Int J Technol Des Educ
DOI 10.1007/s10798-015-9325-0

Constructing the ScratchJr programming language in the early childhood classroom

Dylan J. Portelance[1] · Amanda L. Strawhacker[1] · Marina Umaschi Bers[1]

Accepted: 9 July 2015

Abstract This paper seeks to contribute to the growing literature on children and computer programming by focusing on a programming language for children in Kindergarten through second grade. Sixty-two students were exposed to a 6-week curriculum using ScartchJr. They learned foundational programming concepts and applied those concepts to create personally meaningful projects using the ScratchJr programming app. This paper addresses the following research question: Which ScratchJr programming blocks do young children choose to use in their own projects after they have learned them all through a tailored programming curriculum? Data was collected in the form of the students' combined 977 projects, and analyzed for patterns and differences across grades. This paper summarizes findings and suggests potential directions for future research. Implications for the use of ScratchJr as an introductory programming language for young children are also discussed.

Keywords Elementary education · Early childhood · Instructional design · Programming · ScratchJr

Introduction

Policy makers and educators have grown concerned about alarming statistics regarding STEM education (science, technology, engineering, and math) and technical workforce in the United States. Between 2008 and 2018, STEM jobs are predicted to grow by 17 %, filled with workers who will earn greater salaries and face less chance of unemployment than non-STEM workers (Langdon et al. 2011). However, compared to their international

✉ Dylan J. Portelance
Dylan.Portelance@tufts.edu

[1] Eliot-Pearson Department of Child Study and Human Development, Tufts University, 105 College Ave, Medford, MA 02155, USA

Published online: 05 August 2015

peers, US students lag behind in elementary and secondary math and science performance, are less likely to choose a STEM-related major in college, and are more likely to eschew science and technology career fields even when they are qualified applicants (Chen 2013). Additionally, women and some minorities are underrepresented in STEM occupations compared with non-Hispanic Whites and Asian Americans (Landivar 2013). In order to combat these issues, a new movement for helping children develop computational thinking skills early on began (Henderson et al. 2007). Countless products born from both the public and private sectors seek to help children develop the computing skills needed for success in an increasingly technological world. Software such as Scratch (http://scratch.mit.edu/), Hopscotch (https://www.gethopscotch.com/), and Kodable (http://www.kodable.com/), to name a few, all aim to introduce coding and computational thinking to children within simple, accessible, and fun digital environments. There has also been a recent surge of interest among non-profits and other organizations in bringing computer science to elementary schools. Institutions such as Code.org (http://code.org), the Code-to-Learn Foundation (http://codetolearn.org), and CoderDojo (http://coderdojo.com) all seek to provide opportunities for children to learn how to code.

Previous research shows that children as young as 4 years old can learn simple computer programming concepts (Bers 2007, 2012). These concepts draw upon and reinforce cognitive skills such as number sense, literacy, and creativity in young children (Clements 1999). In addition to domain-specific skills, children who build a strong foundation in computational thinking competency can become more effective problem solvers and critical thinkers (Wing 2006). Learning to code does not just expand a child's career options in the future. Through coding, young programmers learn to comprehend and manipulate the digital landscape in which they live by thinking through "powerful ideas." A powerful idea is a "central concept within a domain that is at once epistemologically and personally useful, interconnected with other disciplines, and has roots in intuitive knowledge that a child has internalized over a long period of time" (Papert 2000; Bers et al. 2002). When programming, children work through powerful ideas of self-expression, computer science and engineering, such as how to communicate ideas through a range of new media and how to test and refine designs as a problem-solving strategy (Resnick 2013).

Many efforts in research have sought to determine the best practices for teaching computational thinking and programming to children in both formal and informal settings. Papert (1980) suggested that programming tools designed for children, like the popular constructionist programming environment LOGO, could spur children to think about their own thinking and concretize abstractions. Resnick et al. (2009) proposed that programming environments ought to have what Papert described as "low floors, high ceilings, and wide walls." In other words, learners should be able to create something easily right away (low floors), maintain their interest over time as they create progressively more complex projects (high ceilings), and allow students across a multitude of learning styles, cultures, and interests to learn and develop (wide walls). In this way, a programming tool can grow with children as they develop their skills and broaden their experience.

Other research has shown that carefully designed technological tools and curricula can allow children to learn computational thinking concepts in the classroom, at the same time that they are practicing foundational skills from more traditional domains like science, literacy, and more (Bers et al. 2002; Bers 2010; Bers and Horn 2010; Lee et al. 2011; Kazakoff and Bers 2012). Multiple researchers cite the positive effect that learning computer programming and computational thinking can have on other skills such as reflectivity, divergent thinking, literacy, mathematics, and social and emotional development (Clements and Gullo 1984; Clements and Meredith 1992; Flannery and Bers 2013).

This paper seeks to expand current research on children and programming by presenting a study on a programming intervention for students in Kindergarten through second grade. Sixty-two students were exposed to a 6-week curriculum in which they learned programming concepts in ScratchJr and then applied these concepts to the creation of personally meaningful projects. This paper addresses the following research question: Which ScratchJr programming blocks do young children choose to use in their own projects after they have learned them all through a tailored programming curriculum? Data from the 977 student-made projects was compared and contrasted across grade. Findings and research implications are discussed below.

ScratchJr: a programming environment for young children

The programming software used in this study is ScratchJr. ScratchJr is a free app, the product of a collaboration between the DevTech research group at the Eliot-Pearson Department of Child Study and Human Development at Tufts University (led by Professor Marina Umaschi Bers), the Lifelong Kindergarten research group at the MIT Media Lab (led by Professor Mitchel Resnick), and the Playful Invention Company (led by Paula Bonta and Brian Silverman).

