

The Coding Stages Assessment: development and validation of an instrument for assessing young children's proficiency in the ScratchJr programming language

Laura E. de Ruiter & Marina U. Bers

To cite this article: Laura E. de Ruiter & Marina U. Bers (2021): The Coding Stages Assessment: development and validation of an instrument for assessing young children's proficiency in the ScratchJr programming language, Computer Science Education, DOI: [10.1080/08993408.2021.1956216](https://doi.org/10.1080/08993408.2021.1956216)

To link to this article: <https://doi.org/10.1080/08993408.2021.1956216>



Published online: 28 Jul 2021.



Submit your article to this journal [↗](#)



View related articles [↗](#)



View Crossmark data [↗](#)



The Coding Stages Assessment: development and validation of an instrument for assessing young children's proficiency in the ScratchJr programming language

Laura E. de Ruiter  and Marina U. Bers 

DevTech Research Group, Eliot-Pearson Department for Child Study and Human Development, Tufts University, Medford, USA

ABSTRACT

Background and Context: Despite the increasing implementation of coding in early curricula, there are few valid and reliable assessments of coding abilities for young children. This impedes studying learning outcomes and the development and evaluation of curricula.

Objective: Developing and validating a new instrument for assessing young children's proficiency in the programming language ScratchJr, based on the Coding Stages framework.

Method: We used an iterative, design-based research approach to develop the Coding Stages Assessment (CSA), a one-on-one assessment capturing children's technical skills and expressivity. We tested 118 five-to-eight-year-olds and used Classical Test Theory and Item Response Theory to evaluate the assessment's psychometric properties.

Findings: The CSA has good to very good reliability. CSA scores were correlated with computational thinking ability, demonstrating construct validity. The items have good discrimination levels, and a variety of difficulty levels to capture different proficiency levels. Younger children tended to have lower scores, but even first graders can achieve the highest coding stage. There was no evidence of gender or age bias.

Implications: The CSA allows testing learning theories and curricula, which supports the implementation of Computer Science as a school subject. The successful remote administration demonstrates that it can be used without geographical restrictions.

ARTICLE HISTORY

Received 4 December 2020
Accepted 8 July 2021

KEYWORDS

computational thinking; assessment; ScratchJr; design-based research

1. Introduction

Computer Science (CS) is now becoming popular in early education, in the US and worldwide. Educators and policy makers are introducing CS education starting in kindergarten, as evidenced by the release of learning standards and best practices for integrating technology into early childhood education (International Society for Technology in Education, 2007; NAEYC & Fred Rogers Center for Early Learning and Children's Media, 2012; Paciga & Donohue, 2017; Smith, 2016; U.S. Department of Education, 2010; U.S.

CONTACT Laura E. de Ruiter  laura.herbst@gmail.com  DevTech Research Group, Eliot-Pearson Department for Child Study and Human Development, Tufts University, 105 College Ave, Medford, MA 02155, USA

© 2021 Informa UK Limited, trading as Taylor & Francis Group

Department of Education & U.S. Department of Health and Human Services, 2016). To support these initiatives, a plethora of free resources have been developed that allow students to begin coding at a young age, for example, code.org, Scratch, ScratchJr, and Kodu (Yu & Roque, 2018).

However, at this point, there are few ways to evaluate whether initiatives and interventions for young children to learn CS are effective, and to gauge where children are in their developmental progression. In contrast, numerous validated assessments are available for evaluating children's progression in literacy development, for example. In order to develop effective early CS curricula and make evidence-based policy decisions, validated and reliable assessments are needed in the CS domain as well. To address this need, we have developed a new assessment, the Coding Stages Assessment (CSA), for the visual programming language ScratchJr, which is one of the most popular free programming languages targeted at children five to seven years old.

Existing coding assessments differ from the one presented here in two ways: 1) they are based on programming languages geared towards older children or teenagers; 2) they typically provide formative feedback rather than being standardized assessments that enable comparisons across different contexts and age groups. For example, Quizly (Maiorana et al., 2015) is based on the programming language AppInventor (Magnuson, 2010), and geared towards middle and high school students. Quizly has been developed as a tool to help teachers design problems for students and automatically compare student answers with model answers. As such, questions have to be developed by the teachers, and are not standardized. Another tool, Dr. Scratch (Moreno-León & Robles, 2015) automatically evaluates projects created in the programming language Scratch (Resnick et al., 2009). Scratch is aimed at children of at least eight years of age. Dr. Scratch can only evaluate what is provided by the user. This is different from an assessment that explicitly and purposefully tests different aspects of coding, in the same way that for example, reading assessments probe different aspects of reading ability (e.g. phonemic awareness, fluency, vocabulary) in an age-appropriate way. In addition to these tools, there are instruments that evaluate computational thinking (CT) rather than coding knowledge with a particular programming language. CT comprises the cognitive processes associated with breaking a complex task into simple components, creating reusable modules and working with abstraction (Wing, 2008; Wing, 2011). As we discuss in more detail below (Purpose and use of the CSA), coding involves computational thinking, but not vice versa. A coding assessment tests skills that are specific to particular programming languages, which each have their own grammar.

