Making Computer Science Accessible

By Alexandria Hansen, Jim Gribble, Amber Moran, Eric Hansen, and Danielle Harlow

"Computer science is like magic—you can make anything you imagine!" said one fourth grader who had just made his first computer program. He was correct. Computer science (CS) is the superpower of the 21st century. In an increasingly digital world, the elementary school students of today who learn to program will be positioned to re-imagine and build the future. As educators, it is essential to consider how such an important skill is made accessible for a diverse range of students, including those diagnosed with learning disabilities.

While many efforts to teach computer science originate in afterschool clubs or summer camps (e.g., Franklin et al. 2011), to reach all students, computer science must be taught during the academic school day. Further, there has been a recent explosion of novice-friendly programming environments and curricula that teachers can use to teach themselves as well as engage their students in learning computer science (see Table 1 for a list of free resources).

In this article, we build on previous work conducted to connect computer programming to science learning. In Hansen et al. (2015), we described an engaging activity that challenged fourth-grade students to program a digital story or animation depicting science content such as how volcanoes form or the stages of chemical reactions. We now revisit the same activity through the lens of Universal Design for Learning (UDL) and discuss how our activity was adapted to support the learning of children with learning disabilities in the classroom.

NOVICE-FRIENDLY CS PROJECTS

While implementing CS projects in the elementary classroom may seem daunting, we advocate for integrating CS into the rest of the learning happening in a classroom, across multiple disciplines. In this section, we describe two examples of novice-friendly CS projects we completed with elementary school students: (1) program a digital story, and (2) create digital animations of physical systems. Then, we describe how we used the UDL framework to ensure the activities were accessible to all of our students.

In all activities, we used a Scratch-style (scratch.mit.edu; see Figure 1) programming environment, which allows users to drag and drop colorful *blocks* (or coding commands) into longer *scripts* (or lines of code) to control the animated actions of *sprites* (programmable characters or objects),

TABLE 1

FREE Resources available for elementary school teachers and students to learn CS.

Programming Environments	Curricula
Scratch scratch.mit.edu	ScratchEd scratched.gse.harvard.edu
Alice alice.org	Code Studio code.org
Tynker tynker.com	CS First cs-first.com
Code Academy codeacademy.com	CS Unplugged csunplugged.org
Agentsheets agentsheets.com	Exploring Computer Science exploringcs.org
Blockly developers.google.com/blockly/	Teacher-led Hour of Code Lessons csedweek.org/educate/curriculum/teacher-led
MIT App Inventor appinventor.mit.edu/explore/	Coding for Kids – created by an author of this paper! udemy.com/coding-for-kids

which are displayed on the *stage*.

Digital Stories. Creating a cartoonlike animated version of written stories through block-based coding (called a digital story) links CS directly to other classroom work involving literacy. Students can program digital story versions of books that they have read in class, or write their own stories to later program. In one specific project connected to science, we tasked fourth-grade students with programming a digital story to communicate science concepts. For example, one student programmed an animation about the formation of volcanoes (4-ESS2-2) while another student focused on the phase changes involved in making ice cream (2-PS1-3) (see Figure 2 for an example). Our students created stories in six, 45-minute class sessions, concluding with a gallery walk to view what others had programmed. While we allowed students to select the science content of their stories, teachers could assign specific topics. Further, this activity could easily be modified to support students in grades 2–6. For younger students who are still learning to read, we recommend using Scratch Jr. (*www.scratchjr.org*) for its inherently visual format. See Hansen et al. (2015) for more specific details about how we facilitated this project with our students.

Digital Animations of Physical Systems. Another novice-friendly computer programming project is creating digital animations of physical systems. We tasked students with creating digital versions of Rube Goldberg machines to learn about forces and motion and simple machines (3-PS2-1) before constructing a physical version to move a series of marbles to designated locations around the classroom. First, students worked in small groups to program their digital linkage in Scratch to visualize their simple machine and how certain motions would cause objects to move in different ways. Then, students were provided with reused, generally recycled, materials such as paper towel rolls, marbles, egg cartons, tape, rubber bands, toy cars, old Legos, and string. They also had access to giant wooden peg boards and were encouraged to use the recycled materials to create linkages, or mechanisms to move a marble across or down the peg board.