ScratchJr was inspired by the popular Scratch programming language, used by millions of young people (ages 8 and up) around the world. The ScratchJr iPad app provides a developmentally appropriate way for children ages 5–7 to create interactive stories and games using a graphical programming language. Created with young children in mind, the app is carefully designed to match their cognitive, personal, social, and emotional development. At the time of data collection for this study, ScratchJr underwent several research and development phases including observations of children using the app to iteratively inform its design (Flannery et al. 2013). The Alpha version of the ScratchJr iPad app was used in the study. All figures in this paper depict graphics from the Alpha interface as participant children viewed them.

The programming language used within ScratchJr is composed of "programming blocks." Using the ScratchJr iPad app, children can create programming scripts (hereafter, "scripts") by dragging and snapping together sequences of blocks to control characters' motions, appearances, and interactions (see Fig. 1).

The design of the block shapes ensures that there is no way to make a syntax error in ScratchJr. The programming blocks in ScratchJr are designed to look like jigsaw puzzle pieces with visual properties that correspond to their syntactic properties. For example, the "Repeat Forever" block can only appear at the end of a program. Since nothing should follow a "Repeat Forever" command, the right side of this block is rounded so that another

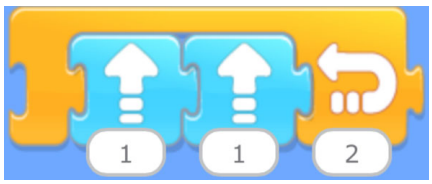

Fig. 1 A script in ScratchJr comprising three programming blocks. When this script runs, the corresponding character will move up a total of 4 times because there are two "Move Up" blocks inside a "Repeat" block with a parameter of 2. (Color figure online)

jigsaw piece cannot attach to. A script also runs as a sequence from left to right instead of the traditional top-to-bottom format of most adult programming languages, to reinforce print-awareness and English literacy skills (Flannery et al. 2013). As a character's script runs, the app highlights each block as it executes, representing the instructions given to that character on the stage. The software includes many additional features rich with opportunities for further research, but this paper will focus on the programming blocks in ScratchJr, which are at the core of the programming language.

Table 1 ScratchJr programming block categories. (Color figure online)

Block category	Sample block	Category description
Yellow Triggering blocks	 "Start on Green Flag"	These blocks can be placed the beginning of a script in order to make that script execute when a certain event happens. For example, when the "Start on Green Flag" block is placed at the beginning of a script, the script will execute whenever the green flag at the top right of the screen is tapped
Blue Motion blocks	 "Move Right" (1 step)	These blocks make characters move up, down, left, and right. They can also make characters go back to the place they started, rotate, and hop
Purple Looks blocks	 "Grow"	These blocks change how characters look. This set includes blocks that change the size of the character, add a speech bubble with user-defined text, and show or hide the character
Green Sound blocks	 "Play Pop"	Sound blocks play a sound in ScratchJr's library. Children can also record a sound and save it in a new Sound block
Orange Control flow blocks	 "Wait" (1/10th second)	Unlike Motion or Looks blocks, which visibly changes the characters on the stage, Control flow blocks change the nature of a character's program. For example, a sequence of blocks can go inside of a "Repeat" block and then the user can change the number of on this block to determine how many times the given script will execute
Red End blocks	 "Repeat Forever"	These blocks can be placed at the end of the program and determine whether something happens when the program finishes executing

For more detailed descriptions of individual programming blocks see http://www.scratchjr.org/learn.html

Programming blocks

There are six different categories of programming blocks in ScratchJr that children can choose from. Each category is represented by a different color. These are: yellow Trigger blocks, blue Motion blocks, purple Looks blocks, green Sound blocks, orange Control flow blocks, and red End blocks (see Table 1). If one imagines a ScratchJr project as a painting, then the array of color-coordinated blocks are the child's palette of paint. When the app opens to the project screen (Fig. 2), the Motion blocks are shown within the blocks palette in the middle of the screen. Children can drag as many Motion blocks from the palette to the programming area below it and then connect them to create scripts. In order to program with blocks from other categories, children may tap one of the color-coded buttons on the left side of the palette. For example, if a child taps the purple button, the Motion blocks on the palette are replaced with Looks blocks. In this way, children have access to over twenty-five programming blocks, without being overwhelmed by options on the screen.

Methods

A pilot study involving a ScratchJr curriculum called "Animated Genres" was implemented in three classrooms (N = 62) at a public elementary school in a suburb of Boston, MA. This curriculum taught children how to use all of the features in the ScratchJr iPad

