What They Learn When They Learn Coding: A Study using the ScratchJr Solve It programming assessment for young children

Corresponding Author:

Amanda Strawhacker

[Amanda.strawhacker@tufts.edu](mailto:Amanda.strawhacker@tufts.edu)

Telephone: (617) 627-4490

Fax: (617) 627-3503

Co-Authors:

Dylan Portelance

Marina Bers

DevTech Research Group, Eliot-Pearson Department of Child Study and Human Development, Tufts University, Medford, MA 02155

#### Abstract

Computer programming for young children has grown in popularity among both education researchers and product developers, but still relatively little is known about how to assess and track young children’s learning through coding. This study presents an assessment tool to track Kindergarten through second grade students’ learning after engaging in a programming curriculum. Researchers worked with N=57 Kindergarten through second grade students over seven weeks to implement a curriculum using ScratchJr to introduce concepts of sequencing to create animated stories, collages, and games. At the end of the learning intervention, students were assessed on their knowledge of the ScratchJr programming language and underlying programming concepts. Results show that while all students mastered foundational coding concepts, there were marked differences in performance and comprehension across the three grade levels. Implications for computer programming education in the context of age-graded early childhood cognitive development are discussed.

#### Keywords

Early childhood education, computer programming, ScratchJr, assessments

#### Biography

Amanda Strawhacker is a research scientist on the ScratchJr research project at the Developmental Technologies Research group at Tufts University. Her research interests include children’s intellectual and creative development through the lens of technological tools for learning.

Dylan Portelance is an M.A. candidate at Tufts University’s Eliot Pearson Department of Child Study and Human Development.

Marina Umaschi Bers is a professor at the Eliot-Pearson Department of Child Study and Human Development and the Computer Science Department at Tufts University. She heads the interdisciplinary Developmental Technologies research group which focuses on designing and studying innovative learning technologies to promote positive youth development.

Introduction

In recent years, the popularity of computer programming as a learning domain has led to the development of many new tools, teaching approaches, and curricular interventions for young children (Kindergarten through second grade) to explore coding. Platforms like Tynker (https://www.tynker.com/), Hopscotch (http://www.gethopscotch.com/), and Move the Turtle (http://movetheturtle.com/) promise to engage children in computer coding at a level fitting their cognitive needs, and organizations like Code.org (http://code.org/) and Code Academy (http://www.codecademy.com/) organize foundational coding principles into interactive lessons for all ages, starting with Kindergarten. The growth of these new technologies and classroom interventions may be related to a shift in popular conceptions and policies about when children are old enough to begin learning computer programming.In 2010, the US Department of Education’s National Education Technology Plan included early childhood technology classrooms as an area for research and innovation, and in 2013 the United Kingdom introduced a mandatory computing syllabus into their national curriculum (2010; 2013). Prior research in early childhood settings has shown the benefits of introducing technology and engineering as academic curricular domains for children’s sequencing ability, design learning, and executive functioning, to name a few (Flannery & Bers 2013; Kazakoff 2014; Wyeth 2008; Kazakoff, Sullivan, & Bers 2013; Kazakoff & Bers 2011; Bers, 2008). The recent popularity of technology and engineering programs has led to the creation of numerous coding tools and environments for young children. However, relatively little research has been conducted to determine developing assessments to examine programming knowledge gains in young learners (Lee, Ko & Kwan, 2013). This study uses innovative programming-knowledge assessments to investigate gains in K-2nd grade students’ knowledge of foundational programming concepts after completing an introductory ScratchJr curriculum.

ScratchJr is an introductory programming language that enables young children (ages 5-7) to explore fundamental computer programming concepts through creating interactive animations, stories, and games. Children snap together graphical programming blocks to make characters move, jump, dance, and sing. The ScratchJr programming environment is the product of an NSF-funded (DRL-1118664) research collaboration between the Lifelong Kindergarten Group at MIT and the Developmental Technologies Research Group at Tufts University (http://www.scratchjr.org). ScratchJr was inspired by the MIT Lifelong Kindergarten group’s popular Scratch programming language (http://scratch.mit.edu) for children ages eight and older, but with interface and programming language adjustments to make them developmentally appropriate for younger children (Flannery, Kazakoff, Bontá, Silverman, Bers & Resnick 2013).

This study presents results from an experimental technology intervention using ScratchJr in a Kindergarten, a first grade, and a second grade classroom. This intervention was completed using a tailored seven-week curriculum on computer programming, communication, and creative expression written by the Developmental Technologies Research Group at Tufts University. Students’ programming knowledge was assessed using tailored performance tasks, and compared with student-created ScratchJr artifacts from class sessions to arrive at a holistic view of the participants’ technology understanding. Average scores are reported, and several students’ results are highlighted as cases to illuminate observed learning patterns throughout the intervention. Conclusions are drawn about learning milestones and expectations for children in this age group who are learning to program. Implications for future research, classroom practice, and curriculum development are also discussed.