FIGURE 1

Scratch-style programming environment used by students to create stories and animations.

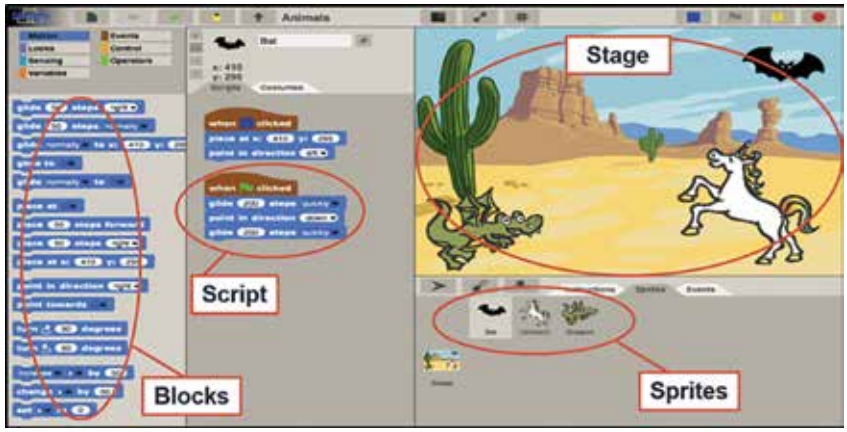
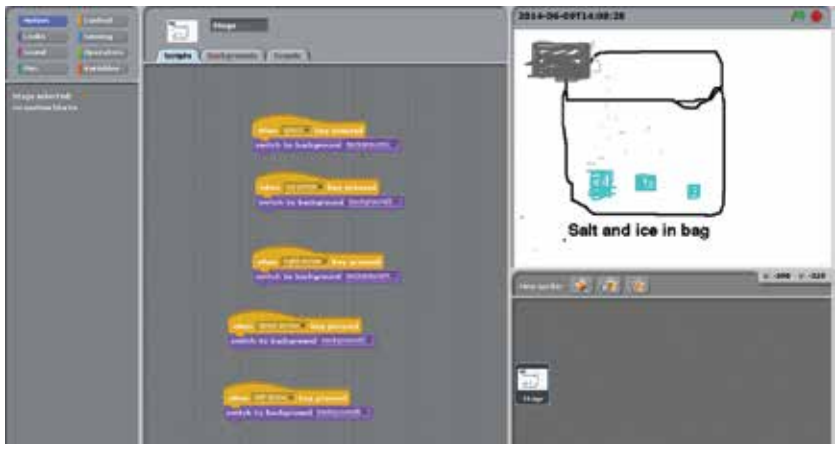


FIGURE 2

Digital animation depicting phase changes involved in making ice cream. Note this was programmed in a test-version of Scratch, so the interface varies slightly.



Students worked in small groups to design one linkage independently, but were also required to communicate with peers nearby to ensure the linkages connected effectively to form one collaborative Rube Goldberg machine. For example, one group constructed a catapult using pencils and rubber bands to launch a marble through a box and down an inclined plane, where it hit a toy car. This triggered the release of another toy car, wedged between rulers, which knocked a toy bus (attached to a string) off a bookshelf and into another marble, before traveling down a pegboard to the next group's linkage (see Figure 3 for the digital version). We share this specific example because it involved a student with a moderate learning disability who was able to program the animation shown in Figure 3 and successfully build the physical linkage to connect with the larger classroom machine. See Figure 4 for another programmed linkage created by a student for this project.

MAKING COMPUTER SCIENCE ACCESSIBLE

Scratch and similar block-based programming environments are naturally differentiated. These environments are described as having "low floors" and "high ceilings" (Resnick and Silverman 2005), indicating that it is easy for novices to begin (low floors), but there is the potential for advanced students to challenge themselves (high ceilings). Because of these design aspects, computer programming in block-based languages, such as Scratch, is particularly well-suited for students of all abilities and levels, including students with learning disabilities.

