Journal of Information Technology Education: Innovations in Practice

An Official Publication of the Informing Science Institute
InformingScience.org

JITEiip.org

Volume 18, 2019

COMPUTER SCIENCE EDUCATION IN EARLY CHILDHOOD: THE CASE OF SCRATCHJR

Marina Umaschi Bers* The DevTech Research Group, Tufts University, Medford, MA, USA Marina.Bers@tufts.edu

Amanda Sullivan The DevTech Research Group, Tufts University, Medford, MA, USA Amanda.Sullivan@tufts.edu

*corresponding author

ABSTRACT

Aim/Purpose: This paper aims to explore whether having state Computer Science standards in place will increase young children's exposure to coding and powerful ideas from computer science in the early years.

Background: Computer science education in the K-2 educational segment is receiving a growing amount of attention as national and state educational frameworks are emerging. By focusing on the app ScratchJr, the most popular free introductory block-based programming language for early childhood, this paper explores if there is a relationship between the presence of state frameworks and ScratchJr's frequency of use.

Methodology: This paper analyzes quantitative non-identifying data from Google Analytics on users of the ScratchJr programming app. Google Analytics is a free tool that allows access to user activity as it happens in real time on the app, as well as audience demographics and behavior. An analysis of trends by state, time of year, type of in-app activities completed, and more are analyzed with a specific focus on comparing states with K-12 Computer Science in place versus those without.

Contribution: Results demonstrate the importance of having state standards in place to increase young children's exposure to coding and powerful ideas from computer science in the early years. Moreover, we see preliminary evidence that states with Computer Science standards in place support skills like perseverance and debugging through ScratchJr.

Findings: Findings show that in the case of ScratchJr, app usage decreases during the summer months and on weekends, which may indicate that coding with ScratchJr is more often happening in school than at home. Results also show that states with Computer Science standards have more ScratchJr users on aver-

Accepting Editor Dennis Kira | Received: July 26, 2019 | Revised: September 16, September 18, 2019 | Accepted: September 19, 2019.
Cite as: Bers, M. U., & Sullivan, A. (2019). Computer science education in early childhood: The case of ScratchJr. *Journal of Information Technology Education: Innovations in Practice, 18,* 113-138.
https://doi.org/10.28945/4437

	age and have more total sessions with the app on average. Results also show preliminary evidence that states with Computer Science standards in place have longer average session duration as well as a higher average number of users returning to edit an existing project.
Recommendations for Practitioners	Successful early childhood computer science education programs must teach powerful ideas from the discipline of computer science in a developmentally appropriate way, provide means for self-expression, prompt debugging and problem solving, and offer a low-floor/high-ceiling interface for both novices and experts. Practitioners should be aware in drops in computer science learning during the summer months when school is not in session.
Recommendation for Researchers	Researchers should consider the impact of state and national frameworks on computer science learning and skills mastered during the early childhood years. Researchers should look for ways to continue engaging students in computer science education during times when school is not in session.
Impact on Society	Results demonstrate the importance of having state CS standards in place to increase young children's exposure to coding and powerful ideas from computer science in the early years. Moreover, we see preliminary evidence that states with Computer Science standards in place support skills like perseverance and debugging through ScratchJr.
Future Research	Future research should continue collecting Google Analytics from the ScratchJr app and track changes in usage. Future research should also collect analytics from a wide range of programming applications for young children to see if the trends identified here are consistent across different apps.

Keywords: early childhood, computer science, coding, STEM, policies, frameworks

INTRODUCTION

Each month, there is an estimated 500,000 openings for computing jobs nationwide, and a lack of adequately trained people to fill these positions (Code.Org, 2018). According to the Bureau of Labor Statistics, computer related occupations are projected to yield over 1 million job openings from 2014 to 2024 (Fayer, Lacey, & Watson, 2017). In less than ten years from now, it is estimated that the United States will need 1.7 million more engineers and computing professionals (Corbett & Hill, 2015).

In order to meet the growing needs of our nation's technical fields, there has been an increase in new educational policies and frameworks at the federal and state level to prepare K-12 students for computer science related professions. However, while most of the implementation is happening at the late elementary, middle school and high school levels, the frameworks mandate to start in kindergarten.

There are both economic and developmental reasons for the choice to start early. Research shows that educational interventions that begin in early childhood are associated with lower costs and more durable effects than interventions that begin later on (e.g., Cunha & Heckman, 2007; Heckman & Masterov, 2007). Two National Research Council reports—*Eager to Learn* (Bowman, Donovan, & Burns, 2001) and *From Neurons to Neighborhoods* (Shonkoff & Phillips 2000) detail the importance of early experiences for later school achievement. Furthermore, research explores how children who are exposed to STEM curriculum and programming at an early age demonstrate fewer gender-based stereotypes regarding STEM careers, an increased interest in engineering (Metz, 2007; Steele, 1997; Sullivan & Bers, 2018) and fewer obstacles entering these fields later in life (Madill et al., 2007; Markert, 1996). Furthermore, research suggests that for addressing the under-representation of women in Computer Science is critical to improve early education experiences in this area (Sullivan, 2019; Sullivan & Bers, 2018; Varma, 2010).

State and national frameworks are slowly beginning to catch up with research on the importance of early computer science education. At the state level, at the writing of this paper, 25 US states have K-12 computer science standards in place and an additional 10 have them in progress. This means that nearly half (49%) of all U.S. states have or will have standards in place, while 25% still have no standards nor a plan to put them in place at this time. But how successful are these standards at actually increasing the amount and quality of computer science education in the early years? The purpose of this study is to explore whether having state Computer Science standards in place will increase young children's exposure to coding and powerful ideas from computer science in the early elementary years. With the rapid growth in K-12 computer science frameworks in recent years, this paper fills a necessary gap in the literature by looking at the relationship between policy and practice: standards and children's experiences with ScratchJr. As of yet, no other researchers have begun to explore the impact of state CS standards on students' usage of different programming applications.

It is important to note, however, that for computer science education to start effectively in the early years, when children are just starting to develop literacy and numeracy skills as well as learn "schooling", there is a need not just for frameworks but for both pedagogical approaches and technologies that are developmentally appropriate for young children (Bers, 2018b). In the Literature Review in the following section, current approaches to early childhood computer science education are categorized into four groups: unplugged computer science, block-based programming languages, programming games, and introductory robotic systems. An analysis of each of these categories shows that early childhood computer education must provide opportunities to expose young children to powerful ideas from the discipline of computer science, provide tools that engage them in self-expression, prompt debugging and problem solving, and offer a low-floor/high-ceiling interface. By focusing on the case of ScratchJr, the most popular free introductory block-based programming language for early childhood which offers all four dimensions described earlier, the case study presented in this paper explores if there is a relationship between the presence of state frameworks and ScratchJr's frequency of use. This allows us to further investigate the complex interplay between policies and practices and to discuss implications for state and national policies that take into consideration developmentally appropriate practice and equitable computer science education in the early years. An analysis of trends by state, time of year, type of in-app activities completed, and more are analyzed with a specific focus on comparing states with K-12 Computer Science in place versus those without will be presented. To contextualize this study, the Literature Review in the following section synthesizes research on computer science education in early childhood and current data on K-12 computer science frameworks in the U.S.

LITERATURE REVIEW

EARLY CHILDHOOD COMPUTER SCIENCE EDUCATION

The push for Computer science education in the United States has grown in conjunction with the STEM (Science, Technology, Engineering, Mathematics) education movement (Bers, 2019). STEM first came into the American consciousness in the 1950's during the height of the Space Race when the United States passed the National Defense Education Act, which provided funding and incentives for schools. But it was the creation of the LOGO computer language by Seymour Papert, Wally Feurzeig and colleagues in 1967 that is generally described as the beginning of Computer Science Education in elementary schools in the U.S. (Blikstein, 2018).

LOGO was the first widely disseminated programming language designed for children (Papert, 1980b). LOGO allows children to explore programming concepts by giving instructions for moving either an onscreen turtle or a small turtle robot on the floor. LOGO prompted students to engage in self-reflection and meta-cognition by thinking about their own thinking, expressing themselves through their programs, and debugging them when things did not work (Guzdial, 2003; Papert, 1980b). Although LOGO became popular as a tool for supporting new ways of thinking about math

and creating microworlds for exploring geometry, angles, and functions (Abelson & DiSessa, 1982; Clements & Sarama, 1996), that was not Papert's intent (Bers, 2008). Papert wanted children to learn to think in new ways, about math, about computer science, about writing, about any subject and, most importantly, about the nature of thinking. Over time, LOGO grew in popularity all over the world, and versions of LOGO were implemented in more than a dozen languages and on a variety of machines (The Logo Foundation, 2015). For example, LOGO for MSX computers became very popular in Europe, South America, and Japan while Atari LOGO and Commodore LOGO were popular in North America (The Logo Foundation, 2015).

Thousands of teachers started to create community networks and curricula for LOGO, and research slowly embarked on understanding its impact. The presence of computer science in the classroom required the adjustment of the school curriculum to make room for the new computer science discipline. Many of the studies set to explore how the gains acquired by learning programming could transfer to other content and skills. A large-scale study of children using the Logo programming language showed that children in grades K-6 scored significantly higher on tests of mathematics, reasoning, and problem-solving (Clements, Battista, & Sarama, 2001) and children who used LOGO in kindergarten were also found to have sustained attention, self-direction, and took pleasure in discovery (Clements, 1987).

Other studies researched the effects of LOGO programming on mathematics knowledge (Feurzeig & Lukas, 1972; Milner, 1973; Kull, 1985; Clements & Meredith, 1993) as well as cognitive abilities and achievement (Clements, 1985; Clements, 1987; Clements & Meredith, 1993). However, not all results were positive. Studies found that children exposed to LOGO did not show increased planning skills compared to those not exposed to LOGO and that problem-solving skills needed to be taught to children directly, rather than emerging spontaneously through LOGO usage (Pea, 1983; Clements & Meredith, 1993). Papert himself did not engage in this research and considered that in order to truly understand the impact of LOGO, or computer science education, the educational system needed to be changed so to accommodate for interdisciplinarity, project-based learning, and for supporting children in following their own passion when programming with LOGO. (Papert, 1987).