Literature Review

Prior research has focused on computer programming as a medium to explore foundational skills of math, logical ordering, and sequential ordering (Kazakoff, 2013; Wyeth, 2008; Kazakoff & Bers, 2014; Pea & Kurland 1984; Clements 2002). Programming is traditionally associated with these kinds of skills, since it can require detail-oriented, logical thinking patterns (Robins, Roundtree & Roundtree 2003). However, Resnick (2006) reminds us of the unique power of computers to virtually represent creative projects that would be difficult or impossible to realize in the physical world. When children learn to program, they are effectively learning to communicate in a new language. Just as with a new language, the ultimate learning goal may not necessarily be simple technical proficiency, but rather fluency to convey and interpret ideas (Buck & Stucki, 2003; Hudson, Isakson, Richman, Lane, & Arriaza-Allen, 2011). Additionally, studies on young children’s learning through “purposeful, goal-directed programming” and debugging exercises has shown that giving children a meaningful goal to work towards, rather than simply assigning a task, helps them engage more deeply with subject matter and learn more cross-domain skills (Wyeth, 2008; Lee & Ko, 2012; Lee, Ko & Kwan, 2013). In this sense, technology and computers can be a well-suited platform for children to practice soft skills of communication, self-expression, and creativity.

The growing popularity of technology interventions in early childhood settings is a source of much innovation and discussion in the education community, but most researchers and educators can agree that a well-designed teaching tool (technological or otherwise) is only as good as its learning outcomes (Barnett 2003, Boschman et al 2014, Vanderline et al, 2009). Within the domain of computing comprehension itself, evidence shows that even kindergarteners can create and discuss programs in a way that demonstrates understanding of fundamental programming concepts. They can name actions that correspond with instructions, order events in a logical sequence, and create a simple program to achieve a hypothetical goal (Kazakoff & Bers, 2012; Strawhacker, Sullivan & Bers, 2013). Still, all of these tasks are cross-domain and may indicate mathematical and spatial/geometric knowledge as well as programming comprehension (Starkey, Klein & Wakeley, 2004). Relatively little work has been done on assessing and modeling children’s understanding of programming as a learning domain in itself.

This study turns to the question of what children can learn about programming after a curriculum intervention that uses ScratchJr as an expressive tool to create personally meaningful projects. In the context of this paper, children’s programming ability is measured through their understanding of individual programming commands and their ability to sequence these commands into a functional program. The research question of this study is as follows: Given exposure to a curriculum on programming and content-creation,

1) How do young children (K-2nd grade) score on tailored assessments of programming instruction recognition,

and

2) Is there a difference across grade and gender for these students?

These assessments are designed to capture student learning in key areas of developing programming comprehension, informed by qualitative results from previous pilot studies of children using ScratchJr (Flannery et al 2013). While these early pilots showed what patterns children demonstrate when learning a new programming language, this study looks more closely at the differences across grade and gender, in an attempt to parse trends and differences in student populations.

Method

Participants and context

A suburban public school in the greater New England area of the US was chosen as the research site for this study. This site was chosen because the school uses a “one iPad per child” policy, which allows every student to have access to an iPad throughout the day during relevant class sessions. One Kindergarten, one first grade, and one second grade classroom were purposively selected based on the relevance of this age range to the pedagogy under study. The classes comprised a convenience sample of N=57 K-2nd grade participant children (27 male, 30 female). To protect participant identities, all names presented are pseudonyms.

In this study, these N=57 K-2nd grade students spent thirteen sessions learning how to program with ScratchJr to explore themes on self-expression and communication. Because of the open-ended nature of ScratchJr, a curriculum was developed by the DevTech Research Group to target specific learning domains and goals, and to introduce the programming in an effective and developmentally appropriate way (Bers 2014).



Figure 1. During ScratchJr lessons, children moved freely around the room, collaborating informally.

The study was conducted in students’ regular classrooms, with Tufts University researchers leading lessons and classroom teachers assisting and collaborating throughout. ScratchJr was introduced during flexible time in the regularly scheduled school day, and was not tied to required standards of the school’s K-2nd grade curriculum. During ScratchJr classes, students were encouraged to move around the room, collaborating and sharing knowledge informally (see Fig. 1). Each student used their school-issued individual iPad tablet, with their ScratchJr classwork saved locally to that device.

All children in this study participated in a constructionist curriculum that fostered open exploration and self-directed project creation (Papert 1980; Bers 2014). This model was chosen because of its success in initial ScratchJr pilot studies. Using the constructivist learning approach, children at the Kindergarten level were able to experiment with and master the foundational programming concepts sufficient to build an interactive project. This ability corresponds with other developing skills in this age range, like sorting objects by color and size, logically completing a sequence of actions, and retelling the day’s events from beginning to end (Kazakoff & Bers, 2012). In the next section, more time will be spent discussing the ScratchJr pilot studies and their influence on the current study design.