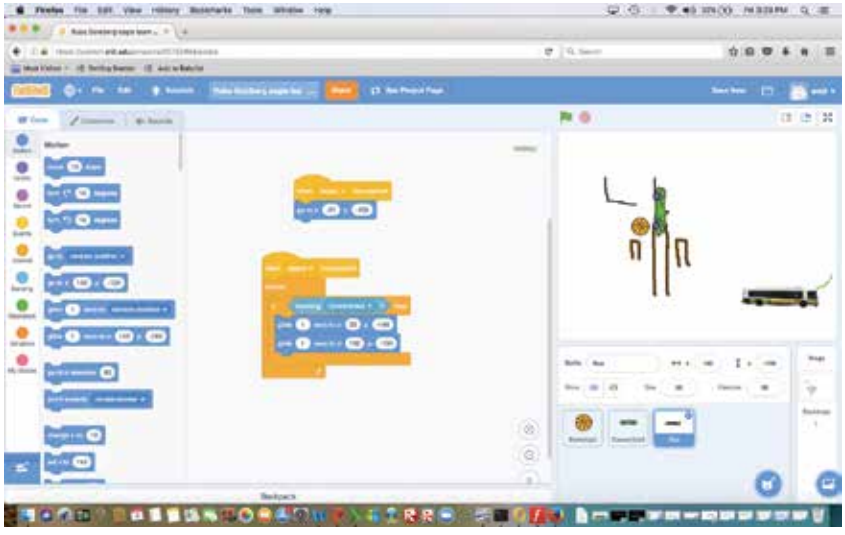
Teachers, however, must still design instruction to support learning computer programming skills. We drew from the Universal Design for Learning (UDL) framework, which calls for design features of instruction that are *essential* for some students, *beneficial* to others, and *not detrimental* to any (King Sears 2014). At the core of UDL is flexibility. Teachers should allow students multiple options to experience the content, or multiple means of "representation, expression, and engagement" (Hall, Meyer, and Rose 2012). For example, a basic UDL strategy that provides multiple representations might involve a teacher providing information to the learner in more than one format, such as through printed handouts, written on the board, and delivered orally. Additionally, a teacher might accept multiple means for response from a student in terms of work completion; responses ranging from written, drawn, oral, or responses communicated through computer programming. See Hansen et al. (2016) for a full description of how UDL guided our work.

Following, we share practical UDL recommendations to support educators interested in inclusive computer science learning experiences at school (Center for Applied Special Technology 2018).

Consider the available technology. While the computer itself is technology, consider how to optimize access to the computer through other types of assistive technology available to students. We found that computer mice are especially important for all young students learning to code (Hansen et al. 2015), but for a specific student with fine motor difficulties, providing a computer mouse allows access that a touchpad on a laptop might not. Additionally, Scratch features a variety of sounds to include while programming, so we found that providing the option of using headphones helped minimize distractions for students

FIGURE 3

An example of a programmed linkage created by a student as part of a collaborative Rube Goldberg machine classroom project.



who were easily distracted. We also used a large overhead screen or projector to review important coding commands and concepts with the whole class. Providing flexibility and choice in what types of technology are made available to students is a UDL strategy we focused on to ensure our lessons were accessible for everyone.

Consider the setting and seat. We found that children work differently in traditional computer labs compared to working at a laptop within the normal classroom (Gribble et al. 2017). Consider what you are trying to accomplish as a teacher when selecting a setting. Are you expecting students to work quietly and independently? A computer lab might be a better fit to allow students to focus their attention. If you're expecting them to collaborate and work in groups, perhaps using laptops or tablets in the normal classroom is a better approach. Further, consider individual seating assignments during computing tasks to help students sustain effort and persistence. We intentionally positioned students who struggled near the front of the classroom or computer lab for easier access to a teacher. Finally, consider where the students who need additional support are in relation to one another in order to optimize support from the teacher. In our work, we placed students who needed additional support near one another so that a teacher could easily engage in small-group instruction to review important concepts when necessary. We also provided our students flexible seating options in the form of standing desks and exercise balls to help them focus their attention on the computing task. Providing variation and flexibility in seating options is a UDL consideration.