As time went by, research on LOGO started to slow down. However, its active use in classrooms continued to exist, particularly amongst a strong group of believers in Constructionism, the philosophical approach proposed by Papert (Kafai, 2006; Kafai, 2018). In the 90's, with the growth of the high-tech industry, there was a newfound push for computer science education in K-12 and new programming languages for children started to emerge. During this time, researchers began looking at whether computational thinking may be transferable to other areas of thinking. For example, a meta-analysis of 65 studies revealed that students who participated in computer programming typically score higher on various cognitive-ability assessments than children who did not participate (Liao & Bright, 1991). While most of these studies were not solely focused on early childhood, a series of pilot studies conducted by the DevTech group at Tufts University with preschoolers and kindergarteners, showed that coding can significantly improve young children's sequencing ability, an important pre-math and pre-literacy skill, on a standardized picture sequencing assessment unrelated to programming (Kazakoff & Bers, 2014; Kazakoff, Sullivan, & Bers, 2013).

Despite of the lack of studies with large sample sizes investigating the longitudinal impact of early childhood computer science education, a wealth of resources, technologies and curriculum, and products for young children started to emerge for both formal and informal early childhood educational settings (Stanton et al., 2017). The next section presents an overview and categorizes these programs and approaches into four groups: unplugged computer science, block-based programming languages, programming games, and introductory robotic systems.

Unplugged Computer Science: The push for computational thinking

Computer Science Unplugged has become a powerful movement in recent years because it allows students (and teachers) with little to no technical background to explore computer science concepts

that drive the technologies we use each day, such as thinking recursively, using abstraction when figuring out a complex task, and using heuristic reasoning to discover a solution (Wing, 2006). Unplugged approaches to computer science claim to enable the development of computational thinking, without spending time or cognitive resources on syntax and grammar of programming languages (Bell, Alexander, Freeman, & Grimley, 2009; Bell, Witten, & Fellows, 1998).

The term "computational thinking" grew out of Papert's pioneering research. In his work, it meant both the solving problems algorithmically and the development of technological fluency for personal expression (Bers, 2010; Papert, 1980b). A child who could think like a computer, was a child who could use the computer to express herself in a fluent way. In 2006, Jeannette Wing's influential article "Computational Thinking," (Wing, 2006) caught the attention of researchers, computer scientists, and educators by arguing that *computational thinking*, a problem-solving skill set rooted in computer science, is a universally applicable skill that should be a part of every child's analytical ability (Wing, 2006). Wing defined computational thinking as "solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (p.33). Computational thinking encompasses a broad set of analytic and problem-solving skills, dispositions, habits, and approaches used in computer science (Barr & Stephenson, 2011; I. Lee et al., 2011). This definition limits computational thinking to a problem-solving process that complements mathematical and engineering thinking, hiding the relevance of personal expression in the act of programming (Bers, 2018b).

While computational thinking is rooted in computer science, Wing argued that this kind of thinking can serve everyone as "it represents a universally applicable attitude and skill set" (Wing, 2006, p. 33). However, since technologies are expensive, an approach that would help people think like computer scientists without access to the technology was needed. Low-cost unplugged methods in K-12 make it possible the teaching of ways of thinking associated with computer science without investing in expensive hardware and software that ultimately becomes absolute every few years. However, they might also open a gap between those who can think and code, and those who can only think, because they do not have access to coding tools

The original Computer Science Unplugged project was based at Canterbury University and has since been widely adopted internationally (translated into 12 languages) and it is also recommended in the ACM K-12 curriculum (Bell et al., 2009). CS Unplugged uses activities, games, magic tricks and more to introduce children to ways of thinking about computer science and to engage them in computational thinking without reliance on learning computer programming.

The unplugged approach thinks of computer science as a set of ideas that can be introduced even without using a programming language. Unplugged activities place emphasis on promoting computational thinking, rather than focusing on learning the syntax of a particular coding language. For reasons we will later explore, this approach grew particularly appealing for the early childhood educational segment. For example, an unplugged computer science activity in kindergarten might involve creating bead necklaces in binary with beads that represent 1s and 0s or using a grid and symbols to put classic fairy tales in a logical order (see: www.csunplugged.org), or making a peanut butter sandwich following a set of instructions or algorithm.

A new crop of unplugged games started to be sold commercially. These offered a low-cost way to engage children with computational thinking as compared with traditional technologies. For example, the Robot Turtles board game teaches coding concepts to children ages three and up (see Figure 1) and is the most backed board game in Kickstarter history. Playing the game is easy: you create a maze on the board with the turtles in the corners and the jewels in the center. Kids play instruction cards (such as, turn right, turn left, move forward, etc.) in order to "program" their turtles to get to their jewels. The board can be set up differently each time and as children get more familiar with the cards, more complex instructions can be used. This game engages young children in computational thinking by having them create sequences and problem solve.

Figure 1: Robot Turtles Board Game

The unplugged computer science approach is particularly appealing to early childhood education not only because it is affordable, but also because it promises to expose children to computational thinking while limiting screen-time, in accordance with the American Academy of Pediatrics recommendations (American Academy of Pediatrics, 2003). While computers, tablets, and robots can be quite expensive and often have screens that require children to sit down to manipulate them, materials like beads, crayons, and string are often already present in early childhood classrooms and assessed as developmentally appropriate.

There is some evidence that unplugged CS activities are effective at teaching computational thinking (Rodriguez, Kennicut, Rader, & Camp, 2017). However, although unplugged activities can engage children in computational thinking, they do not expose them to learning programming and the ability to master a new language (Bers, 2018b). A child playing with a board game, might be able to problem solve, but might not understand the possibilities and challenges associated with learning a programming language and using it to create a project to express herself. Languages, both natural and artificial, provide the opportunities to create and inhabit worlds and ultimately, meaning-making. However, those children who have not been exposed early on, might have a harder time. And thus, the risk of a growing digital divide: those who can *think* computationally and those who can *act* computationally (Bers, 2018b). Children in wealthier neighborhoods might go to schools that expose them to coding through tablets, computers, and robots from an early age, and will learn how to develop their own voices and appropriate the tools for creating the artifacts and systems they need; however, those from poorer neighborhoods will not be exposed to these tools and might encounter the new illiteracy of the XXIst century (Bers, 2018b; Herold, 2017; Hohlfeld, Ritzhaupt, Dawson, & Wilson, 2017).

In summary, while unplugged computer science is growing in popularity in the early childhood segment due to its low cost and its affordance to engage children in computational thinking without exposing them to screen-time, this approach might increase the digital divide between those who can and those who can't code, because they never had access to the programming languages.

Block-based programming languages

Programming languages that utilize text are not developmentally appropriate in the early childhood classroom, an educational segment that includes pre-readers and emergent readers. Thus, if a programming language is to be used, it must make programming visual. In contrast to text-based programming languages, block-based programming languages represent instructions as icons or blocks. They introduce coding to young children, and to novices of all ages, by simplifying the syntax of a

programming language and removing (or limiting) text (Kurihara, Sasaki, Wakita, & Hosobe, 2015; Mladenović, Boljat, & Žanko, 2018).

These languages allow users to write code in a similar fashion to connecting puzzle pieces: only pieces that are meant to fit together will fit together- which removes possible syntax errors and frees up cognitive space for problem solving and creativity. Block-based programming languages are growing in early childhood education because they do not require reading and writing. One of the most popular block-based programming languages for young children is ScratchJr, explicitly designed for ages 5-7 (See Figure 2).

ScratchJr is free and was created as a collaboration between the DevTech Research Group at Tufts University, the MIT Lifelong Kindergarten Group, and the Playful Invention Company through generous funding from the National Science Foundation (DRL-1118664 Award) and the Scratch Foundation (Bers & Resnick, 2015). ScratchJr was inspired by the Scratch programming language developed by the MIT LifeLong Kindergarten Group for older children ages 8-16 (Resnick et al., 2009). Over a decade of research with Scratch showed that students can successfully learn foundational computer science concepts (Meerbaum-Salaunt, Armoni, & Ben-Ari, 2013), however as a block-based programming language, Scratch is still too complex to be used by young children (Flannery et al., 2013). Thus, the design of ScratchJr specifically aimed at the developmental needs of emergent young readers (Bers, 2018a, 2018b).

Figure 2: Screenshot of the ScratchJr Interface

In ScratchJr the programming blocks are simplified and organized into six categories represented by different colors: yellow Trigger blocks, blue Motion blocks, purple Looks blocks, green Sound blocks, orange Control flow blocks, and red End blocks. The programming blocks span concepts from simple sequencing of motion to control structures. When put together as a jigsaw puzzle, these programming blocks allow children to control their character's actions on the screen. The result can be as simple or as complex as the user desires, from creating animated collages to complex stories or games. As such, it has been categorized as an "Animation/Game Development" computational tool by researchers Ching, Hsu, & Baldwin (2018). However, at its core, ScratchJr is an introductory programming language.

The target age for ScratchJr is 5 to 7 years old, a time when children are learning how to read and write. In most Western countries, writing happens in a sequence from left to right. Therefore, the

ScratchJr programming script also runs as a sequence from left to right instead of the traditional top-to-bottom format of most programming languages, including Scratch (Bers, 2018b).

As a block-based programming language, ScratchJr supports young children to explore powerful ideas of computer science such as algorithms, debugging, modularization, control structures and the design process in a fun and developmentally appropriate way. ScratchJr introduces computational concepts such as sequencing, loops, events, and operators (Ching, Hsu, & Baldwin, 2018) and engages them in cognitive processes associated with problem solving (Strawhacker & Bers, 2019). However, most importantly, ScratchJr invites children to engage with coding as a literacy of the 21st century and develop the ability to use a symbol system (a language) to comprehend, generate, communicate, and express ideas or thoughts by making a sharable product that others can interpret (Bers, 2018b). As a literacy, coding invites new ways of thinking (i.e. computational thinking) and new ways of producing an artefact detached from its creator (i.e. coding). In the process, both problem-solving and personal expression emerge (Bers, 2019).

In the summer of 2014, ScratchJr was released as a free app. Today the app has over 13 million iOS downloads, is available on iPads, Android tablets, Amazon tablets, and Chromebooks and is used in every country in the world with the exception of North Korea. As of the writing of this paper, the top ten countries of ScratchJr usage are: United States (makes up 33% of ScratchJr users worldwide), United Kingdom (13% of ScratchJr users), Australia (9% of ScratchJr users), Canada (5% of ScratchJr users), Sweden (4% of ScratchJr users), France (3% of ScratchJr users), Spain (3% of ScratchJr users), China (2% of ScratchJr users), South Korea (2% of ScratchJr users), and Japan (2% of ScratchJr users). For an in-depth analysis of ScratchJr usage in Europe, see (Bers, 2018a)

Weekly usage patterns reveal that the highest usage each year occurs in December during Computer Science Education Week, a program dedicated to inspiring students to get involved with Computer Science. Furthermore, ScratchJr usage remains lower when school is typically not in session (Leidl, Bers, & Mihm, 2017).