ScratchJr

Children used the iPad app ScratchJr to create their computer programs. ScratchJr is a programming environment based on the popular Scratch software, and modified for children in kindergarten through second grade (Flannery et al. 2013). The workspace of one project in the app consists of four parts: a programming “editor,” or workspace to drag programming blocks, a stage where characters act out the instructions on the programming blocks, a list of all the characters on the stage, and a gallery of up to four pages, each with a new stage and workspace to continue the project (see Fig. 2).



Figure 2. A screenshot of the ScratchJr interface. The interface consists of (A) a programming “editor,” or workspace, (B) a stage where characters execute programs, (C) a list of all the characters on the stage, and (D) a gallery of up to four pages, each with a new stage, workspace, and character list to continue the project.

Children use the programming blocks to create programs (also called “scripts”) that outline instructions for their selected characters to perform on the stage. These programs can range from very simple instructions (i.e. a single character growing in size) to quite complex (i.e. two or more characters interacting and conversing on stage). Children can remember which character they are programming by referring to the watermark on the editor, as well as the highlighted image of the character in their character list. Programs can be copied to across characters, characters can be copied to different pages, and pages can be reordered. These features were intended to create a sense of cohesion in a project’s narrative (Flannery et al. 2013). Because there are hundreds of potential combinations of the 28 programming blocks, ScratchJr projects are usually quite unique. This makes it a powerful platform to focus on the creative, open-ended aspects of computer programming, as well as the learning affordances of a programming environment with the power and open-endedness reminiscent of more advanced languages.

Data Collection

Data was collected in the form of assessments designed to capture programming comprehension and fluency in the ScratchJr icon-based language. These data are analyzed using mainly quantitative methods to determine patterns in student understanding of programming concepts.

Solve Its: Assessing learning through problem solving

Children’s programming ability was measured using the following two metrics: block recognition and sequencing ability. To ascertain a child’s ability to recognize ScratchJr programming commands, they are shown a short animation of one or more ScratchJr characters while the corresponding commands used to construct that animation remain hidden. Then, using a print-out with all of the ScratchJr programming commands listed, they are asked to circle the blocks they believe were used to create the program they observed (see Fig. 3). For example, if a child is shown an animation of a cat jumping, she might recognize that the cat’s program included a “jump” block, and circle that on her assessment sheet. A total of six block recognition Solve Its was administered to each child in 1st and 2nd grade, and five Solve Its to Kindergarten students since that task addressed concepts that had not yet been covered in the Kindergarten classroom (see Table 1).



Figure 3. A completed block recognition assessment sheet. Circled blocks represent the student’s answers for viewing Solve It #4, #5, etc.

Sequencing ability was measured using similar methods. Each child was asked to observe the same program being executed that they just watched for the block recognition task. However, this time they are given paper cut-outs of every single block, and asked to choose and arrange them into the program sequence that they believe created the program they observed. This task essentially tests a child’s ability to observe the results of complete programs and reverse-engineer them from a larger set of isolated parts. For the purposes of this paper, analysis will focus on the block recognition tasks, and the sequencing tasks will be used to help to inform analysis.

Table 1.

*Description of Solve It Tasks*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Solve It Task | Grades Assessed | Difficulty Levela | Concept Assessed | Correct Programs |
| Task  #1 | K, 1, 2 | Easy | Motion Blocks | Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 12.40.03 AM.png |
| Task #2 | K, 1, 2 | Easy | Looks Blocks: Resizing | Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 12.44.23 AM.png |
| Task #3 | K, 1, 2 | Medium | Control Flow: Speed | Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 12.51.18 AM.png |
| Task #4 | K, 1, 2 | Medium | Control Flow: Timing | Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 12.41.28 AM.png |
| Task #5 | K, 1, 2 | Hard | Control Flow: Endless loops  Triggering Commands: Start on bump | Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 12.46.40 AM.pngMacintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 12.46.45 AM.png |
| Task #6 | 1, 2 | Hard | Control Flow: Message passing interactions  Control Flow: Repeat loops | Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-14 at 11.57.10 PM.pngMacintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-14 at 11.57.26 PM.png |

a. “Difficulty level” was arrived at by consensus of the research team and informed by pilot ScratchJr curriculum research.

The above chart shows all of the programs that children viewed in order to complete their Solve It tasks. Questions were first created, and then organized by the research team into Easy, Medium, and Hard questions based on the complexity of the program and the blocks used, and prior performance of students using ScratchJr in pilot studies (Flannery et al 2013). Some strong patterns emerged throughout the initial pilot research of ScratchJr with K-2nd grade students that informed the difficulty categorization of these Solve Its. For example, around halfway through the pilot studies, most children could comfortably use Motion and Looks blocks, and modify their programs using the Start on Green Flag, End, and Repeat Forever blocks. Most students could explore unfamiliar blocks representing concrete actions to learn what they do, which is perhaps how they originally learned the effect of most Motion and Looks blocks on their characters. Additionally, some patterns indicated differences in programming and problem solving capability. Kindergarten aged children generally showed difficulty with such tasks as mastering the functions of blocks with meta-level instructions or invisible outcomes (i.e. Repeat Loops, Speed blocks) and coordinating multiple programs within a character, or multiple characters on a page. Students in all grades with lower programming capability were less likely when stuck to choose a problem solving strategy that allowed them to continue working on the same task, opting instead to change the final outcome of their project (Flannery et al 2013). These trends helped shape the Solve It assessments, and questions were designed to target many of these observed differences in programming ability from high- to low-mastery students (see Table 1).