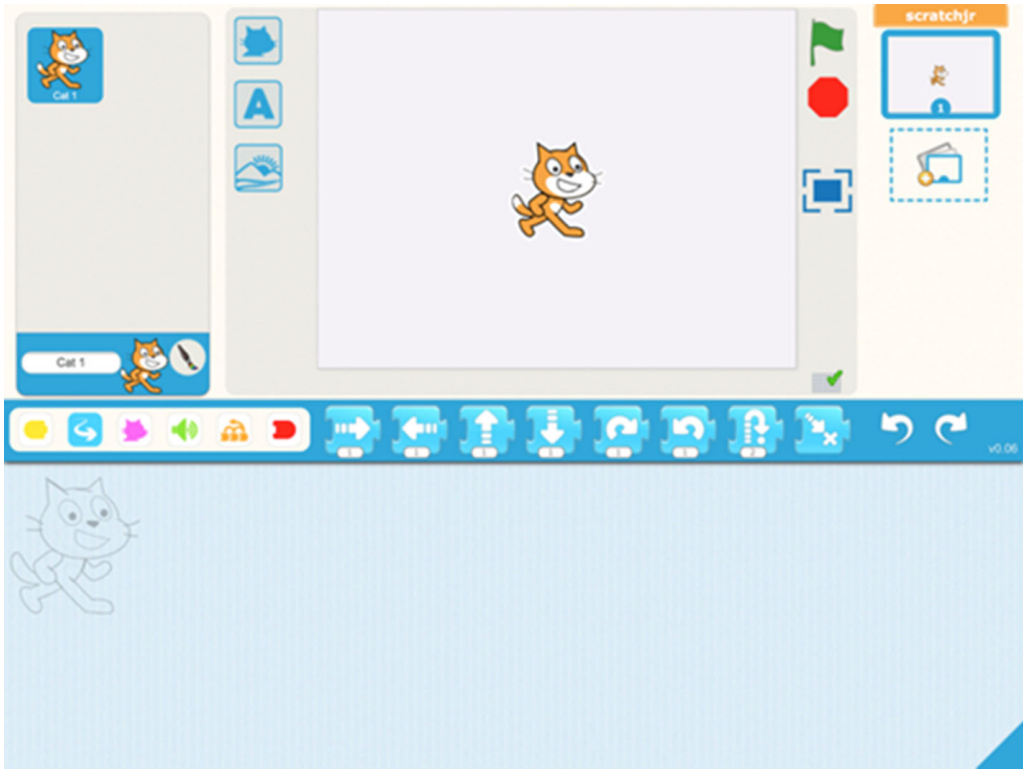

Fig. 2 A screenshot of the ScratchJr Alpha iPad app as it looks when a user starts a new project. (Color figure online)

app and particularly focused on how the programming blocks could be used to communicate and express oneself. During the curriculum, students created projects within three "animated genres:" collages, stories, and games. Hereafter, these genres will be referred to as "animated genres." The curriculum was taught by a team of research assistants with the help of regular classroom teachers. Although every lesson was led by researchers, classroom teachers were not only present during each lesson but also helped with classroom management and suggested changes to the curriculum based on their perceptions of what might work best for their students. At the end of the curriculum, ScratchJr project data was collected from each student's iPad. This data directly contributes to an investigation of which ScratchJr programming blocks young children use in their personally meaningful projects after they have learned them all through a tailored curriculum.

Participants

Participants in this study were $N = 62$ students in three classrooms at a public elementary school in a suburb of Boston, MA. The school was selected because of its close proximity to Tufts University, the willingness of its teachers and administrators to participate in the study, and its high inventory of iPads (1 for every student that participated in the study). The 62 students comprised one kindergarten class with $N = 21$ students, one first grade class with $N = 17$ students, and one second grade class with $N = 24$ students. Each class had one regular classroom teacher, one lead-teaching researcher, and between two and four assisting researchers. Participating teachers and parents/guardians of students gave signed informed consent before participating in this study.

Procedure

Twelve lessons were taught to each of the three classes. These lessons were planned to occur twice a week for 6 weeks for a period of 1 h, but occasional modifications were made to this timeline. Such changes did not impede data collection, but were typical consequences of working in a classroom setting. Researchers taught each lesson but classroom teachers were present in their respective classrooms to help with classroom management. Lessons were adjusted slightly to address specific class needs related to students' age, classroom culture, and teachers' style.

Data

After the intervention was completed, data was transferred from each student's ScratchJr project library to a database. The database contains a textual representation of 977 projects written by the 62 students. These projects comprise 3878 characters and 4044 programming scripts. Researchers used custom computer programs to count the frequency with which specific blocks appeared in students' scripts. The researchers also recorded open-ended field notes after each lesson on how the lessons were taught and how students reacted and behaved during the activities.

Curriculum

The curriculum was designed to teach all of the programming blocks in ScratchJr and then have students apply them to the creation of personally meaningful projects. Lessons in the kindergarten classroom were sometimes abbreviated or simplified in order to ensure they

were developmentally appropriate for that age group but each of the programming blocks was always taught in *some* context (e.g. kindergarten students learned about the repeat block but did not learn to use the repeat block in a way that involved arithmetic). During the lessons, the teaching team introduced ScratchJr features and programming blocks through activities, interactive demos, hands-on challenges, and free exploration time. Teachers and researchers also introduced concepts of storytelling and self-expression, focusing in particular on three genres of communication explored though the curriculum: collage-making, story-telling, and game design. Periodically throughout the curriculum, there were project days that challenged students to create a culminating ScratchJr project within a given genre and apply what they had learned in previous lessons.

Programming block lessons

Programming blocks were introduced progressively so that lessons taught early on provided scaffolding for later ones. Simpler blocks were taught before more complex ones, and early concepts were chosen in order to provide a foundation for developing more advanced ideas. Each of the lessons was divided in two segments. First, students were taught a selection of blocks and their functions. Second, students were invited to explore those blocks and create their own programs. Teachers also took advantage of moments when students used programming blocks in creative ways to highlight alternative approaches for the class. Vignettes that describe these instances are provided below.