As a classic example of block-based programming, ScratchJr allows children to drag and drop blocks to create a program for each character they select (See Figure 3). Children snap together motion, sound, looks, and communication blocks to program their own stories and games. Furthermore, children can make their stories with up to four "pages" in each project as to have a beginning, middle, and end. The ScratchJr blocks were created to look like puzzle pieces with specific syntactic goals. Blocks to end a program (e.g. "Repeat Forever") have a rounded edge on the right side while blocks that start a script (e.g. "Start on Green Flag") have a rounded edge on the left side. The middle scripts are cut in a way that they can be snapped together to form a complete and syntactically correct programming script (Portelance, Strawhacker, & Bers, 2015).

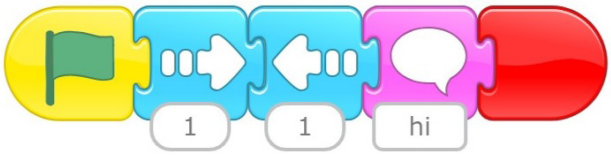

Figure 3: Sample ScratchJr Program

ScratchJr is described as a technological "playground" for young children (Bers, 2012; Bers, 2018b). Like a playground, the environment is open-ended and allows for child-directed exploration and the creation of projects that express the child's unique interests and individuality. ScratchJr users are encouraged to learn by experimenting and by making mistakes, by fixing their bugs and by problem solving.

Like most block-based programming languages, ScratchJr's blocks have visual properties that correspond to their syntactic properties. This aids with preventing syntax errors and enables young users to focus all of their attention in the project they are working on. At the same time, the richness of

ScratchJr as a programming language, although introductory, allows children to both express themselves and encounter powerful ideas from computer science.

Programming games

There is an emerging category of computer games and puzzles aimed at young children's learning of computer science concepts and skills, without the need of exposure to a programming language. Most of these focuses on sequencing and logic as they engage children in progressing through problem solving levels in a typical game-like fashion. For example, the game Lightbot (see Figure 4) is a popular programming puzzle game for young children. The goal is to complete pre-set tasks such as making a robot light up all of the blue tiles on a 3D grid. Children have to program their screen-based robot with a series of instructions. There are different versions of Lightbot for different ages, including Lightbot Jr. for young children ages 4-8.

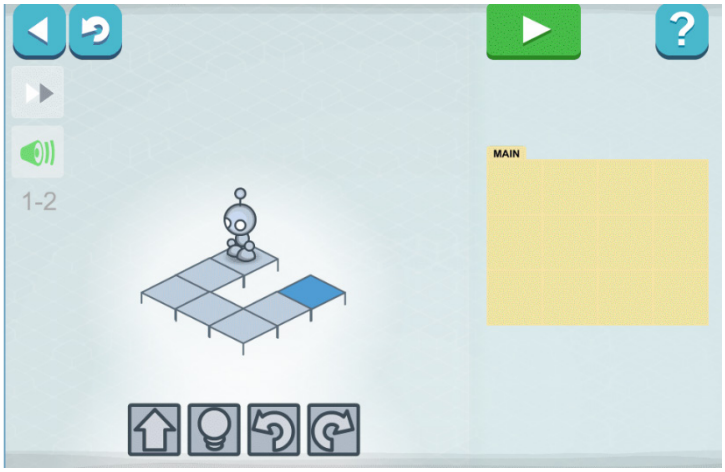

Figure 4: Screenshot of Lightbot (free browser version)

Other popular programming games include Kodable (which includes maze-like levels) and Cargo Bot (which engages young children in learning programming concepts while using a crane to move boxes back and forth between platforms). The website Code.org offers a variety of coding games for children, ranging in age from young children (categorized as "pre-readers") up through high school, as well as Hour of Code activities including Candy Quest (a multi-level coding quest for candy), Code with Anna & Elsa (explore coding with characters from the popular movie *Frozen* by helping them create snowflakes and more), Dragon Blast (embark on a quest for treasure using coding skills), and more. In Code.org's "Classic Maze game" kids write lines of code in a setting inspired by the popular game Angry Birds. In this game, players help Angry Birds get to the Naughty Pigs (see Figure 5). Each level becomes increasingly more difficult to navigate to get to the pigs and focuses on different coding concepts.

Programming games appeal to young children who enjoy video game style play (i.e. specific levels to beat and tasks to complete). Research on students using Code.org's "Classic Maze" activity and the "Flappy Code" activity found that students showed significant changes in their attitudes towards and self-efficacy with computer science after engaging in just one Hour of Code activity (Phillips & Brooks, 2016). However, it is important to note that these games present a more limited set of experiences as compared to block-based programming languages. While block-based languages offer an open-ended setting to create any project of choice, while engaging with powerful ideas from computer science, programming games are typically limiting and prompt players to explore and practice a particular aspect of programming such as cause and effect, sequence, logic and problem solving. Bers (2018b; 2012) uses the metaphor of "playgrounds and playpens" to characterize the differences between programming languages and programming games. Playgrounds are open-ended and invite imagination, creativity, mastering skills and solving conflicts. Programming languages are coding

playgrounds. In contrast, playpens convey lack of freedom to experiment, lack of autonomy for exploration, lack of creative opportunities, and lack of taking risks. When compared to programming languages, coding games are playpens or, in Papert's language, "microworlds", "a subset of reality or a constructed reality whose structures matches that of a given cognitive mechanism… so structured as to allow a human learner to exercise particular powerful ideas or intellectual skills" (Papert, 1980a p. 204).

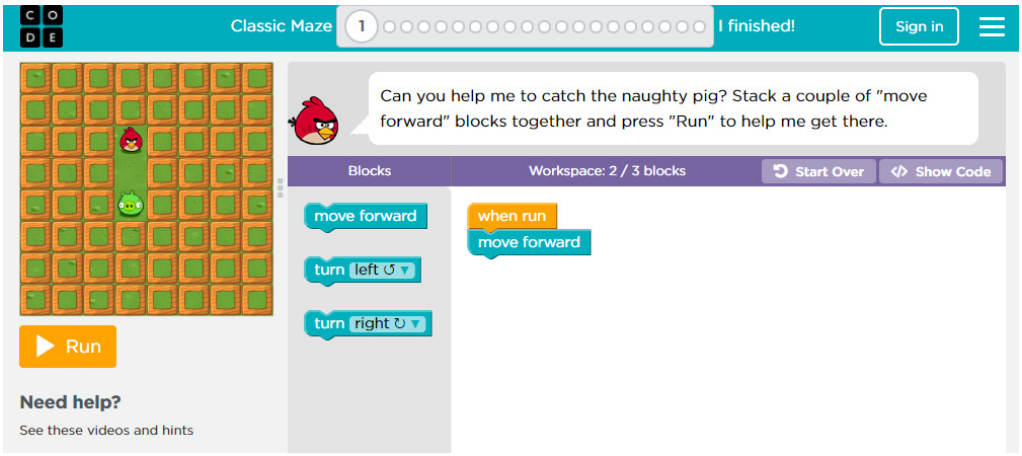

Figure 5: Screenshot of Level 1 of "Classic Maze Game" on Code.org

One of the major differences between programming languages and programming games, is that the former provide tools for learners to produce their own open-ended projects, and not just to play with already developed games. In this process of creation, programmers become producers of personal meaning as well as technical problem solvers.

Introductory robotic systems

Programmable robotics kits are becoming increasingly popular to teach young children the foundations of computer science in a hands-on way. Some robotic systems are programmed using tangible programming languages (Bers & Horn, 2010; Horn, Crouser, & Bers, 2011) and others with block-based programming in screens.

The use of educational robotics is ideal for early childhood because it facilitates cognitive as well as fine motor and social development (Bers, 2008; Clements, 1999; K. T. H. Lee, Sullivan, & Bers, 2013; Svensson, 2000). Young children become engineers by playing with motors and sensors as well as storytellers by creating and sharing personally meaningful projects that react in response to their environment (Bers, 2008, 2018b). Thus, the use of robotic systems in early childhood can expand the range of computer science concepts and skills and include topics related to hardware and software, inputs and outputs.

There are now many examples of introductory robotic systems for young children, some are more similar to playpens and others to playgrounds. For example, Code-a-Pillar (See Figure 6), a robotic caterpillar toy created by Fisher Price, prompts preschool aged children to arrange (and rearrange) easy-to-connect segments (or pieces of code) to decide where Code-a-pillar should move. The Beebot robot is also popular with preschool and early childhood students. The original Beebot, designed to look like a welcoming yellow bee, was programmed to move with the directional keys on its back. A newer version called "Blue-Bot" (see Figure 7) is transparent, allowing children to see and explore the technology inside the robot. Additionally, Blue-Bot is Bluetooth enabled and is compatible with tablets and computers. This allows children to plan algorithms onscreen and send them remotely to the Blue-Bot to perform. A small study on Bee-Bot with 5 to 6-year-olds has found that interventions with the robot can lead to significant improvement in visual-spatial working memory

and inhibition skills (Di Lieto et al., 2017). However, in order to program it, children need to use screens as well as receive help from adults to manipulate the interface.

Figure 6: Code-a-Pillar with body segments that serve as programming instructions

Figure 7: Blue-Bot

The KIBO robotics kit (See Figure 8), developed by the DevTech Research Group at Tufts University and commercially available from KinderLab Robotics, offers young children (4 to 7) an opportunity to explore building and engineering (through assembling their robot) as well as programming (using a tangible block language) without the need of screens or adult assistance. KIBO is a coding playground in which children can create open-ended projects of their choice by engaging in the design process while they build a mobile robot using wheels, motors, lights, and a variety of sensors. KIBO is programmed using interlocking wooden programming blocks. These wooden blocks contain no embedded electronics and are scanned by the KIBO robot. KIBO's design builds on extensive research on tangible programming that uses physical objects to represent the various aspects of computer programming (Horn & Bers, 2019)

KIBO's block programming language is composed of 21 individual wooden programming blocks. Some of these blocks represent simple motions for the robot such as, move Forward, Backward, Spin, and Shake. Other blocks represent complex programming concepts such as Repeat Loops and Conditional "If" statements that involve sensor input. KIBO's design was based on years of research in collaboration with researchers, teachers and early childhood experts to meet the learning needs of young children in a developmentally and fun appropriate way (Kazakoff & Bers, 2014; Sullivan & Bers, 2016; Sullivan, Elkin, & Bers, 2015).