Scoring Solve It Assessments

Assessments were scored using a rubric developed on a subset of the children’s responses, based on calculating errors against closest potential correct answers. Scoring rubrics and grading methods were based on prior research with programming assessments for Kindergarteners in robotics and programming contexts (Strawhacker, Sullivan & Bers 2013).

A perfect score was a score of zero. Raw number scores were given base on d how many steps were required to change a child’s answer to a correct program. For example, a child who missed a correct block and circled an incorrect block would receive a score of 2. Several patterns emerged in the data involving commonly confused blocks. Some blocks were mistaken for others with similar programmatic functions, as when a child circled both a “move up” and a “move down” block when they observed a character using the “jump” block. Other confusions seemed to be based on the interface of the blocks themselves, for example when a child chose a “spin left” instead of a “spin right” block. Although it is difficult to know whether children were struggling with the look or function of blocks, these types of errors caused alterations to how these responses were scored to minimize error inflation.

Since several Solve It tasks could be recreated with multiple “correct” programs, researchers graded student responses against the program that was closest to the child’s answer. For example, one Solve It task showed a cat moving forward multiple times. Some children circled the “move right” block, which can be used to create a correct program (see Table 2) Other students circled the “move right” and “Repeat” loop blocks, which could also be used to create a correct program. Some limitations were introduced, however. For example, in the same Solve It task, a student could technically make a correct program using a “move left” block and negative number parameters, causing the cat to walk backwards. Since students had not covered negative numbers in either their ScratchJr or regular math lessons (and the cat would be facing the wrong direction anyway), this was considered an incorrect answer, and graded against the nearest correct answer (see Table 2). Similarly, answers missing correct blocks lost points since the program would not be executed exactly as shown. For example, a missing trigger block is always an error, since programs without trigger blocks can only be executed from the editor and not from presentation mode (all Solve It tasks were shown in presentation mode).

Table 2.

*Sample programs made with circled-block responses to Solve It Task 1, select blocks to make a character move forward when green flag is tapped.*

|  |  |  |
| --- | --- | --- |
| Correct programs using two blocks | Correct programs using three blocks | Incorrect programs |
| Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 11.50.23 AM.png | Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 11.51.16 AM.png | Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 11.51.41 AM.png |
| Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 11.51.01 AM.png | Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 11.51.27 AM.png | Macintosh HD:Users:astraw01:Desktop:Screen Shot 2014-12-15 at 12.04.28 PM.png |

Results

In order to address our research question, “How do young children (K-2nd grade) score on tailored assessments of programming instruction recognition,” Solve It assessment scores were compiled and analyzed for differences and trends.

After testing for outliers using boxplots and skewness values, three extreme outliers emerged from the data. Based on the results of these students’ assessments, as well as the researchers’ field notes and experiences administering the assessments, these outliers probably reflect a combination of several factors. Researchers observed that some children were unable to maintain attention while the assessment was being administered, which resulted in Solve It responses that may not reflect the child’s true understanding of the content. This also led to off-task play with the assessment materials as a diversion. This makes it unclear whether to attribute these outlier scores to a deep lack of understanding of the ScratchJr environment, or developmental variability in levels of attention and self-control. For this reason outliers were removed, bringing the N from 57 to 54 (29 females, 25 males).

Table 3.

*Summary statistics for mean scores on ScratchJr block recognition assessment, by grade.*

|  |  |  |  |
| --- | --- | --- | --- |
| Grade | N | Range | Mean ± SD |
| Kindergarten | 19 | 1.8 – 4.6 | 2.9 ± 0.78 |
| First | 16 | 1.0 – 3.17 | 1.86 ± 0.64 |
| Second | 16 | 0.67 – 2.67 | 1.45 ± 0.59 |

On average, Kindergarten showed the highest number of errors (*M* = 2.9), followed by first grade (1.86) and second grade (1.45). Standard deviation was larger for Kindergarten (*SD* = 0.78) and narrowed in first (*SD* = 0.64) and again in second grade (*SD* = 0.59). These trends indicate that the higher a child’s grade in school, the more likely that their performance on the block recognition Solve Its will have fewer errors, and that a prediction of their score will be accurate.

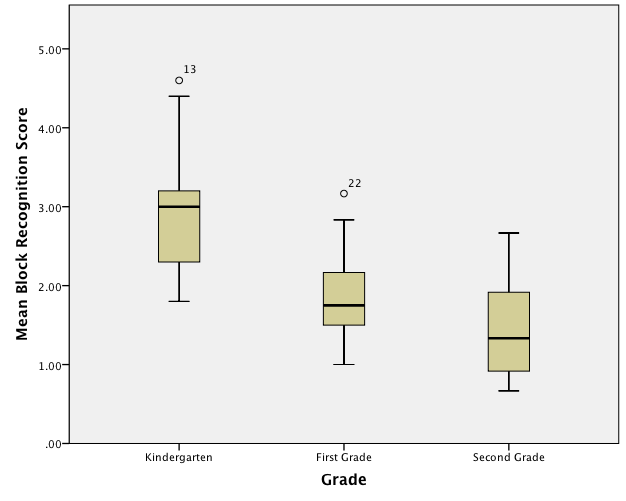


Figure 4. Boxplots showing distributions by grade of mean scores on block recognition assessment.