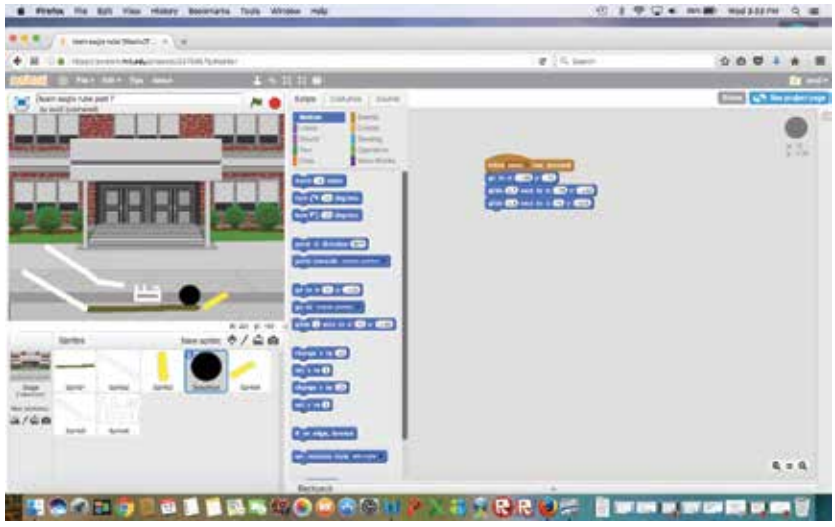
Consider collaboration and community. To foster collaboration, we used pair programming as it provides both students an important role. One student assumes the role of the "driver" and remains in control of the computer and mouse. The other student assumes the role of the "navigator" and is in charge of communicating coding decisions to their partner to implement. It is important that students are provided opportunities to serve in both roles. We frequently had students switch halfway through a lesson. It is also important to consider how partner pairs are formed. We found heterogeneous pairs (of mixed abilities) to be the most effective approach to ensure all students can experience success. This video provides a helpful overview of pair programming: *www.youtube.com/watch?v=vgkahOzFH2Q*.

Consider ways to activate background knowledge. We found it essential for students to brainstorm ideas on paper before turning to a computer. For example, in our digital story project, we had students create hand-drawn storyboards before programming (see Figure 5 for an example). This was helpful for students, but also served as a formative assessment opportunity for the teacher to gauge content knowledge and programming skills. Additionally, we found success engaging parents at home through aspects of a flipped classroom. We assigned parent's homework in the form of watching videos that introduced computer science and the programming interface so they were better positioned to help their children and become involved in their learning.

Consider the use of available adults. Often, students with learning disabilities are assigned a paraprofessional aide to support them. Get this additional adult involved, but do so with guidelines. We found it helpful to have a separate meeting with paraprofessionals to acquaint them with Scratch and upcoming projects. It was also helpful to establish some basic

FIGURE 4

Example of a computer program created for a Rube-Goldberg machine.

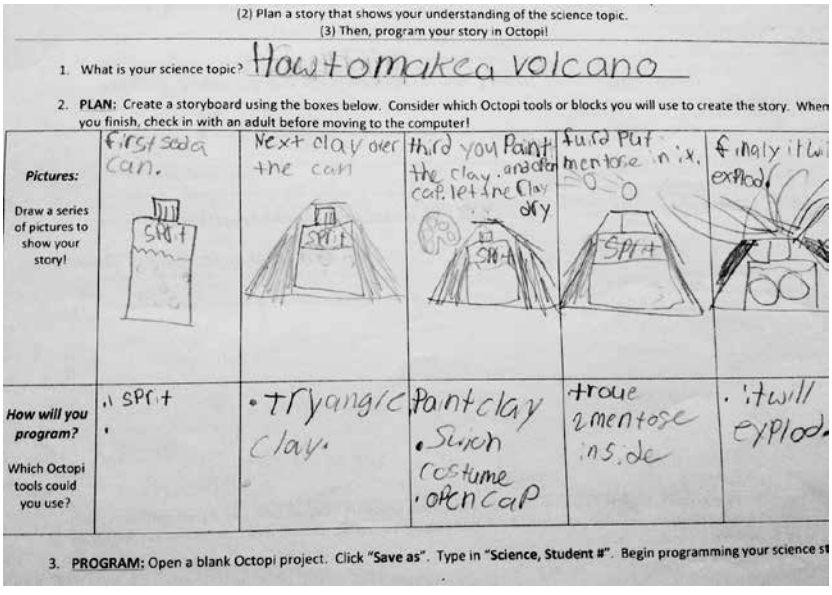


ground rules, such as to not take over the computer for the student. Instead, provide guidelines to help facilitate student coping skills (e.g., taking a short break) or prompt metacognitive learning strategies toward the main lesson objective. Snodgrass, Israel, and Reese (2016) found that students with severe learning disabilities had adults more frequently take over the computing task for them, essentially limiting the learning opportunities. Adults should avoid taking over the computing tasks unless a student has a severe motor deficiency. In this case, we advise using the pair programming approach previously described, with the student assuming the role of the "navigator." Parent classroom volunteers are also especially poised to help and can assume the role of the "driver" with students navigating the coding decisions.