In addition to the tangible programming language, the KIBO robot comes with sensors and actuators (motors and light bulb and microphone/sound recorder), as well as art platforms. These modules can be interchangeably combined on the robot body. The use of sensors, such as light, distance and sound sensor, is well aligned with most early childhood curriculum that engages children in exploring both human and animal sensors. Motors are also included with the robot, two can be connected to the opposite sides, for mobility, and one motor can be located on top, for rotation of an attached element such as the art platform. All of these elements increase the potential of children to create and imagine different projects that can move around and react to the environment.

Figure 8: The KIBO Robot and Wooden Programming Blocks

These four pedagogical approaches and tools for computer science education in the early years (unplugged experiences, block-based programming languages, programming games and introductory robotics) can be integrated, mixed-and-matched, according to the curricular and logistical needs of the schools and the professional development of the teachers. However, regardless of the choice of approach, successful early childhood computer science education programs must have four pedagogical dimensions (see Table 1):

1. **Powerful ideas of computer science taught in a developmentally appropriate way:** Regardless of the interface or approach used, one of the goals of early computer science education is to enable children to encounter, explore and develop "powerful ideas" from the discipline in a developmentally appropriate way. Seymour Papert used the term "powerful idea" to mean a central concept or skill within a disciplinary domain that is personally meaningful and useful as well as epistemologically valid (Papert, 2000). Building on this work, Bers describes seven powerful ideas from computer science that every young child can and should learn, and that are developmentally appropriate: algorithms, modularity, control structures, representation, hardware/software, the design process, and debugging (Bers, 2018b). These powerful ideas connect to many curricular areas or experiential domains beyond computer science. As this paper will later show, these powerful ideas are consistent with the content proposed by most national and state computer science frameworks.
2. **Children's expressiveness by creating meaningful projects:** With a focus on concepts and skills, it is often easy to overlook computer science as an expressive medium. However, coding can be a means of self-expression, akin to writing, speaking, and the arts (Resnick & Siegel, 2015). Early childhood computer science initiatives must provide children with opportunities to use computer science in personally meaningful, creative, and expressive ways. This is consistent with understanding programming languages as one more of the "hundred languages of children" described by the popular Reggio Emilia's early childhood pedagogical approach (Bers, 2008; Edwards, Gandini, & Forman, 1998)
3. **Low-floor/high-ceiling activities for both novices and experts:** The Low-Floor/High-Ceiling approach in computer science education means offering activities that are easy for novices to get started on (low floor) but also challenging for experts who need increasingly complex projects (high ceiling). Teaching materials or curriculum for young children must offer both. In early childhood, there is great variability in cognitive abilities and literacy skills.

Thus, in a same grade, we might find both pre-readers and fluent readers with different cognitive abilities. All of these children need to be engaged and challenged to learn.

4. **Opportunities for debugging and problem solving:** Debugging, both *identifying* errors and completing steps to *fix* these errors is one of the most important skills to emphasize in computer science education (McCauley et al., 2008). However, when working with young children, debugging also involves learning how to manage frustrations, develop perseverance and team-work skills. Thus, when teachers provide opportunities for debugging, they must plan for supporting both cognitive growth and socio-emotional development.

Table 1: Four Dimensions of Computer Science Education in Early Childhood

I. Powerful Ideas CS initiatives support young children to learn powerful ideas and concepts from computer science in developmentally appropriate ways.	**II. Expressiveness** Beyond learning problem-solving strategies and technical content, CS provides young children with a new means of self-expression, just like language and the arts, by programming digital artifacts.
III. Low -Floor/ High Ceiling CS initiatives are easy for novices to get started (low floor) but also allow experts to work on increasingly complex tasks (high ceiling).	**IV. Debugging** CS initiatives prompt children to practice problem-solving skills while engaging in socio emotional development.

As next section will show, these four dimensions of early childhood computer science education can be found in most state and national computer science frameworks. However, they are approached with different levels of depth and sophistication.

COMPUTER SCIENCE FRAMEWORKS IN EARLY CHILDHOOD

Despite the growing popularity of coding, educational policies and frameworks were slow to emerge to advocate for elevating Computer Science to a core subject area in K–12 education. Beginning in the mid-1980s, the Association for Computing Machinery (ACM) established a K–12 "task force" to propose a model curriculum and frameworks. While the U.S. National Council of Teachers of Mathematics (NCTM) was founded in 1920 and the National Science Teachers Association (NSTA) was formed in 1944, the Computer Science Teacher's Association (CSTA) was not launched until 2004 along with the non-partisan computing in the core coalition (a precursor of the current Code.org Advocacy Coalition).

In 2016, the Association for Computing Machinery, Code.org, the Computer Science Teachers Association, the Cyber Innovation Center, and the National Math and Science Initiative collaborated with states, districts, and the computer science education community to develop the K-12 Computer Science Framework (K-12 Computer Science Framework, 2016).

This national framework provides conceptual guidelines for computer science education at each level of the education system and includes both *practices* and *concepts* that should be mastered with a scope and sequence. For example, by the time children finish second grade, the framework recommends they should know introductory concepts or powerful ideas about hardware, software, and algorithms and skills such as debugging, (K-12 Computer Science Framework, 2016).

More specifically, between grades K-2, the framework asks that students are exposed to the following content areas: Computing Systems (i.e. learning about digital devices, hardware, software, and troubleshooting), Networks & the Internet (i.e. learning that computer networks can be used to connect

people to other people), Data & Analysis (i.e. learning that everyday digital devices collect and display data over time and data can be stored and accessed later), Algorithms & Programming (i.e. learning programming concepts of algorithms, variables, control, and modularity), and the Impacts of Computing (i.e. learning about the ways people use computing technology and its positive and negative societal impact) (K-12 Computer Science Framework, 2016). The framework exemplifies how these concepts and skills could be brought to life in the early childhood classroom by using the four different pedagogical approaches and tools described earlier: unplugged activities, block-based programming languages, programming games, and introductory robotic kits.

At the state level, despite the push for STEM education, at the writing of this paper, 25 US states have K-12 computer science standards in place and an additional 10 have them in progress (see Table 2). This means that nearly half (49%) of all U.S. states have or will have standards in place, while 25% still have no standards nor a plan to put them in place at this time (see Table 2). This information is provided by Code.org's and regularly updated (see: http://bit.ly/9policies).

Table 2: States with Computer Science Standards in Progress and In Place (as of Oct 2018)

25 States with K-12 Standards in Place	10 States with K-12 Standards in Progress
AL, AR, AZ, CA, CT, DE, FL, HI, IA, ID, MA, MD, MS, NH, NJ, NV, OK, PA, RI, SC, UT, VA, WA, WI, WV	AK, GA, KS, KY, MI, MO, MT, ND, OH, WY

Most of the states with computer science standards at the early childhood level (K-2) only address three of the four dimensions identified earlier: 1) explicit references to powerful ideas of computer science that need to be taught in a developmentally appropriate way; 2) a "low-floor/high-ceiling" approach with a scope and sequence that starts with basic familiarity with technology to engage beginners, and continues with more complex coding activities and 3) opportunities to engage children in debugging and problem solving. With some exceptions, the expressiveness dimension (using programming as a means to create personally meaningful computational projects) is mostly lacking.

Examples of those exceptions are the Virginia CS standards that make explicit that "the process of developing computational artifacts embraces both creative expression and the exploration of ideas to create prototypes and solve computational problems" (Board of Education Commonwealth of Virginia, 2017, p. 6), the Massachusetts Digital Literacy and Computer Science Framework, that describes students engaging in "evaluating various digital tools for best expression of a particular idea or set of information" and "Selecting and using digital media and tools to communicate effectively (Massachusetts Department of Elementary and Secondary Education, 2016, p. 12) and the Nevada CS Standards that list personal expression as a practice that should be fostered through CS, stating that students should be able to "Create a computational artifact for practical intent, personal expression, or to address a societal issue." (Nevada Department of Education, 2018, p.12).

While an analysis of the differences and similarities between each of the state standards is beyond the scope of this paper, the next section explores the question whether states with computer science standards are more likely than others to see an increase in the teaching of computer science. This question is explored through the lens of a case study on ScratchJr's usage, the most popular introductory programming language explicitly designed for early childhood education.

The Case of ScratchJr

This section explores whether the presence of computer science state frameworks results in stronger usage of ScratchJr. The choice of ScratchJr, for building this case study, is due to three different factors. First, it is the most popular programming language for early childhood education. Second, it is

free. Third, it encompasses all four dimensions previously described: exposes children to powerful ideas from computer science in developmentally appropriate ways (e.g. it was specifically designed for children ages 5-7), provides a means for self-expression (e.g. allows children to create stories and projects in an open-ended way), prompts debugging and problem solving (e.g. prompts problem-solving with complex concepts such as loops and control flow), and offers a low-floor/high-ceiling interface (e.g. children can get started easily just making one character move and go on to create complex multi-character, multi-page projects) . This makes ScratchJr an ideal candidate for a case study.

Since 2015, the ScratchJr team has been collecting non-identifying data from Google Analytics to examine patterns of ScratchJr usage. Google Analytics is a free tool that allows access to user activity as it happens in real time on the app, as well as audience demographics and behavior.

As of September 2018, 25 million ScratchJr projects have been created with over 200,000 active users each week. Over 950,000 ScratchJr projects have been shared with others via email or Apple AirDrop. To research the impact of state standards on children's ScratchJr usage, we examined state level computer science policies in conjunction with ScratchJr Google Analytics data. Figure 9 provides an overview of ScratchJr users per state. As shown, states with large populations (i.e. California and Texas) have the largest number of users.