A Pearson product-moment correlation was run to determine the relationship between a student’s grade and their mean score on the block-recognition Solve It assessment. The data showed no violation of normality, linearity, or homoscedasticity. There was a strong, negative correlation between age and errors made, which was statistically significant (r = -.68, n = 52 p < .001). This indicates that older children are statistically less likely to make errors than younger ones, by a factor of about 1.3 fewer errors on average for every one-year increase in age. For example, a six-year-old is likely to make about 2 to 3 more errors than an eight-year-old on a full six-question Solve Its Assessment.

A one-way between subjects ANOVA was conducted to compare the effect of a student’s grade on their performance on the block recognition Solve Its in the Kindergarten, first grade, and second grade conditions. In this case, an ANOVA was warranted since the data met the following assumptions of no significant outliers, normality, and homogeneity of variance. As mentioned above, some extreme outliers were eliminated from the data, which revealed two emergent outliers (see Fig. 4). To ensure that these outliers were not significant, an ANOVA was run both with and without those outlying data points. Since the results were not significantly different, these outliers were determined not to be exercising a strong influence on the mean, and so the ANOVA results including the outliers will be discussed here. Histograms of the score distributions in each grade were relatively normally distributed, with skewness values falling between 1 and -1 for all groups. Finally, Levene’s test for equality of variances was found to be non-significant, F(2, 51) = 0.556, p > .05, justifying the use of a one-way ANOVA to evaluate the data.

There was a significant main effect of grade on block selection Solve It performance, F(2, 51) = 23.227, p < 0.001. Post Hoc comparisons using the Tukey-HSD test revealed that mean scores for Kindergarten students were significantly different from both first and second grade scores. However, first grade and second grade mean scores did not significantly differ from each other.

To further elucidate the differences in mean scores across the three grades, ANOVA analyses were also conducted on each individual Solve It task. Of the six tasks, scores from the two Easy tasks violated assumptions of normality and homogeneity. This could potentially be due to the fact that these distributions showed floor effects in their distributions. It is perhaps not surprising that many students would score close to a zero on the least challenging questions. Although the mean differences in scores for these two tasks seem to follow the patterns of the other Solve Its, these two tasks will be omitted from statistical analysis. The most difficult task (#6) was only administered to first and second grade students, as it assessed a content topic that was not covered in Kindergarten (see Table 1). This task was analyzed using an independent-samples T-test to determine differences in the scores of first and second grade students.

A one-way between-subjects ANOVA revealed a significant main effect of grade on scores for Solve It tasks #3 [F(2, 51) = 17.874, p<0.001)], #4 [F(2, 51) = 6.524, p=0.003)], and #5 [F(2, 51) = 12.008, p<0.001)] and Post Hoc comparisons using the Tukey-HSD test revealed that mean scores for Kindergarten students were significantly different from scores in first and second grade for all three Solve It tasks. However, first grade and second grade mean scores did not significantly differ from each other.

As previously mentioned, the most difficult Solve It task (#6) assessed a content topic that was not covered in Kindergarten (see Table 1). Since Solve It task #6 was only completed by two groups, an independent-samples t-test was conducted to compare scores in first and second grade students. There was a significant difference in scores for first grade (*M* = 2.38, *SD* = 1.025) and second grade (*M* = 1.26, *SD* = 0.806) students; t(33) = 3.594, p = 0.001. This means that second grade students significantly outperformed their first grade peers on only the most challenging Solve It task.

To determine the effect of gender on a student’s performance, an independent-samples t-test was conducted to compare Solve it performance in male and female students. There was no significant difference in boys’ scores (M=2.4, SD=1.13) and girls’ scores (M=2.28, SD=2.03); t (55)=-.282, p > 0.05. These results suggest that gender does not have an effect on Solve It scores. Specifically, our results suggest that boy and girl students did not perform significantly differently from each other on the Solve It Tasks. A two-way ANOVA was conducted that examined the effect of gender and grade level on Solve It performance, in case gender means varied by grade. There was not a statistically significant interaction between the effects of gender and grade level on Solve It task scores, *F*(2, 51) = 0.181, *p* > 0.05.

Discussion

Based on the results presented above, there are several conclusions to draw about students’ learning of the ScratchJr programming language, as well as a host of new questions to investigate about children’s programming capability.