Different teaching methods were used depending on the perceived complexity of the programming blocks. "Beginner Blocks" were taught using large group explorations. For

Fig. 3 ScratchJr project with Cat 1 script for counting down and "sending a message" to Rocket 1. (Color figure online)

the "Intermediate blocks," students were shown examples of ways those blocks could be used in a program. Finally, for "Advanced Blocks," students were shown specific scenarios in which the blocks would be useful, since these concepts required more explanation than simpler ones. For example, the "Start on Message" and "Send Message" blocks are similar to function calls in computer programming. Suppose a cat character has a "Send Message" block in its script and a seahorse character begins its script with a "Start on Message" block. This will cause the cat's program to "call" the seahorse's program to action. When the seahorse program is complete, the cat's script will continue. This set of blocks is useful for coordinating multiple characters in one project, but is relatively difficult to explain. To make the concept accessible to children, researchers showed a demonstration in which a character counts down, and a rocket ship blasts off. Teachers explained that the "Start on Message" and "Send Message" blocks made it possible for the rocket ship to blast off as soon as the character had finished counting down (see Figs. 3, 4). This lesson gave students a concrete and fun example of how to use one of the more abstract programming concepts in ScratchJr.

Beginner blocks

Vignette

Alexandra is a kindergarten student who has an affinity for pigs. During the lessons preceding the collage project day, she delighted in the fact that she could fill her ScratchJr

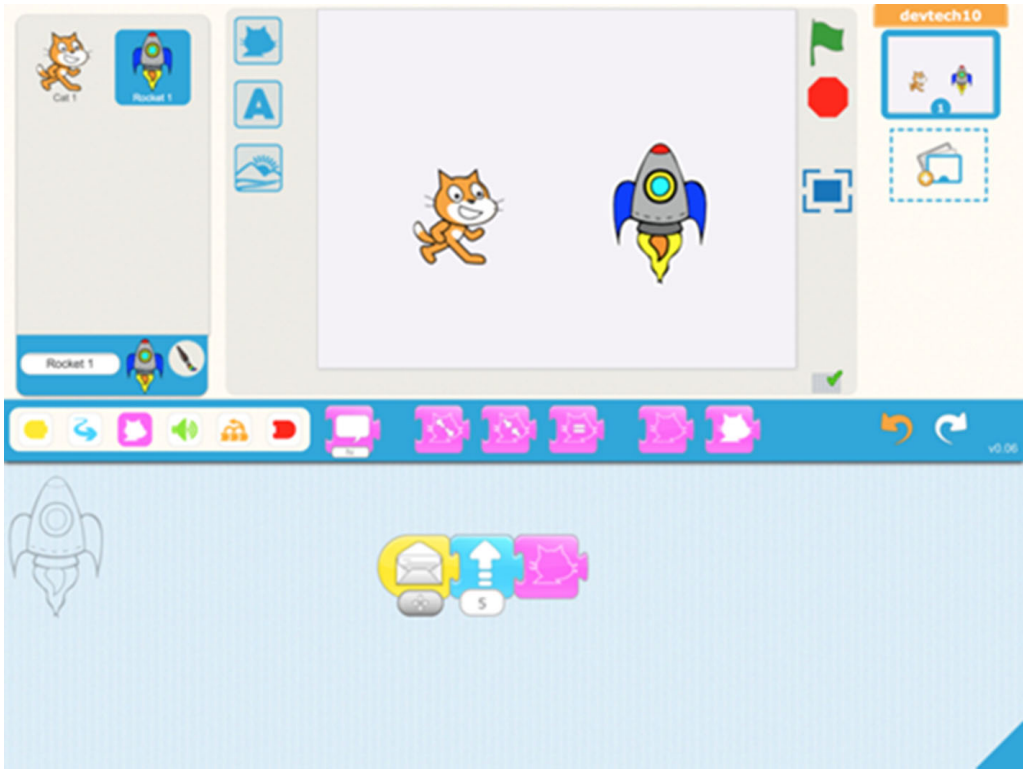

Fig. 4 Same ScratchJr project as in Fig. 1 but this time showing the Rocket 1 script that "receives the message" from Cat 1 and then makes it move up five spaces and hide. (Color figure online)

project stage with multiple pigs, each with their own names based on the names of her favorite classmates. Soon after, she learned from her ScratchJr teachers that she could change the colors of her pigs from the usual few shades of pink to any colors within the ScratchJr paint editor's palette. For her collage project, she decides to "paint" several pigs, each with a unique color scheme and then program them to walk back and forth and hop on the screen. As she designs her characters and programs their actions, she exclaims several times to her teachers and classmates, "Look! I made all these pigs and they're all rainbow but they're all different!" (Table 2).

Intermediate blocks

Vignette

Lilly is a second grade student who has been captivated for multiple lessons by the speech bubble programming block in ScratchJr. For her story project she decides to create a love story between two cats—both with custom colors courtesy of the app's paint editor. When she programs her two characters to say things with their speech bubbles, she notices that the cats are speaking at the same time, unlike conversations in real life where, "you're supposed to wait for the other person to talk." She tinkers with different blocks for a while and then discovers a perfect solution to this problem. After each speech bubble block, she connects a wait block. This block pauses the program so that there are intervals of time between her characters' speech bubbles and the speech looks more natural. After solving this problem, she goes to create a narrative in which the two cats meet in a meadow, teleport to outer space (using a "page turn" block), and then confess their love for each other amidst a backdrop of stars and planets. When it is Lilly's turn to share her project with the class, she talks excitedly about the narrative she constructed. But then many students want to know how she "got [her] cats to talk to each other." Beaming, she shows the class her code where she carefully programmed the amount of time that her characters would "wait" before saying their lines (Table 3).