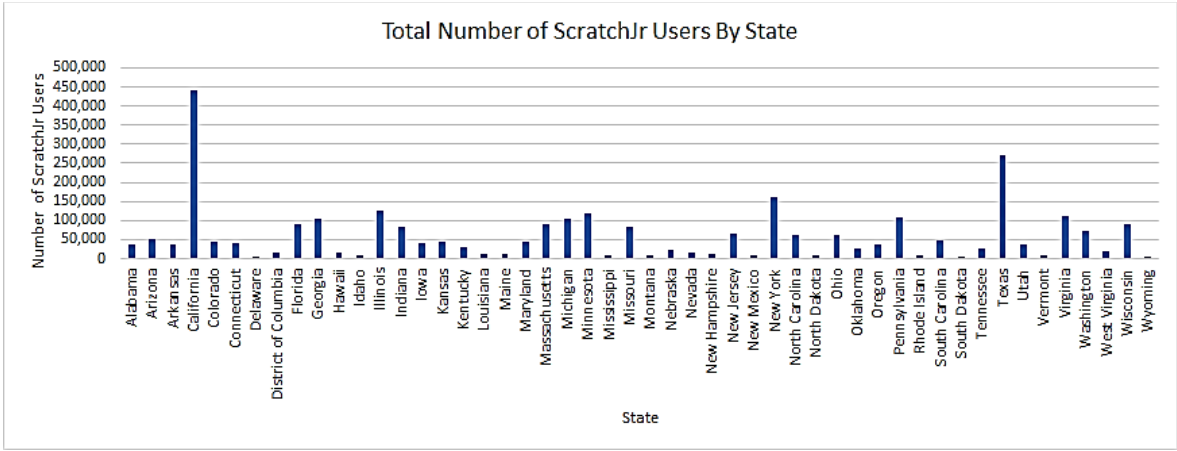

Figure 9: ScratchJr Users by State

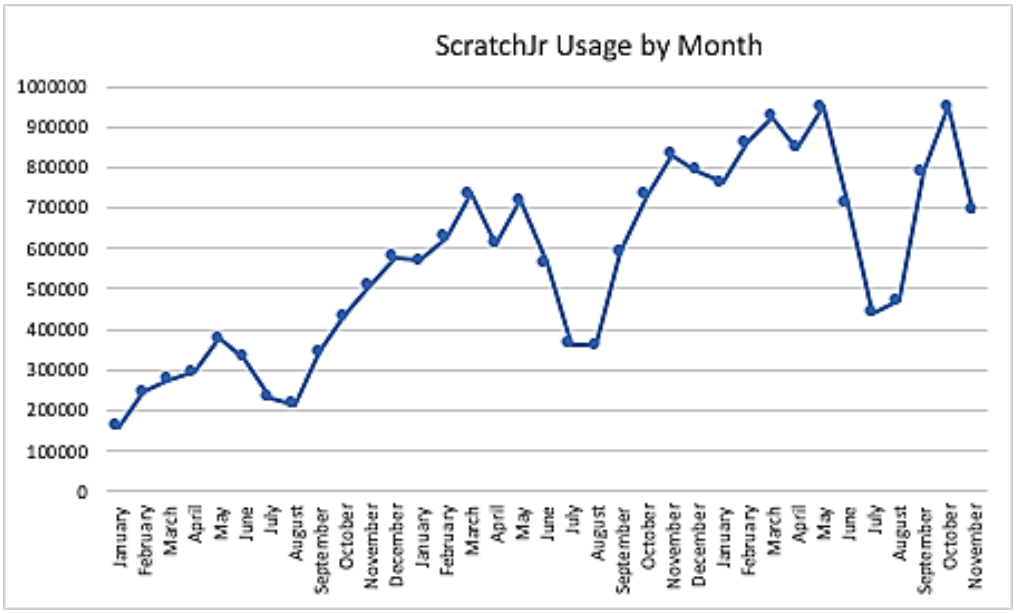

Figure 10: ScratchJr Usage by Month

However, when considering percent of state population (rather than just raw number of users), we see that the states with the largest percent of their population using the app are Washington DC, Minnesota, and Vermont. Looking at ScratchJr usage by month (Figure 10) and by day (Figure 11) data from analytics suggest that children are using ScratchJr more frequently at school than at home. Data also shows that usage of the app decreases in summer months when school is not in session and that ScratchJr is used more on weekdays than on weekends. Combined, these data points to the fact that ScratchJr is used in formal school settings more than in informal ones. These formal settings, schools, are guided in their choice of introducing CS by the state frameworks.

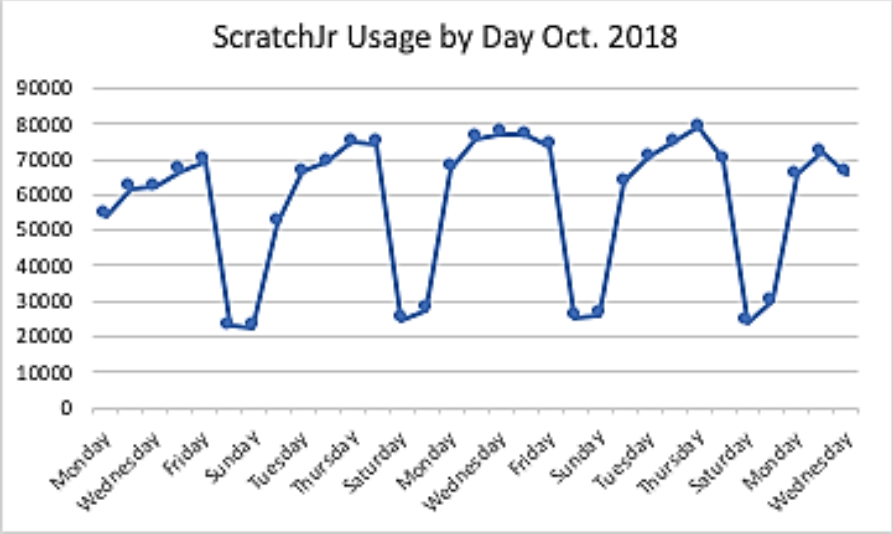

Figure 11: ScratchJr Usage by Day of the Week from October 2018

In order to gain insight on the typical ScratchJr experience, analytics were examined to find the most popular events to occur within the ScratchJr application (see Table 3) and the most popular blocks used (see Table 4). The two most popular events within the app were adding new characters and adding new programming blocks to a project. This was more popular than, for example, using the Paint Editor. This indicates that ScratchJr is most often used for its intended purpose (i.e. coding and computer science education) and that the artistic features like the Paint Editor are used in support of coding.

When it came the most popular programming blocks used, we can see that basic motion blocks (i.e. forward, up, etc.) were most commonly used. The "Start of Flag" trigger block was also commonly used. This indicates that children are experimenting with creating complete lines of code that include a trigger block, rather than just dragging in motion blocks and tapping them to see characters move. The use of simple blocks supports the low-floor idea so that children can easily get started with ScratchJr. More complicated blocks, such as the repeat loops, were still used but not as commonly, supporting the "high ceiling" design principle. This makes sense, since most users of ScratchJr are novices, not experts.

Table 3: Five Most Common ScratchJr Events

Event	Number of Times Occurred
New Block Added	179,066,278
New Character Added	44,091,507
Paint Editor Opened	34,106,661
Existing Project Edited	11,843,159
New Project Created	9,998,054

Table 4: Most Commonly Used ScratchJr Programming Blocks

Programming Block	Number of Times Used
Forward	18,220,868
Start on Flag	10,095,428
Up	7,351,636
Back	6,291,630

Comparing Analytics from States with vs. without CS Standards

When comparing states with and without Computer Science standards, analytics show that states with standards in place have a higher average number of ScratchJr users compared to those without standards or with standards still in progress (See Figure 12). States with Computer Science standards also have a higher average number of total ScratchJr sessions to date (Figure 13). A session is defined as the period time a user is actively engaged with the app. Although this data doesn't show causality, it provides preliminary evidence that states with Computer Science Standards have more ScratchJr users and more active ScratchJr sessions.

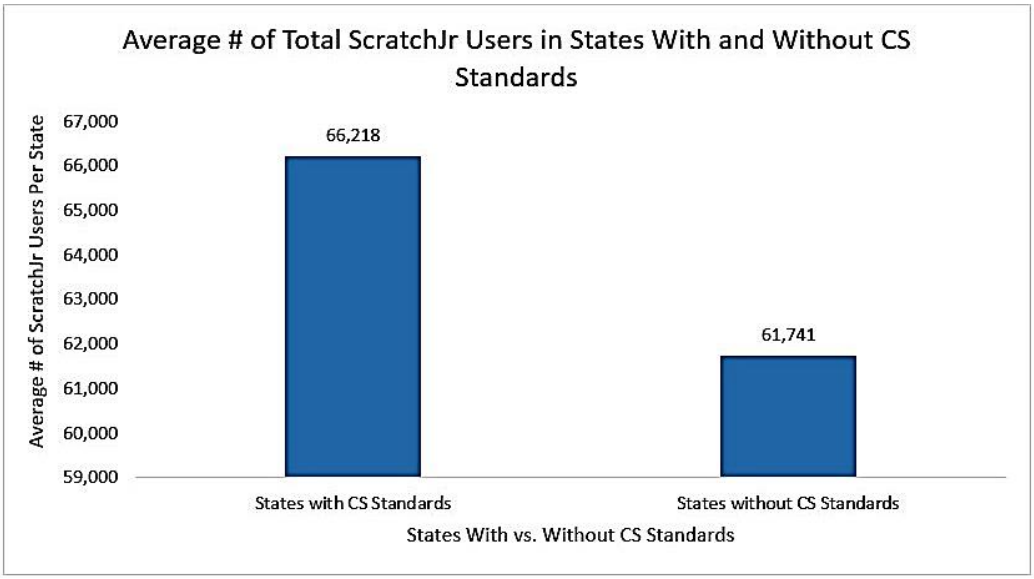

Figure 12: Average Number of ScratchJr Users in States with vs. without K-12 Computer Science Standards

Analytics were also examined to find preliminary evidence of debugging within the ScratchJr app in states with and without CS standards. While it is not possible to directly ascertain whether children are engaging in debugging, we can look at two elements that may hint at debugging: length of time spent on a project and number of times users edit an existing project (rather than simply creating a new project). Figure 14 shows the average number of times ScratchJr projects were edited since 2015 in states with versus without CS standards. Here we see that states with standards have users editing projects an average of 4,605 times more than in states without standards. Looking at session duration (Figure 15) we also see that states with CS standards show a slightly longer session duration than those without standards (almost a minute longer). Taken together, these two findings provide preliminary evidence that states with CS standards may promote children's debugging in ScratchJr and support their perseverance to continue returning to and improving existing projects.