Individual Solve Its ranged in difficulty from Easy to Medium to Hard, with two of each difficulty level administered to all grades except for Kindergarten, which only received one Hard task (see Table 1). Easy Solve Its were inconclusive, probably because of a floor effect (too many perfect scores). Although this makes it difficult to discuss the students’ differences in understanding, it is not surprising since the concepts covered in the Easy Solve Its were the ones mastered by all grade levels in the early pilot studies (Flannery et al 2013).

Conceptually, the two Easy Solve Its assessed knowledge of Look and Motion blocks. It is logical that children would quickly understand the purpose of these blocks, since they directly influence the movement and appearance of the character. For example, a block with a “move left” arrow makes the character move to the left, and it is easy for students to view this or even act it out themselves (with their whole bodies or hands, etc.) until they internalize and understand the meaning of the symbol (Lakoff & Núñez 2000).

Tasks of Medium and Hard difficulty levels showed statistically significant differences across the three grades, with a distinct correlation between higher grades and better performance on assessments. Kindergarten scored significantly lower than first or second grade on the two Medium tasks and the one Hard task (#5) that all children completed. On these three tasks, first and second grade students showed no significant difference in mean scores, meaning that the comprehension challenges facing Kindergarteners in completing those tasks were overcome by first grade. The Medium tasks focused on control flow (i.e. program structure) and timing, in the form of wait and speed blocks. It could be that these were more difficult for Kindergarteners because the effect that those blocks have on a character’s actions is less obvious than Motion or Looks blocks. For example, if a child drags a “move left” block onto the canvas and taps it, the cat on screen will immediately move to the left. The immediate cause-and-effect result strengthens the child’s understanding of the actions and consequences of the “move left” block. On the other hand, a child dragging a “set speed” block onto the canvas may tap it and observe no movement or change in the cat. This is because the set speed block actually affects the speed of other blocks in a program, and not anything inherent to the cat character. In order to see the effect of the speed block, a child would need to connect it to the front of a motion block or script of blocks. This may explain why even young children can master Motion blocks quickly, but understanding the Control Flow blocks requires more sequential and logical thinking.

Task #5 (a Hard task) similarly assessed two other aspects of control flow: repeating loops and triggering conditions. The multiple characters on the stage demonstrated a program in which one character touches another one, causing a chain reaction. The tricky part of this task is realizing that one character caused the other to move. While there is still a visual clue, it requires an abstract leap to imagine two programs that must interact. This is significantly more challenging than understanding the relatively concrete, one-to-one correspondence of a “move left” block with a character moving left across the screen. Evidently, that abstract thinking was not as challenging for either first or second grade as it was for Kindergarten.

Interestingly, second graders did significantly outperform first graders on the most difficult Solve It, task #6. This task focused on two unique (and complicated) control-flow features. One was message passing, which allows one character to trigger another with no visual indication during the program. For example, a cat can “call” a pig to action in the middle of the cat’s program, even if they are on opposite sides of the screen. This is a relatively abstract concept, because there is no obvious visual clue that the cat is calling the pig. The same move could technically be achieved with a wait block, but messages are ultimately more useful for controlling many characters and coordinated events within a single program. The other concept targeted in this Solve It task is the Repeat loop with a finite end. While the other Hard task required children to know about the “Repeat Forever” block, a finite repeat loop can actually be more complicated. It causes the programmer to think about how many times each action was completed, and the order of the actions within the loop that would cause it to start and stop where it does. After seeing that first and second grade students scored so similarly on other Solve It tasks, it may seem surprising that they showed significantly different scores on this one. After all, they received the same level of instruction about the two concepts tested, and they performed similarly on the other Hard task. However, it may make sense when one considers that finite repeat loops and especially message blocks are most useful for programming multiple characters and complex action sequences. The fact that more second graders understood the concept may indicate that they experimented with the blocks more, to create more complex and multi-staged projects. Indeed, field notes from the classrooms of first and second graders show that while both groups created inventive, story-driven projects, second graders were more likely to make pages with multiple plot stages or even game-like elements (i.e. characters and objects programmed to be interactive with a user). This may indicate more what Wyeth (2008) refers to as “purposeful, task oriented programming,” or programming with a goal, which is more sophisticated than simply building a program to experimentally observe its results. The difference in performance on the most challenging Solve It task could imply that developmentally, different processes happened between first and second grade that might enable the exploration of complex programming techniques, and more complete mastery of abstract programming concepts.

In summary, students generally performed on Solve It tasks in a manner consistent with what one would expect from the pilot trials of ScratchJr (Flannery et al 2013). Children at the Kindergarten level had more trouble with meta-level control flow blocks and coordinating multiple characters. First and second grade students found these concepts more accessible, and spent more time exploring complex systems of instructions and multi-step strategies to further develop programmed stories and even games. Again, these concepts map onto other areas of cognitive growth for this slightly older age, such as using and inventing rules for games, modeling problems visually and abstractly, and creating short narratives for different audiences (MA DOE 2011a, 2011b).