Advanced Blocks

Vignette

Brian is a first grade student who plays a lot of video games at home. When he sees the demonstration of the *Frogger*-like game for his class he has an idea. Instead of having one enemy character for his playable character to avoid, he wants to have two. He likes how this will increase the difficulty of his game and make it more fun for his classmates to play. To make the implantation of this game project particularly easy for himself he decides to use the "duplicate program" feature in ScratchJr. First, he creates his hero character, a fish

Table 2 Teaching method for beginner blocks

Blocks	Teaching method
Move Right, Move Left, Move Up, Move Down, Grow, Shrink, Show, Hide, Stop	Blocks were introduced using large illustrations and group exploration activities. As a large group, students tested blocks by tapping them and observing how the character changed or moved. Then students individually continued to experiment on their own by creating short sequences

Table 3 Teaching method for intermediate blocks

Blocks	Teaching method
Hop, Go Home, Reset Size, Turn Right, Turn Left, Start on Green Flag, End, Wait, Say	Blocks were introduced with a demonstration by the teacher. Then students explored these blocks in ScratchJr independently by adding them to their own projects

from the ScratchJr character library. Then he creates two identical enemy cat characters. He programs the fish to move from left to right when a player taps it on the screen. Next, he decides how he wants his enemy cats to behave. He settles on an infinite loop of upward and downward motion at medium speed and programs the fish to disappear if it touches the cat. Finally, he cleverly drags the program he wrote for the first cat to the second cat in order to give the two enemies identical programs. He proudly shows his teacher how he did this so that he did not have to program two cats. Many of his classmates come over to his desk to play his game and complain, "it's so hard with two cats!" Brian giggles every time his friends cannot beat his game (Table 4).

Results

The purpose of this study was to address the research question: Which ScratchJr programming blocks do young children choose to use in their own projects after they have learned them all through a tailored programming curriculum? Throughout the intervention, students created hundreds of ScratchJr projects on their iPads. These projects were created during lessons, independent exploration time, and dedicated project-making days. After the intervention was finished, these projects were entered into a database using a custom computer program created by the lead researcher. This program identified and categorized aspects of students' work, such as number and type of blocks used. These data are described and analyzed below.

Figure 5 shows the average ratio of each programming block category between students in the three different grades: kindergarten, first grade, and second grade. On average, the most used blocks were Motion blocks, with "Move Right" being the most frequently used block across all grades. This means students dedicated most of their programming blocks to moving their characters around the screen. The second most common block in ScratchJr projects was the "Start on Green Flag" trigger block. The least common block was the "Stop" Control flow block for all three grades. This block stops a character's script.

Table 4 Teaching method for advanced blocks

Blocks	Teaching method
Set speed, Repeat, Repeat Forever, Go to Page, Start on Touch, Start on Tap, Send Message, Start on Message, Play Recorded Sound, Stop	Blocks were introduced with a special demonstration that showed unique scenarios where the blocks might be useful. Students were instructed to create similar projects to the demonstrations immediately after being shown them. They were then encouraged to explore independently once they seemed to have mastered their own close reproduction of the demonstration

It is evident from Table 5 that there are differences in blocks used by each grade. Analysis was conducted to determine whether any of these differences were statistically significant. A one-way between subjects ANOVA showed that there were significant differences in blocks used by children in different grades. Notably, there was a statistically significant difference ($p < 0.05$) between grades in the use of Trigger blocks [$F(2, 54) = 4.782$, $p = 0.012$] and red End blocks [$F(2, 54) = 44.872$, $p < 0.001$]. Post hoc comparisons of using the Tukey HSD test indicated that the average number of Trigger blocks use by second graders ($M = 29.2$, $SD = 2.97$) was significantly lower than the number used by Kindergartners ($M = 41.95$, $SD = 15.2$). Similarly, second graders used significantly fewer red End blocks on average ($M = 7.8$, $SD = 5.46$) than either Kindergartners ($M = 24.5$, $SD = 8.3$) or first graders ($M = 26.9$, $SD = 7.6$) (see Table 5; Fig. 6, 7).

Distributions for the Trigger and End block data allow those results to be generalized to larger populations of kindergarten through second grade students. Although significant differences between grades were also found for other block categories, these results need to be interpreted with caution for several reasons. Children used fewer Looks, Sound, and Control flow blocks overall, so the sample size for those data sets is much smaller. The Motion blocks data, despite being a large data set, did not meet conditions of homogeneity, meaning the data did not follow a predictable enough pattern to draw larger conclusions. The following statistical analysis for the other block types was conducted with non-parametric testing, meaning any interpretations can only be used to explain patterns within the sample.

A Kruskal–Wallis H test showed that there was a statistically significant different in number of Looks blocks ($\chi^2(2) = 9.48$, $p = 0.009$), Control flow blocks ($\chi^2(2) = 6.7$, $p = 0.035$), and Sound blocks ($\chi^2(2) = 7.8$, $p = 0.021$) used between the three grades. First graders were much more likely on average to use Looks blocks, while second grade students used more Control flow and more Sound blocks than their younger counterparts (see Table 5).