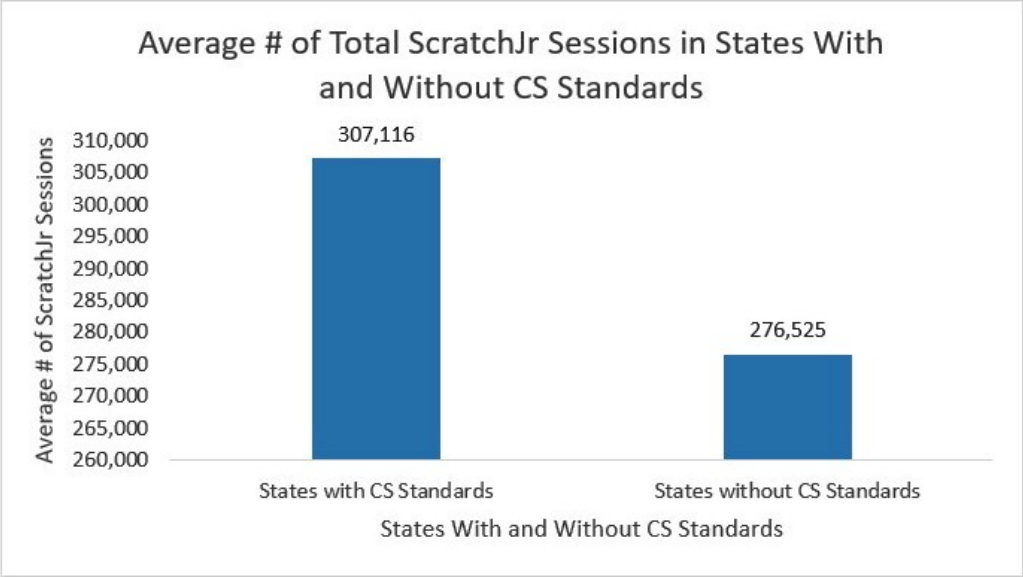

Figure 13: Average Number of ScratchJr Sessions in States with vs. without K-12 Computer Science Standards

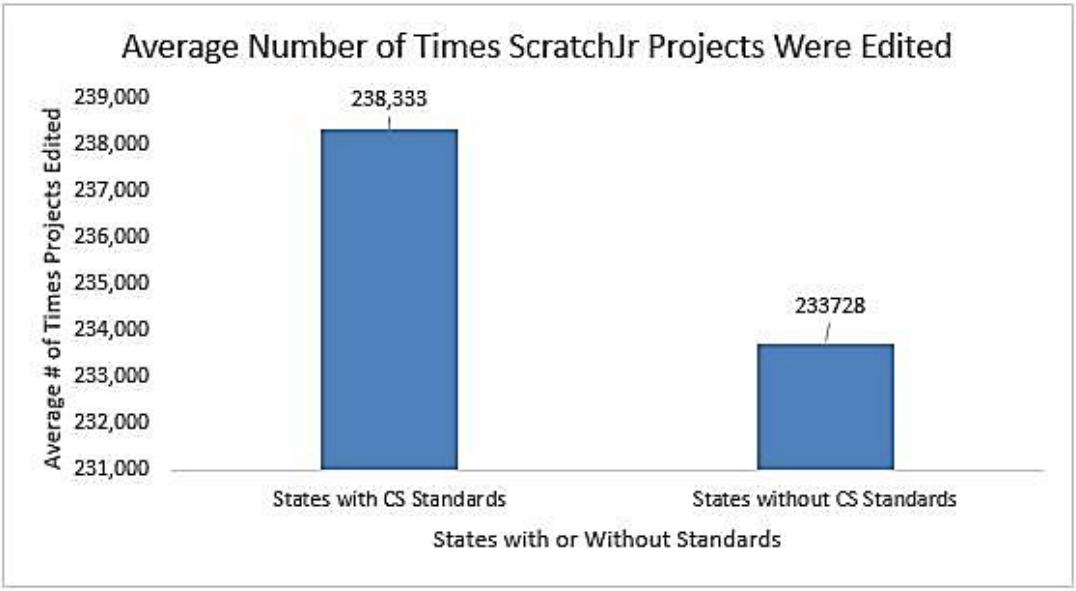

Figure 14: Average Time Projects Were Edited in States with versus without CS Standards

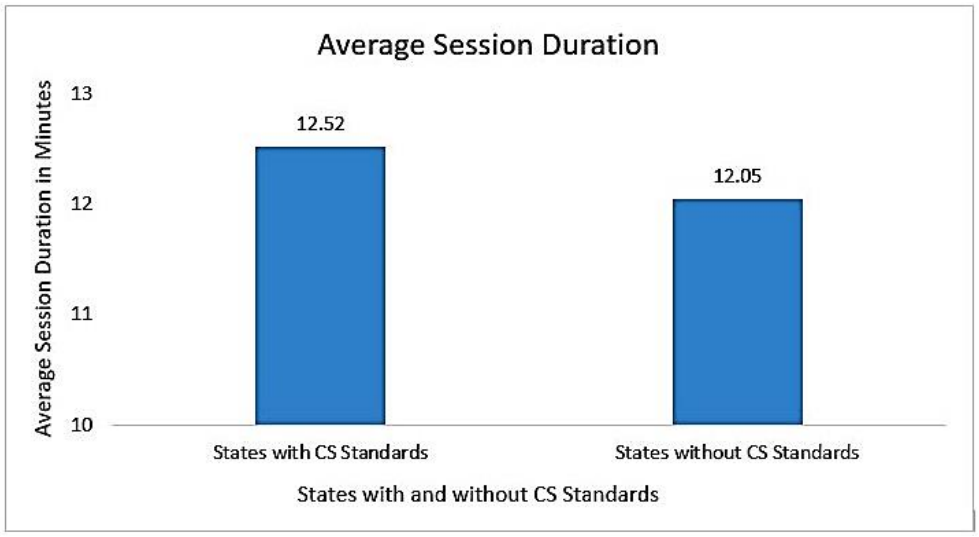

Figure 15: Average Session Duration in States with versus without CS Standards

Analytics were also collected on how children used the Paint Editor function in the ScratchJr app. Here we see that states with CS standards in place use the Paint Editor more often on average than those without (see Figure 16). Use of the Paint Editor may be related to using the application for creative and expressive purposes, as it allows children to personalize characters and backgrounds (or create their own). Therefore, this finding may indicate that states with CS standards are more actively supporting students' expressiveness through coding. On the flipside, the Paint Editor is an entertaining part of the ScratchJr app that *supports* a user's programming project but does not require any programming in and of itself. Therefore, use of the Paint Editor may also indicate an avoidance of programming. Follow up studies need to be conducted to understand what is the case.

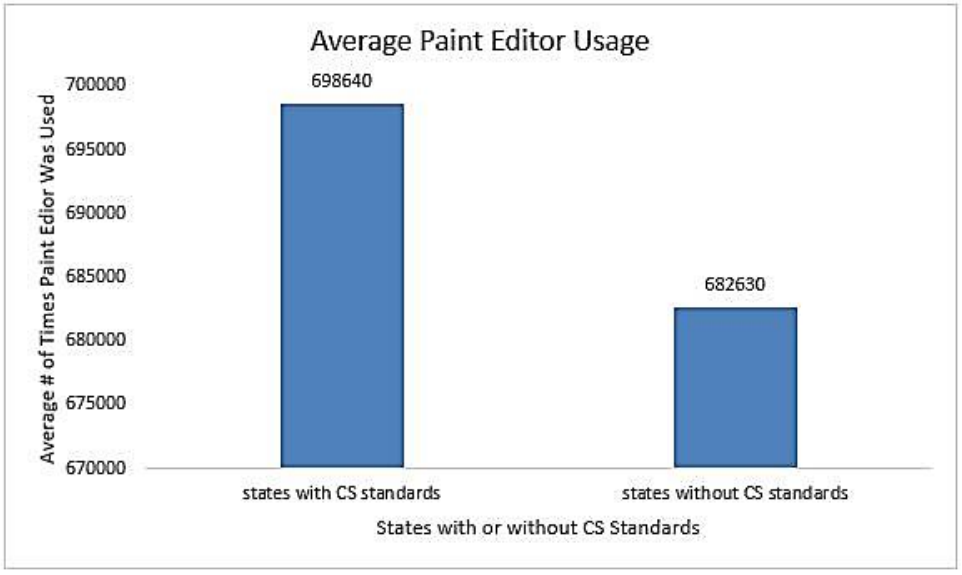

Figure 16: Average Paint Editor Usage in States with and without CS Standards

DISCUSSION & CONCLUSION

This paper explores the emergent field of early childhood computer science education by first categorizing what is currently available into four different pedagogical approaches and tools: unplugged activities, block-based programming languages, programming games, and introductory robotic kits. Each of these addresses a particular need and provides a unique learning experience, thus they can be integrated, mixed-and-matched, according to the curricular and logistical needs of the schools and the professional development of the teachers. However, regardless of the choice of approach, successful early childhood computer science education programs must teach powerful ideas from the discipline of computer science in a developmentally appropriate way, provide means for self-expression, prompt debugging and problem solving, and offer a low-floor/high-ceiling interface for both novices and experts.

By using ScratchJr, the most popular block programming language explicitly designed for young children, the paper explored whether the presence of computer science state frameworks results in stronger usage patterns. Results from these analytics demonstrate the importance of having state standards in place to increase young children's exposure to coding and powerful ideas from computer science in the early years. In the case of ScratchJr, analytics show that app usage decreases during the summer months and on weekends, which indicates that coding with ScratchJr is more often happening in school than at home. Results also show that states with CS standards have more ScratchJr users on average and also have more total sessions with the app on average. Additionally, the results show that states with CS standards in place use the Paint Editor more often than those without, which may indicate that states with CS standards are more actively supporting students' expressiveness or artistic exploration along with coding. Finally, the results also show that states with CS standards have users editing projects more often more than in states without standards and that states with CS standards show a slightly longer average session duration than those without standards (almost a minute longer). Taken together, these two findings provide preliminary evidence that CS standards may promote children's debugging and perseverance with coding applications like ScratchJr. Overall, the findings from this study provide initial evidence of the possible positive impact of state standards on coding in early childhood

LIMITATIONS AND FUTURE RESEARCH

This case-study of ScratchJr provides a preliminary look at the impact of state computer science frameworks on supporting young children's exploration of computational thinking and the development of computer science skills. As a pilot case study, this paper relies on descriptive statistics to describe trends, and it is important to note that no statistical significance claims are made. Furthermore, this study was limited by the data that could be collected through Google Analytics, while maintaining users' anonymity and privacy. Therefore, deeper analyses making comparisons by gender, age of users, etc. was not feasible to collect within the confines of this study. For example, although ScratchJr is designed for children ages 5-7, there is no way of knowing from this data whether the users in this study were actually young children or not. Future research should pair Google Analytics with surveys, classroom assessments, and/or observational studies in order to learn more about the impact of state standards on students' mastery of concepts and interest in coding, as well as the impact of different demographic characteristics.