Limitations

In the course of implementing this programming intervention, there were several challenges to data collection and interpretation. One limitation was that, based on the paper-and-pencil method of administering the Solve It tasks, it was not possible to control for the contextual difference between programming with ScratchJr on a touchscreen tablet and manipulating paper cut-outs of programming blocks. Interestingly, using this format allowed children to express things not possible on the ScratchJr iPad app such as sequencing upside-down and sideways blocks, leaving different sized gaps between blocks (ostensibly to indicate a pause in the program), and mixing multiple characters into different portions of the program to indicate when a character should move. These alternative answers were not analyzed as they did not pertain to directly to ScratchJr programming knowledge, but rather showed approaches to creativity in problem-solving and exploration. However, these results will certainly influence later iterations of assessments designed by the lead researchers specifically to target creativity and in programming.

Another challenge was that the intent of the child when answering a Solve It task became a variable to account for in scoring Solve It. Intentions play a role in understanding several aspects of a child’s recognition and comprehension. What part of the animation caused the child to circle a given programming block or include it in their program? Did the child’s answer influenced by the test-like environment of Solve Its administration, or distracted and bored by the lack of shared work while working on answers? When did they feel confidence about answers, and when were they guessing? These questions are challenging for any test administrator, but in some cases it meant the difference between a very high and very low score on a Solve It task. For example, if a child circled an advanced block to solve a simple question, was it because she was attempting to use a sophisticated (and correct) strategy, or because she simply guessed at the answer? This issue was addressed as much as possible by implementing early pilot tests of all Solve It tasks and designing questions with limited solutions.

Future Work

Results from this study have inspired many questions for the researchers, mainly focused on how to better understand students’ intentions before and while they program projects in ScratchJr. This can be achieved by taking a qualitative approach to data collection and analysis. One possibility is the use of interviewing, in particular video interviewing, to ascertain how children themselves think about their own programming work. This method could be used to learn about students’ reasoning behind choosing specific programming blocks in their projects, and the sequence that they chose. Video interviewing could also be a useful tool in order to view and interpret the physical processes children use to think about programming in ScratchJr. By analyzing videos, researchers will be able to examine what features of the ScratchJr iPad app children touch, tap, and move around as they problem solve and debug their solutions, and more importantly, delve into their problem-solving and creation process when interacting with an computer programming interface.

Another next step for researchers in this area is to examine children’s in-rocess and completed code creations. It is important to look at children’s programming not just from the perspective of cognition and understanding of the language, but also with the aim of understanding children’s actual problem solving strategies and results while creating projects. This could include analyzing the types of programs children use and create, tracking the number of programs per project or per character in a single work, and examining the overall finished animation, game, or story from a qualitative perspective. Investigations of work that students actually create while exploring computer programming can shed light on more complex processes than simply recognizing straightforward coding challenges, and may reveal how students approach computer programming using systems thinking, multi-tasking, and creativity.

Conclusion

Children’s programming ability has been a new flourishing area for debate and education research, and this study contribute to the investigation of assessment and cognitive capabilities of young students exploring computer programming. The differentiated programming capabilities of Kindergarteners, first graders and second graders lends credence to the importance of considering the age and ability of a young child user when designing new educational technologies. The lack of differentiated performance between males and females in any age group also strengthens the argument that age and cognitive level are the major factors to examine when researching young children’s programming knowledge development. The results of this study should be viewed in the context of children’s logical and cognitive development. Children in our sample were able to observe a complete programmed animation and deduce, through reverse logic, the programming instructions necessary to create the same actions. This kind of reverse engineering is a skill that many classroom techniques seek to develop through color and size matching activities, rule-based games like hopscotch, and even daily tasks like learning a class schedule. This study supports the claim that computer programming provides a uniquely powerful and visual way to explore these abstract concepts of problem solving and logical reasoning.

Finally, this study holds implications for the development of assessment measures and curricular units for young children exploring computer programming. With any new educational technique, it is important to determine a match between the skill that children need to focus on, and the tool being employed to aid that development. The assessment rubric developed for this research was designed specifically to capture students’ ability to recognize and identify component programming instructions within a longer program. This tells us much about children’s ability to observe and deduce facts from programming, but many questions still remain. Are children still able to maintain their reverse engineering capabilities when remembering the sequence of programming instructions? Does high performance on an assessment of programming instruction recognition correlate with high complexity of creative programming endeavors? Do individual vs collaborative programming assessments result in different problem-solving approaches? It is time now for researchers to use what we already know about assessing young children, and apply that to the developmentally-appropriate examination of students’ thinking patterns, assumptions, and logical processes when engaging in computer programming. In this way, the curricular models used to introduce these concepts may be designed to enhance children’s understanding of computer programming concepts, without overburdening a young learner.

Acknowledgements

This project is supported by National Science Foundation grant DRL-1118664 and the Code-to-Learn Foundation. The researchers also thank the many teachers, school administrators, and children who contributed to the research described in this paper.

Sources

Bers, M. (2008). *Blocks to Robots: Learning with Technology in the Early Childhood Classroom*. Teachers College Press, NY, NY.

