepted). Ecological design based research in computer science education: Affordances and effectivities for elementary school students. *Cognition and Instruction*.

Harvey, B., & Mönig, J. (2010). Bringing "no ceiling" to Scratch: Can one language serve kids and computer scientists? In J. Clayson & I. Kalas (Eds.), *Proceedings of Constructionism 2010 Conference* (pp. 1–10). Paris, France.

K–12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org

Kelleher, C., Pausch, R., & Kiesler, S. (2007). Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 1455–1464).

Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: Logo vs. Scratch. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 346–350). New York, NY.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with Scratch. *ACM SIGCSE Bulletin*, *40*(1), 367–371.

Maloney, J. H., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 16.

Norman, D. (1999). Affordance, conventions, and design. *Interactions*, *6*(3), 38–43.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic books.

Papert, S., Watt, D., diSessa, A., & Weir, S. (1979). *Final report of the Brookline Logo Project: Project summary and data analysis (Logo Memo 53)*. Cambridge, MA: MIT Logo Group.

Price, T. W., & Barnes, T. (2015). Comparing Textual and Block Interfaces in a Novice Programming Environment (pp. 91–99). Presented at the ICER '15, ACM Press. https://doi.org/10.1145/2787622.2787712

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., … Silver, J. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60.

Seiter, L. (2015). Using SOLO to Classify the Programming Responses of Primary Grade Students. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 540–545). New York, NY, USA: ACM. https://doi.org/10.1145/2676723.2677244

Weintrop, D., & Wilensky, U. (2017). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 3. https://doi.org/10.1145/3089799