Additionally, this study focused on analyzing different design approaches to teaching computer science and computational thinking in early childhood. However, it was beyond the scope of this study to look at the role of the instructor in states with (or without) computer science standards in place. While technologies are important, the role of the educator and the curriculum is also an important piece to effectively teaching computer science in the early years. K-12 teachers and researchers have not clearly identified best practices for teaching computational thinking as of yet (Hsu, Chang, & Hung, 2018). Future research should examine the role of the teacher in states with and without com-

puter science standards and determine how to better support them as new frameworks are being put into place.

Finally, the case-study nature of this paper meant it was focused explicitly on one technology: ScratchJr. Future research should look at the relationship between state standards on children's usage of other computational tools and technologies in order to more fully understand the impact of the standards.

CONCLUSION

This study demonstrates preliminary evidence that states with Computer Science standards in place support skills like perseverance and debugging through ScratchJr. The analytics show that states with Computer Science standards have longer average session duration as well as a higher average number of users returning to edit an existing project. Finally, the analytics also demonstrate that usage of the ScratchJr Paint Editor is more often used in states with Computer Science standards than those without. This may suggest that these states support expression and creativity through coding projects more than those without standards. However, this cannot be stated as a fact and more research needs to be done. As the US, amongst other nations, is moving forward with policy decisions regarding how to introduce computer science in the early childhood curriculum, this paper can inform the many different choices available.

REFERENCES

Abelson, H., & diSessa, A. (1982). *Turtle geometry*. Cambridge, Massachusetts: MIT Press. Retrieved from https://mitpress.mit.edu/books/turtle-geometry

American Academy of Pediatrics (2003). Prevention of pediatric overweight and obesity: Policy statement. *Pediatrics*, *112*(2), 424–430. https://doi.org/10.1542/peds.112.2.424

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is Involved and what is the role of the computer science education community? *Inroads*, *2*(1), 48-54. https://doi.org/10.1145/1929887.1929905

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer Science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, *13*(1), 20-29. Retrieved from https://pdfs.semanticscholar.org/a5e8/fdef0bfb5b41138fb79e611781cfb7a0b305.pdf

Bell, T. C., Witten, I. H., & Fellows, M. (1998). *Computer Science unplugged: Off-line activities and games for all ages*. Computer Science Unplugged. Retrieved from https://classic.csunplugged.org/wp-content/uploads/2015/01/unplugged-book-v1.pdf

Bers, M. U. (2008). *Blocks to robots: Learning with technology in the early childhood classroom*. New York, NY: Teachers College Press. https://doi.org/10.1093/acprof:oso/9780199757022.001.0001

Bers, M. U. (2010). The TangibleK robotics program: Applied computational thinking for young children. *Early Childhood Research and Practice*, *12*(2), Online. Retrieved from http://ecrp.uiuc.edu/v12n2/bers.html/

Bers, M. U. (2012). *Designing digital experiences for positive youth development: From playpen to playground*. Cary, NC: Oxford.

Bers, M. U. (2018a). Coding and computational thinking in early childhood: The impact of ScratchJr in Europe. *European Journal of STEM Education*, *3*(3), 08. https://doi.org/10.20897/ejsteme/3868

Bers, M. U. (2018b). *Coding as a playground: Programming and computational thinking in the early childhood classroom*. New York, NY: Routledge. https://doi.org/10.4324/9781315398945

Bers, M. U. (2019). Coding as another language: Why Computer Science in early childhood should not be STEM. In C. Donohue (Ed.). *Exploring key issues in early childhood and technology: Evolving perspectives and innovative approaches* (Chapter 10). New York, NY: Routledge. https://doi.org/10.4324/9780429457425-11

Bers, M. U., & Horn, M. (2010) Tangible programming in early childhood: Revisiting developmental assumptions through new technologies. In M. J. Berson, & I. R. Berson (Eds.). *High-tech Tots: Childhood in a digital world*. Greenwich, CT: Information Age Publishing. Retrieved from https://ase.tufts.edu/Devtech/publications/Bers-Horn_May1809.pdf

Bers, M. U. & Resnick, M. (2015). *The official ScratchJr book*. San Francisco, CA: No Starch Press.

Blikstein, P. (2018). *Pre-college Computer Science education: A survey of the field*. Mountain View, CA: Google LLC. Retrieved from https://services.google.com/fh/files/misc/pre-college-computer-science-education-report.pdf

Board of Education Commonwealth of Virginia (2017). *Computer Science standards of learning for Virginia public schools*. Richmond, VA: Commonwealth of Virginia. Retrieved from http://www.doe.virginia.gov/testing/sol/standards_docs/computer-science/2017/stds-compsci-all.pdf

Bowman, B. T., Donovan, M. S., & Burns, M. S. (2001). *Eager to learn: Educating our preschoolers*. National Research Council Report. Washington, DC: National Academies Press. Retrieved from https://files.eric.ed.gov/fulltext/ED447963.pdf

Ching, Y. H., Hsu, Y. C., & Baldwin, S. (2018). Developing computational thinking with educational technologies for young learners. *Tech Trends*, *62*(6), 563–573. https://doi.org/10.1007/s11528-018-0292-7

Clements, D. H. (1985). Differential effects of computer programming (Logo) and computer assisted instruction (CAI) on young children's executive processes and cognitive development [Summary]. In *Proceedings of the 51st Biennial Meeting of the Society for Research in Child Development, Volume 5* (p. 59).

Clements, D. H. (1987). Longitudinal study of the effects of Logo programming on cognitive abilities and achievement. *Journal of Educational Computing Research*, *3*(1), 73–94. https://doi.org/10.2190/RCNV-2HYF-60CM-K7K7

Clements, D. H. (1999). The future of educational computing research: The case of computer programming. In *Information Technology in Childhood Education Annual* (pp. 147-179). Retrieved from https://www.learntechlib.org/p/10815/article_10815.pdf

Clements, D. H., Battista, M. T., & Sarama, J. (2001). Logo and Geometry. E. Yackel (Ed.). *Journal for Research in Mathematics Education Monograph Series, Volume 10*. https://doi.org/10.2307/749924

Clements, D. H., & Meredith, J. S. (1993). Research on Logo: Effects and efficacy. *Journal of Computing in Childhood Education*, *4*(4), 263-290. Retrieved from https://el.media.mit.edu/logo-foundation/resources/papers/pdf/research_logo.pdf

Clements, D. H., & Sarama, J. (1996). Turtle Math: Redesigning Logo for elementary Mathematics. *Learning and Leading with Technology*, *23*(7), 10-15.

Code.Org (2018). Retrieved from https://code.org/

Cunha, F., & Heckman, J. (2007). The technology of skill formation. *American Economic Review*, *97*(2), 31-47. https://doi.org/10.1257/aer.97.2.31

Corbett, C., & Hill, C. (2015). *Solving the equation: The variables for women's success in engineering and computing*. Washington, DC: The American Association of University Women. Retrieved from https://files.eric.ed.gov/fulltext/ED580805.pdf

Di Lieto, M. C., Inguaggiato, E., Castro, E., Cecchi, F., Cioni, G., Dell'Omo, M., Laschi, C., Pecini, C., Santerini, G., Sgandurra, G., & Dario, P. (2017). Educational robotics intervention on executive functions in preschool children: A pilot study. *Computers in Human Behavior*, *71*, 16-23. https://doi.org/10.1016/j.chb.2017.01.018

Edwards, C. P., Gandini, L., & Forman, G. E. (1998). *The hundred languages of children: The Reggio Emilia approach – Advanced reflections*. Westport, Connecticut: Greenwood Publishing Group.

Fayer, S., Lacey, A., & Watson, A. (2017). *BLS spotlight on statistics: STEM occupations – Past, present, and future*. Washington, DC: U. S. Department of Labor, Bureau of Labor Statistics. Retrieved from https://pdfs.semanticscholar.org/638a/854617c4dd1a177d02c4eb59584d0acc2919.pdf

Feurzeig, W., & Lukas, G. (1972). LOGO – A programming language for teaching Mathematics. *Educational Technology*, *12*(3), 39-46. Retrieved from https://www.jstor.org/stable/44417811

Flannery, L. P., Kazakoff, E. R., Bontá, P., Silverman, B., Bers, M. U., & 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 (IDC '13)* (pp. 1-10). New York, NY: ACM. https://doi.org/10.1145/2485760.2485785

Guzdial, M. (2003). Programming environments for novices. In *Computer Science Education Research* (pp. 127-154). Retrieved from http://coweb.cc.gatech.edu/mediaComp-plan/uploads/37/novice-envs2.pdf

Heckman, J. J., & Masterov, D. V. (2007). The productivity argument for investing in young children. *Applied Economic Perspectives and Policy*, *29*(3), 446-493. https://doi.org/10.1111/j.1467-9353.2007.00359.x

Herold, B. (2017, June 12). Poor students face digital divide in how teachers learn to use tech. *Education Week*. Retrieved from: https://www.edweek.org/ew/articles/2017/06/14/poor-students-face-digital-divide-in-teacher-technology-training.html

Hohlfeld, T. N., Ritzhaupt, A. D., Dawson, K., & Wilson, M. L. (2017). An examination of seven years of technology integration in Florida schools: Through the lens of the levels of Digital Divide in schools. *Computers & Education*, *113*, 135-161. https://doi.org/10.1016/j.compedu.2017.05.017

Horn, M., & Bers, M. U. (2019). Tangible computing. In S. A. Fincher, & A.V. Robins (Eds.). *The Cambridge handbook of computing education research* (pp. 663-678). Cambridge, UK: Cambridge University Press. https://doi.org/10.1017/9781108654555.023

Horn, M., Crouser, R., & Bers, M. U. (2011). Tangible interaction and learning: The case for a hybrid approach. *Personal and Ubiquitous Computing*, *16*(4), 379–389. Special Issue on Tangibles and Children. https://doi.org/10.1007/s00779-011-0404-2

Hsu, T. C., Chang, S. C., & Hung, Y. T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education*, *126*, 296-310. https://doi.org/10.1016/j.compedu.2018.07.004

K-12 Computer Science Framework Steering Committee (2016). *K-12 Computer Science Framework*. Retrieved from https://k12cs.org/

Kafai, Y. B. (2006). Playing and making games for learning: Instructionist and constructionist perspectives for game studies. *Games and Culture*, *1*(1), 36-40. https://doi.org/10.1177/1555412005281767

Kafai, Y. B. (2018). Constructionist visions: Hard fun with serious games. *International Journal of Child-Computer Interaction*, *18*, 19-21. https://doi.org/10.1016/j.ijcci.2018.04.002

Kazakoff, E. R., & Bers, M. U. (2014). Put your robot in, put your robot out: Sequencing through programming robots in early childhood. *Journal of Educational Computing Research*, *50*(4), 553-573. https://doi.org/10.2190/EC.50.4.f

Kazakoff, E. R, Sullivan, A., & Bers, M. U. (2013). The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education Journal*, *41*(4), 245-255. https://doi.org/10.1007/s10643-012-0554-5

Kull, J. A. (1985, April). Programming, problem-solving, and mathematical learning in young children learning Logo: A collaborative, qualitative study in the first grade. Paper presented at the *Annual Meeting of the American Educational Research Association (AERA)*. Chicago, IL: American Educational Research Association.