Bers, M. U. (2014). Tangible kindergarten: Learning how to program robots in early childhood. In Sneider, C. I. (Ed.), *The Go-To Guide for Engineering Curricula PreK-5: Choosing and using the best instructional materials for your students* (pp. 133-145). Thousand Oaks, CA: Corwin.

Boschman, F., McKennen, S., & Voogt, J. (2014). Understanding decision making in teachers’ curriculum design approaches. *Educational Technology Research and Development.* Advance Online Publication. doi: 10.1007/s11423-014-9341-x

Clements, D. H. (2002). Computers in Early Childhood Mathematics. *Contemporary Issues in Early Childhood, 3*(2), 160-181.

Department for Education. (2013). The National Curriculum in England: Framework document. London: The Stationery Office.

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. <http://ase.tufts.edu/DevTech/publications/JRTE-robot-hokey-pokey.pdf>

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). ACM, New York, NY, USA, 1-10. doi:10.1145/2485760.2485785

Gülbahar, Y. (2007). Technology planning: a roadmap to successful technology integration in schools. *Computers & Education, 49*(4), 943-956. doi: 10.1016/j.compedu.2005.12.002

Hudson, R. F., Isakson, C., Richman, T., Lane, H. B., & Arriaza-Allen, S. (2011). An examination of a small-group decoding intervention for struggling readers: Comparing accuracy and automaticity criteria. *Learning Disabilities Research & Practice, 26*(1), 15-27. doi: 10.1111/j.1540-5826.2010.00321.x

Kazakoff, E. R. (2014). Cats in Space, Pigs that Race: Does self-regulation play a role when kindergartners learn to code? (Unpublished doctoral dissertation). Tufts University, Medford, MA. Retreived from: http://ase.tufts.edu/DevTech/resources/Theses/EKazakoff\_2014.pdf

Kazakoff, E.R., & Bers, M.U. (2011). The Impact of Computer Programming on Sequencing Ability in Early Childhood. *Paper presented at American Educational Research Association Conference(AERA)*, 8 - 12 April, 2011, Louisiana: New Orleans.[Handout](http://ase.tufts.edu/DevTech/publications/aera%20handout%20sequencing.pdf)

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

Kazakoff, E.R. & Bers, M.U. (2014).  [Put your robot in, Put your robot out: Sequencing through programming robots in early childhood](http://ase.tufts.edu/DevTech/publications/Kazakoff%20Put%20Your%20Robot%20In.pdf).  *Journal of Educational Computing Research, 50*(4).

Kazakoff, E., 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. doi:10.1007/s10643-012-0554-5.

Lakoff, G., & Núñez, R. (2000). *Where mathematics comes from: How the embodied mind brings mathematics into being*. New York: Basic Books.

Lee, M. J., & Ko, A. J. (2012). Investigating the Role of Purposeful Goals on Novices' Engagement in a Programming Game. In: Proceedings of the *IEEE Symposium on Visual Languages and Human-Centric Computing*, Innsbruck, Austria, 163-166.

Lee, M. J., Ko, A. J., & Kwan, I. (2013). In-game assessment increases novice programmers’ engagement and level completion speed. In: Proceedings of the *ACM International Computing Education Research Conference (ICER)*, San Diego, California, 153-160.

MA DOE. (2011a). Massachusetts Curriculum Framework for English Language Arts and Literacy. Retrieved from http://www.doe.mass.edu/frameworks/ela/0311.pdf

MA DOE. (2011b). Massachusetts Curriculum Framework for Mathematics. Retrieved from http://www.doe.mass.edu/frameworks/math/0311.pdf

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

Pea, R. D. & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*, 137-168.

Robins, A., Rountree, J., & Rountree, N. (2003) Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13*(2), 137-172.

Starkey, P, Klein, A., & Wakeley, A. (2004). Enhancing young children’s mathematical knowledge through a pre-kindergarten mathematics intervention. *Early Childhood Research Quarterly, 19*(1), 99-120.

Strawhacker, A. L., & Bers, M. U. (2014). "I want my robot to look for food": Comparing children's programming comprehension using tangible, graphical, and hybrid user interfaces. *International Journal of Technology and Design Education.* Advance online publication. doi: 10.1007/s10798-014-9287-7

Strawhacker, A. Sullivan, A. & Bers, M.U. (2013). TUI, GUI, HUI: Is a bimodal interface truly worth the sum of its parts?. In Proceedings of the 12th International Conference on Interaction Design and Children (IDC '13). ACM, New York, NY, USA, 309-312.

US Department of Education, Office of Educational Technology (2010). Transforming American education: Learning powered by technology. Draft National Educational Technology Plan 2010. Washington, DC: Author.

Vanderlinde, R., van Braak, J., & Hermans, R. (2009). Educational technology on a turning point: curriculum implementation in Flanders and challenges for schools. *Educational Technology Research and Development, 57*(4), 573-584. doi: 10.1007/s11423-009-9126-9

Wyeth, P. (2008). How young children learn to program with sensor, action, and logic blocks. *Journal of the Learning Sciences, 17*(4), 517-550. doi: 10.1080/10508400802395069