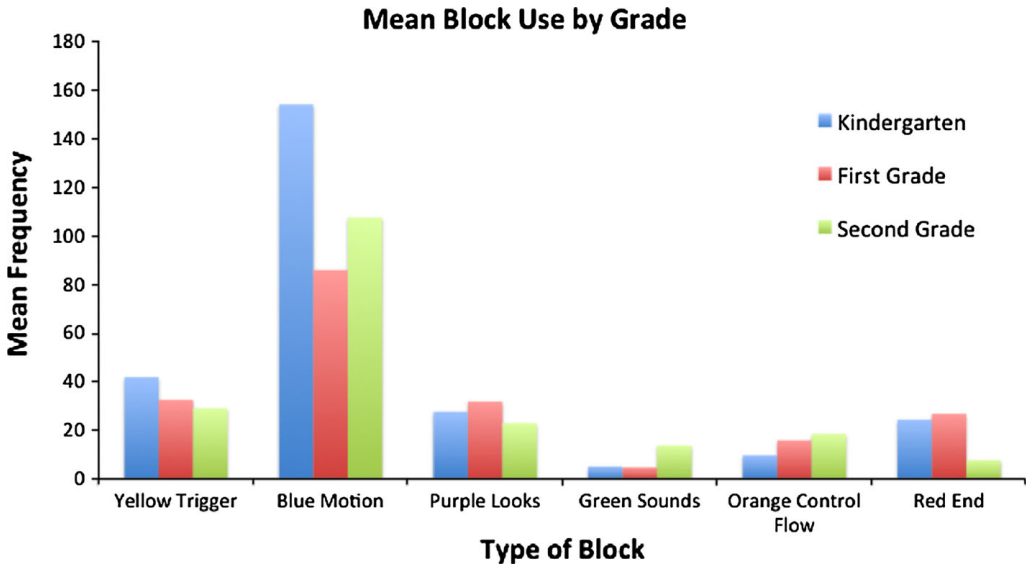

Fig. 5 The average block use by block category across all grades. (Color figure online)

Table 5 Average block usage by type of block (e.g. Trigger, Motion, Sound) for each grade

	Kindergarten		First grade		Second grade	
	Mean	SD	Mean	SD	Mean	SD
Trigger blocks	41.9	15.2	32.7	8.5	29.2	14.6
Motion blocks	154.3	90.5	86.1	31.4	107.7	49
Looks blocks	27.7	17.8	31.9	11.1	23	17.1
Sound blocks	5.2	4.2	4.8	5.7	13.9	13.5
Control flow blocks	9.8	7.4	15.9	12.9	18.6	14.7
End blocks	24.5	8.3	26.9	7.6	7.8	5.5

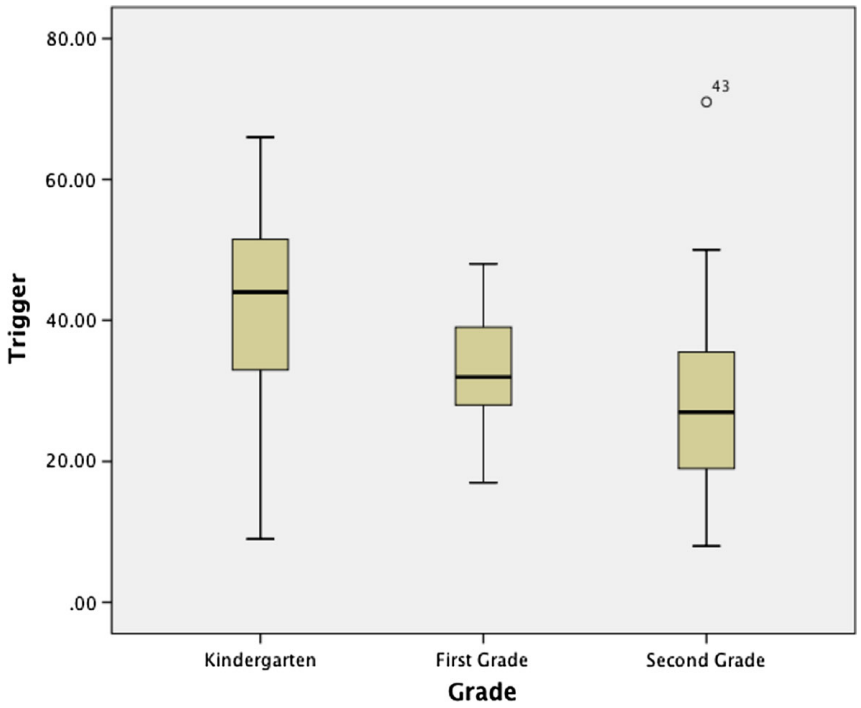

Fig. 6 *Box* and *whisker plot* showing Trigger block usage by grade. (Color figure online)

Discussion

The results provide a unique look at the raw usage of programming blocks during this particular intervention and a few key insights. First of all, it is unsurprising that Motion blocks were used the most for several reasons. Most notably, there are more Motion blocks than any other category of block. Additionally, they were taught early on in the curriculum because they were seen by the researchers as the easiest to teach and the easiest to learn. The blue motion blocks are also an illustrative example of the concept of "low floor." With Motion blocks, students can easily create a working project that is demonstrable to their peers while only understanding the programming concept of instructions that control characters. Furthermore, the blue motion blocks are shown as soon as someone opens the ScratchJr project screen. This would certainly increase the chances of them being used in programming scripts. Interestingly, there was no significant difference between the ratio of