Kurihara, A., Sasaki, A., Wakita, K., & Hosobe, H. (2015). A programming environment for visual block-based domain-specific languages. *Procedia Computer Science*, *62*, 287-296. https://doi.org/10.1016/j.procs.2015.08.452

Lee, K. T. H., Sullivan, A., & Bers, M. U. (2013). Collaboration by design: Using robotics to foster social interaction in Kindergarten. *Computers in the Schools*, *30*(3), 271-281. https://doi.org/10.1080/07380569.2013.805676

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J; & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, *2*(1), 32-37. https://doi.org/10.1145/1929887.1929902

Leidl, K., Bers, M. U., & Mihm, C. (2017). Programming with ScratchJr: A review of the first year of user analytics. In *Proceedings of International Conference on Computational Thinking Education* (pp. 116-121). Retrieved from https://ase.tufts.edu/devtech/publications/Leidl_Bers_Mihm_ScratchJrAnalyticsHongKong.pdf

Liao, Y. K. C., & Bright, G. W. (1991). Effects of computer programming on cognitive outcomes: A meta-analysis. *Journal of Educational Computing Research*, *7*(3), 251-268. https://doi.org/10.2190/E53G-HH8K-AJRR-K69M

The Logo Foundation (2015). *Logo History*. Retrieved from: http://el.media.mit.edu/logo-foundation/what_is_logo/history.html

Madill, H. M., Campbell, R. G., Cullen, D. M., Armour, M. A., Einsiedel, A. A., Ciccocioppo, A. L., & Coffin, W. L. (2007). Developing career commitment in STEM-related fields: Myth versus reality. In R. J. Burke, M. C. Mattis, & E. Elgar (Eds.). *Women and minorities in science, technology, engineering and mathematics: Upping the numbers* (pp. 210-244). Northampton, MA: Edward Elgar Publishing. https://doi.org/10.4337/9781847206879.00019

Markert, L. R. (1996). Gender related to success in science and technology. *The Journal of Technology Studies*, *22*(2), 21-29. https://doi.org/10.21061/jots.v22i2.a.4

Massachusetts Department of Elementary and Secondary Education (2016). *Digital literacy and Computer Science curriculum framework: Grades Kindergarten to 12*. Retrieved from http://www.doe.mass.edu/frameworks/dlcs.pdf

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education*, *18*(2), 67-92. https://doi.org/10.1080/08993400802114581

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning Computer Science concepts with scratch. *Computer Science Education*, *23*(3), 239-264. https://doi.org/10.1080/08993408.2013.832022

Metz, S. S. (2007). Attracting the engineers of 2020 today. In R. J. Burke, M. C. Mattis, & E. Elgar (Eds.). *Women and minorities in science, technology, engineering and mathematics: Upping the numbers* (pp. 184–209). Northampton, MA: Edward Elgar Publishing. https://doi.org/10.4337/9781847206879.00018

Milner, S. (1973, February). The effects of computer programming on performance in Mathematics. Paper presented at the *Annual Meeting of the American Educational Research Association (AERA)*. New Orleans, Louisiana: American Educational Research Association. Retrieved from https://files.eric.ed.gov/fulltext/ED076391.pdf

Mladenović, M., Boljat, I., & Žanko, Ž. (2018). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Education and Information Technologies*, *23*(4), 1483-1500. https://doi.org/10.1007/s10639-017-9673-3

Nevada Department of Education (2018). *2018 Nevada K-12 Computer Science standards*. Draft Version. Retrieved from http://www.doe.nv.gov/uploadedFiles/nde.doe.nv.gov/content/Standards_Instructional_Support/Nevada_Academic_Standards/Comp_Tech_Standards/DRAFTNevadaK-12ComputerScienceStandards.pdf

Papert, S. (1980a). Computer-based micro-worlds as incubators for powerful ideas. In R. Taylor (Ed.). *The computer in the school: Tutor, tool, tutee* (pp. 203-210). Totowa, NJ: Teacher's College Press. Retrieved from https://eric.ed.gov/?id=ED207670

Papert, S. (1980b). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books, Inc. Retrieved from http://worrydream.com/refs/Papert%20-%20Mindstorms%201st%20ed.pdf

Papert, S. (1987). Computer criticism vs. techno-centric thinking. *Educational Researcher*, *16*(1), 22-30. https://doi.org/10.2307/1174251

Papert, S. (2000). What's the big idea? Toward a pedagogy of idea power. *IBM Systems Journal*, *39*(3.4), 720-729. IBM. https://doi.org/10.1147/sj.393.0720

Pea, R. D. (1983). *Logo programming and problem solving*. Technical Report No. 12. New York, NY: Center for Children and Technology. Retrieved from https://eric.ed.gov/?id=ED319371

Phillips, R., & Brooks, B. (2016). *The hour of code: Impact on attitudes towards and self-efficacy with Computer Science*. Code.Org. Retrieved from https://code.org/files/HourOfCodeImpactStudy_Jan2017.pdf

Portelance, D. J., Strawhacker, A. L., & Bers, M. U. (2015). Constructing the ScratchJr programming language in the early childhood classroom. *International Journal of Technology and Design Education*, *26*(4), 489-504. Retrieved from https://ase.tufts.edu/devtech/publications/Portelance-2015-Constructing-ScratchJr.pdf

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60-67. https://doi.org/10.1145/1592761.1592779

Resnick, M., & Siegel, D. (2015). A different approach to coding. *Bright Magazine*. Retrieved from https://brightthemag.com/a-different-approach-to-coding-d679b06d83a

Rodriguez, B., Kennicutt, S., Rader, C., & Camp, T. (2017, March). Assessing computational thinking in CS unplugged activities. In *Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 501-506). New York, NY: ACM. https://doi.org/10.1145/3017680.3017779

Shonkoff, J. P., & Phillips, D. A. (2000). *From neurons to neighborhoods: The science of early childhood development*. Washington, DC: National Academy Press.

Stanton, J., Goldsmith, L., Adrion, R., Dunton, S., Hendrickson, K., Peterfreund, A., Yongpradit, P., Zarch, R., & Zinth, D. J. (2017). *State of the States landscape report: State-level policies supporting equitable K–12 Computer Science education.* Education Development Center. Retrieved from https://www.edc.org/state-states-landscape-report-state-level-policies-supporting-equitable-k-12-computer-science

Steele, C. M. (1997). A threat in the air: How stereotypes shape intellectual identity and performance. *American Psychologist*, *52*(6), 613–629. https://doi.org/10.1037/0003-066X.52.6.613

Strawhacker, A., & Bers, M. U. (2019). What they learn when they learn coding: Investigating cognitive development and computer programming in young children. *Educational Technology Research and Development, 67*(3), 541-575. https://doi.org/10.1007/s11423-018-9622-x

Sullivan, A. (2019). *Breaking the STEM stereotype: Reaching girls in early childhood*. Lanham, Maryland: Rowman & Littlefield.

Sullivan, A., & Bers, M. U. (2016). Robotics in the early childhood classroom: Learning outcomes from an 8-week robotics curriculum in pre-kindergarten through second grade. *International Journal of Technology and Design Education*, *26*(1), 3–20. https://doi.org/10.1007/s10798-015-9304-5

Sullivan, A., & Bers, M. U. (2018). Investigating the use of robotics to increase girls' interest in engineering during early elementary school. *International Journal of Technology and Design Education*, Online First, 1–19. https://doi.org/10.1007/s10798-018-9483-y

Sullivan, A., Elkin, M., & Bers, M. U. (2015). KIBO Robot Demo: Engaging young children in programming and engineering. In *Proceedings of the 14th International Conference on Interaction Design and Children (IDC '15)* (pp. 418-421). Boston, MA: ACM. https://doi.org/10.1145/2771839.2771868

Svensson, A. (2000). Computers in school: Socially isolating or a tool to promote collaboration? *Journal of Educational Computing Research*, *22*(4), 437-453. https://doi.org/10.2190/30KT-1VLX-FHTM-RCD6

Varma, R. (2010). Why so few women enroll in computing? Gender and ethnic differences in students' perception. *Computer Science Education*, *20*(4), 301-316. https://doi.org/10.1080/08993408.2010.527697

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. https://doi.org/10.1145/1118178.1118215

BIOGRAPHIES

Professor Marina Umaschi Bers is Co-Founder and Chief Scientist at KinderLab Robotics; Professor and Chair, Eliot-Pearson Department of Child Study and Human Development; Adjunct Professor, Computer Science Department; Director, DevTech Research Group, Tufts University; Director, Early Childhood Technology (ECT) Graduate Certificate Program, Tufts University; and Author of Coding as Playground: Computational Thinking in the Early Childhood Classroom (2018), Designing Digital Experiences for Positive Youth Development: From Playpen to Playground (2012), and Blocks to Robots: Learning with Technology in the Early Childhood Classroom (2007)

Dr. Amanda Sullivan is a researcher at the DevTech Research Group at Tufts University. She has a Master's and Ph.D. in Child Development from the Eliot-Pearson Dept. of Child Study & Human Development at Tufts University where she specialized in educational technology for young children. Amanda's research focuses on exploring gender differences in STEM fields and developing strategies for engaging girls and women in technology and engineering. She is the author of the new book Breaking the STEM Stereotype: Reaching Girls in Early Childhood (2019) and co-creator of the ScratchJr Coding Cards (2018).