Consider multiple pathways for students. At the core of UDL is the notion of flexibility. Teachers must be willing and able to provide flexible assignment options for students. Scratch, in particular, allows students to create exceptionally complex things. If a student is ready for a challenge beyond what the class was assigned, she can explore the seemingly endless features of Scratch to enhance projects. This became an essential feature of our classroom for students who finished early—they were allowed to openly program anything of their choosing. In contrast, if a student was struggling to the point of frustration, it was important for us to modify the task. In our work, this frequently took the form of lessening the requirements in the project alongside targeted support from a teacher or aide. For instance, when students were tasked with programming a digital story with three distinct scenes, struggling students had the option to create only two scenes, while students who needed more of a challenge were prompted to add advanced computing concepts (variables, sensing) and external hardware (Makey Makey, Lego WeDo Robotics, etc.). Scratch projects allow for natural differentiation, but teachers must make the instructional decisions as to when modifications are necessary.

FIGURE 5

An example storyboard of a digital story programmed by a fourth-grade student about volcanoes.



CONCLUSION

Scratch and other block-based programming environments are intentionally designed to allow learners of all ages and skills to program complex creations (Resnick and Silverman 2005). We have shared lessons learned from our work over the past five years using UDL in the context of elementary computer science education. We hope these suggestions offer a foundation for other educators who are interested in providing greater access to computer science for a diverse range of students—including those diagnosed with learning disabilities.

REFERENCES

Center for Applied Special Technology. 2018. *Universal design for learning guidelines version 2.2*. Retrieved from *http://udlguidelines.cast.org*

Franklin, D., P. Conrad, G. Aldana, and S. Hough. 2011. Animal tlatoque: attracting middle school students to computing through culturally-relevant themes. In *Proceedings of the 42nd ACM technical symposium on computer science education* (pp. 453–458). Association for Computing Machinery (ACM).

Gribble, J., A.K. Hansen, D.B. Harlow, and D. Franklin. 2017. Cracking the code: The impact of computer coding on the interactions of a child with autism. In *Proceedings of the 16th International Conference on Interaction Design and Children* (IDC '17). Palo Alto,

CA: Association for Computing Machinery (ACM).

Hall, T.E., A. Meyer, and D.H. Rose. 2012. *Universal design for learning in the classroom: Practical applications*. Chicago: Guilford Press.

Hansen, A.K., A. Iveland, H.A. Dwyer, D. Franklin, and D.B. Harlow. 2015. Programming science digital stories: Computer science and engineering design in the science classroom. *Science and Children* 53 (3): 60–64.

Hansen, A.K., E.R. Hansen, H.A. Dwyer, D.B. Harlow, and D. Franklin. 2016. Differentiating for diversity: Using universal design for learning in computer science education. In *Proceedings of the 47th Technical Symposium on Computer Science Education* (SIGCSE '16). Memphis, TN: Association for Computing Machinery (ACM).

King Sears, P. 2014. Introduction to learning disability quarterly special series on universal design for learning. *Learning Disability Quarterly* 37 (2): 68–70.

Resnick, M., and B. Silverman. 2005. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 conference on Interaction design and children,* pp. 117–122. ACM. Association for Computing Machinery (ACM).

Snodgrass, M.R., M. Israel, and G.C. Reese. 2016. Instructional supports for students with disabilities in K–5 computing: Findings from a cross-case analysis. *Computers and Education*. 100: 1–17.

Alexandria Hansen (*akhansen@csufresno.edu*) is an assistant professor of STEM Education at California State University, Fresno. **Jim Gribble** is a PhD student, and **Amber Moran** is an assistant teaching professor, both at the University of California, Santa Barbara. **Eric Hansen** is a mild/moderate special education teacher and master's student at California State University, Fresno. **Danielle Harlow** is a professor and associate dean at the University of California, Santa Barbara.