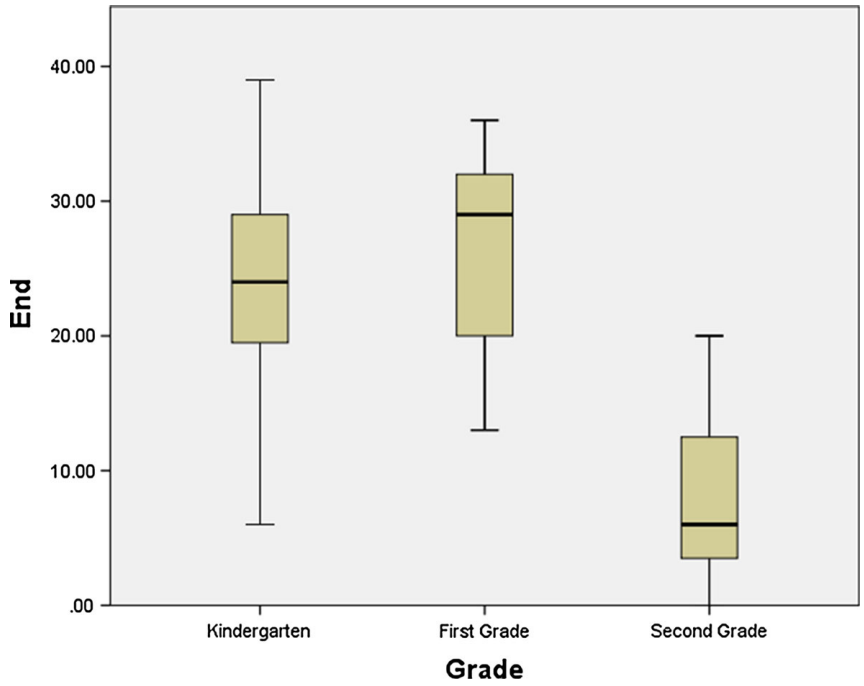

Fig. 7 *Box* and *whisker plot* showing End block usage by grade. (Color figure online)

blue programming blocks between males and females. Perhaps, this indicates that males and females are similarly attracted to the possibility of making their characters physically move around the screen as opposed to make them speak, change their size, or make them disappear and appear.

Results showed that second grade students used significantly less End blocks than kindergartners. This could be for a couple of reasons. First, it is possible that the notion of repeating forever that is exemplified by the "Repeat Forever" End block is a less enticing and novel to the second graders. Perhaps, there is a point in development where children are less captivated by the idea of never-ending action. It is also possible that the second graders clued into the fact that the "End" End block does not have any effect on the program it is a part of. The kindergarten students may be attracted to the symmetry that a "End" block can provide to a program with a Trigger block. Or perhaps the concept of a block that only serves the purpose of marking the end of a program as opposed to causing the program or character to change is not something that is developmentally appropriate for kindergarten and first grade students. Another quality that distinguishes the "End" block from all the other blocks is its lack of a logo—it is completely blank. This characteristic may make it more enticing and mysterious to some children.

Additionally, results showed that second grades students used significantly less Trigger blocks than kindergartners. One explanation for this significant difference could have to do with the way the ScratchJr interface is designed. If a student programs a script and does not include any Trigger blocks, there is still a way that the student may begin the script's execution. The student can simply tap the first block in the script and all of the following blocks will execute. It is possible that second graders are more likely to discover and use convenient features of the software that provide alternatives to the ones they are formally taught in the classroom and therefore used less Trigger blocks. It may also be the case that,

similarly to what was discussed about End blocks, kindergarten students are more attracted to the symmetry that the Trigger blocks provide when paired with an End block in a script. This would cause them to have a higher frequency of Trigger blocks.

It is no surprise that second graders used more Control flow blocks than students in younger grades because most of these blocks were identified as difficult to learn. Perhaps the younger students were less likely to use blocks they did not understand and second graders were more likely to understand the blocks deemed more complex. On a similar note, while the functions of the Sound blocks are not particularly complex, the interface for recording a sound was something that, anecdotally, many students had trouble with. Perhaps this could explain why more of the second grade students used Sound blocks than younger ones.

Limitations and future directions

There are many limitations to conducting a study in a real classroom rather than a laboratory setting. First of all, for logistical reasons, it was impossible for the same researchers to teach every lesson and every class. Although researchers and teachers taught the same curriculum, there were distinct variations in teaching practices across grades and across lessons based on variable physical classroom environments, students' social dynamics, times of day, and other contextual factors. With the assumption that teachers knew their students and classroom best, they were given complete freedom to make changes to the curriculum in order to garner the best possible learning experience for their classes. One salient example of this occurred during the story project day. The kindergarten teacher suggested to researchers that the students be read a story and then create ScratchJr projects with characters in that story rather than build story projects without any constraints. The decision to allow teachers this kind of freedom undoubtedly could have affected the data presented earlier. Studies that build upon this work should consider the effects of different teaching styles and classroom environments on how students learn and use programming blocks to create projects in ScratchJr.

This study was also limited by its data collection technique. Unfortunately, due to logistical reasons and the incomplete state of the ScratchJr app during the time of this research, the data regarding content that students developed using ScratchJr is confined to the scripts and projects. The data that was collected does not include the time during the intervention that different scripts or projects were created. Another shortcoming regarding this study's data is that programming block parameters were not recorded. Many blocks such as directional motion blocks and the speed block allow the user to input parameters to customize them. For example, the speed block allows the user to choose slow, medium, or fast speed. Due to a bug in the app, blocks with user-determined parameters were reset when the app was restarted even if the blocks belonged to a script within a saved project. Further work on ScratchJr will certainly benefit from examining how students' programming and content change throughout a curricular intervention as well as how students use the parameters of the programming blocks.

Many other challenges that limited this study's effectiveness were created by technological problems, and bugs. First of all, although each child had his or her own iPad during the study, not every iPad was able to connect to a classroom's Apple TV whenever a ScratchJr project on that iPad needed to be shared with the class. Because of this, students were afforded varying of opportunities to share their work with their peers. Whether or not a given student was able to share frequently with the class or was disappointed by his or her iPad's inability to do so may have affected his or her project content and/or interest in

learning and creating with ScratchJr. Furthermore, because the version of ScratchJr used during the curricular intervention was not a production ready app, there were many bugs in the app that researchers were unaware of. Throughout the study, several students experienced moments where the app would freeze unexpectedly and they would lose some of their recent work on a project. Without these instances and the inconsistency of the learning experience that they created, the scripts presented earlier in this paper would be between. In addition, these freezes could have attributed to frustration with the app for those students who experienced them more frequently. Further research on ScratchJr in the classroom could avoid these problems by removing the Apple TV from the equation if sharing projects is included in the curriculum. Future studies will also benefit from the existence of a production quality app that can be downloaded onto any student's iPad.

Conclusion

As products and calls to action continue to address the growing need for STEM workers and attempt to alleviate the disadvantages of underrepresented groups in STEM fields, research will need to address not only the tools used to teach topics like computational thinking and computer programming but also the ways these tools are taught. In order to address the research question—"Which ScratchJr programming blocks do young children choose to use in their own projects after they have learned them all through a tailored programming curriculum?"—this paper introduced a new curriculum for teaching the programming blocks in ScratchJr and built upon a small amount of work regarding the ScratchJr programming environment. A few key differences between how children in different grades use ScratchJr programming blocks were identified. In particular, these differences were found when students were taught each block individually and then encouraged to use them to create personally meaningful projects within animated genres.

There are many opportunities for further research on ScratchJr's role as an instructional tool in a classroom engaged in learning computational thinking and self-expression with technology but one particularly fruitful avenue to explore will be how programming blocks are used differently between each animated genre. Research that seeks to draw a picture of how young children learn the ScratchJr programming language and make their own projects with it will help further examine just how low the "floor" is, how high the "ceiling" is and how wide the "walls" are in ScratchJr. Furthermore, it will illuminate what concepts and project archetypes correspond to the "floor," "ceiling," and "walls." With this knowledge, the ScratchJr app and other programming environments for young children can be optimized for their role in producing positive learning and developmental outcomes.

Acknowledgments This research was supported by the National Science Foundation (NSF DRL-1118664) and the Scratch Foundation. The researchers would also like to extend a thank you to the many teachers, staff, students, and parents who participated.

References

Bers, M. U. (2007). Project InterActions: A multigenerational robotic learning environment. *Journal of Science and Technology Education., 16*(6), 537–552.

Bers, M. U. (2010). Beyond computer literacy: Supporting youth's positive development through technology. *New Directions for Youth Development, 128*, 13–23.

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

Bers, M. U., & Horn, M. S. (2010). Tangible programming in early childhood: Revisiting developmental assumptions through new technologies. In I. R. Berson & M. J. Berson (Eds.), *High-tech tots: Childhood in a digital world* (pp. 49–70). Greenwich, CT: Information Age Publishing.

Bers, M. U., Ponte, I., Juelich, K., Viera, A., & Schenker, J. (2002). Teachers as designers: Integrating robotics into early childhood education. *Information Technology in Childhood Education, 2000*(1), 123–145.

Chen, X. (2013). *STEM attrition: College students' paths into and out of STEM fields (NCES 2014-001)*. Washington, DC: National Center for Education Statistics, Institute of Education Sciences, U.S Department of Education.

Clements, D. H. (1999). The future of educational computing research: The case of computer programming. *Information Technology in Childhood Education Annual, 1999*(1), 147–179.

Clements, D. H., & Gullo, D. F. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology, 76*(6), 1051–1058. doi:10.1037/0022-0663.76.6.1051.

Clements, D. H., & Meredith, J. S. (1992). *Research on logo: Effects and efficacy*. Retrieved from http://el.media.mit.edu/logo-foundation/pubs/papers/research_logo.html

Flannery, L. P., & Bers, M. U. (2013). Let's dance the "Robot Hokey-Pokey!": Children's programming approaches and achievement throughout early cognitive development. *Journal of Research on Technology in Education, 46*(1), 81–101.

Flannery, L.P., Kazakoff, E.R., Bontá, P., Silverman, B., Bers, M.U., and 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. doi:10.1145/2485760.2485785

Henderson, P. B., Cortina, T. J., Hazzan, O., & Wing, J. M. (2007). Computational thinking. In *Proceedings of the 38th ACM SIGCSE technical symposium on computer science education (SIGCSE'07)* (pp. 195–196). New York, NY: ACM Press.

Kazakoff, E., & Bers, M. (2012). Programming in a robotics context in a kindergarten classroom: The impact on sequencing skills. *Journal of Educational Multimedia and Hypermedia, 21*(4), 371–391.

Landivar, L. C. (2013). *Disparities in STEM employment by sex, race, and hispanic origin* [PDF Document]. Retrieved from http://www.census.gov/prod/2013pubs/acs-24.pdf

Langdon, D., McKittrick, G., Beede, D., Khan, B., and Doms, M. (2011). *STEM: Good jobs now and for the future* [PDF Document]. Retrieved from http://www.esa.doc.gov/sites/default/files/reports/documents/stemfinalyjuly14_1.pdf

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., et al. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32–37.

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

Papert, S. (2000). What's the big idea? Toward a pedagogy of idea power. *IBM Systems Journal, 39*(3.4), 720–729.

Resnick, M. (2013). *Learn to code, code to learn: How programming prepares kids for more than math*. EdSurge (May 8, 2013). Retrieved from https://www.edsurge.com/n/2013-05-08-learn-to-code-code-to-learn

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60–67.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.