What is unique about the CSA is both the approach in which it is couched and its development and validation process: We combined design-based research with psychometric methods in order to create an instrument that builds on the pedagogical and design traditions of decades of work focused on children's learning to code (Bers, 2019, 2020) and that has good measurement properties. This paper presents the development and validation process of the CSA for the visual programming language ScratchJr.

In the remainder of the introduction, we first provide a summary of the Coding Stages theoretical framework and an overview of the programming language ScratchJr and its design features. Following that, we define the purpose of the CSA, describe the development of the instrument and its current content and format. We illustrate how we used an

iterative, design-based research approach to develop an engaging testing format for young children. In the following section, we then report the results of the first field test of the CSA with 118 children, using both Classical Test Theoretical (CTT) and Item Response Theory (IRT) methods. We conclude the paper with a discussion of the findings and future directions.

1.1 The coding stages framework

The Coding Stages framework is couched within the pedagogical approach of Coding as Another Language (CAL) (Bers, 2020). Most approaches view coding as a part of or an extension of STEM (Science, Technology, Engineering and Math) disciplines. Thus, coding is seen primarily as a problem-solving activity that engages children with abstraction and logic while developing computational thinking (Wing, 2006, 2011). In contrast, the CAL approach understands coding as an expressive activity and therefore links it to literacy (Bers, 2018a; Resnick, 2013). This perspective highlights coding as a means of expression for children, rather than as solely a problem-solving activity. Just as with a new language, the ultimate learning goal may not necessarily be simple technical proficiency, but rather fluency with a symbolic and grammar system to convey and interpret ideas through the making of projects (Hudson et al., 2011).

The Coding Stages framework, developed by Bers (Bers, 2018a, 2019, 2020), serves to describe a learning path in coding for young children, akin to the learning progressions and stages that have been defined in other areas of early childhood research and education such as mathematics (Clements & Sarama, 2004) and early literacy (Chall, 1983; Lonigan et al., 2008). This developmental progression starts with simple skills and progresses to more complex ones. Bers describes how just as children who are learning to speak do not start by uttering complete sentences, and children that are learning to read do not start by reading novels, children who are learning to code do not start by programming complex algorithms (Bers, 2019). In learning computer programming, children's journey might begin with understanding that computers and robots are human-engineered, and that there is an interface that uses symbols (without knowing what these are). Later on, they might learn that programming languages also use symbols and that they need to understand the vocabulary and grammar of the programming language to create projects. Through a combination of exploration and instruction, children learn how to use those symbols meaningfully in efficient and purposeful ways. As children master the programming language, they also develop (more) ways to think in computational terms.

The Coding Stages framework originally described by Bers comprises six stages capturing this developmental trajectory: Emergent, Coding and Decoding, Fluency, New Knowledge, Multiple Perspectives, and purposefulness (Bers, 2019). Later on, after empirical work was conducted with children, Bers' first formulation of the framework was revised: the last two stages were combined into just one stage (Purposefulness). Thus, the current version of the Coding Stages framework posits five stages (see Table 1 below) (Bers, 2021). The framework draws parallels with stages of literacy development. However, language and literacy development span an individual's lifetime, from early childhood (scribbling) to late adulthood (expanding vocabulary). In contrast, the coding stages focus only on early childhood, describing the typical

Table 1. Coding stages and the corresponding concepts children at that stage have mastered.

Coding Stage	Description
1. Emergent	<ul style="list-style-type: none"> • The child recognizes that technologies are human-engineered and are designed with a variety of purposes. • The child understands the concept of symbolization and representation (i.e. a command is not the behavior, but represents the behavior). • The child understands what a programming language and the purpose of its use is (knows that a basic sequence and control structure exists). • The child is familiar with the basics of the interface (turn the tool on and off and correctly interact). <p>This is a beginner's stage.</p>
2. Coding and Decoding	<ul style="list-style-type: none"> • The child understands that sequencing matters and that the order in which commands are put together generates different behaviors. • The child has learned a limited set of symbols and grammar rules to create a simple project. • The child can correctly create simple programs with simple cause and effect commands. • The child can identify and fix grammatical errors in the code. • The child performs simple debugging through trial and error. • The child engages in goal-oriented command exploration. <p>The most growth can be seen at this stage. Children learn the basics of the programming language and understand it can serve to create projects of their choice.</p>
3. Fluency	<ul style="list-style-type: none"> • The child has mastered the syntax of the programming language and can correctly create programs. • The child is personally motivated to create complex programs. • The child understands how to distinguish and fix logical errors in the code. • The child is beginning to be strategic in debugging. <p>This stage is characterized by the child moving from a "learning to code" to a "coding to learn" creative stance.</p>
4. New Knowledge	<ul style="list-style-type: none"> • The child understands how to combine multiple control structures and create nested programs that achieve complex sequencing. • The child engages in more goal oriented logical exploration with their programs. • The child is personally motivated to create complex programs. • The child is strategic in debugging and has developed strategies. • The child learns how to learn new commands or novel uses of the interface. <p>This stage is characterized by the child's ability to use their knowledge to create a personally meaningful project and if needed, acquire new knowledge on her own to meet the demands of the project.</p>
5. Purposefulness	<ul style="list-style-type: none"> • The child can skillfully create complex programs for their needs and purposes. • The child understands how to analyze, synthesize, and translate abstract concepts into code and vice versa. • The child is able to identify multiple ways to translate abstract concepts into code. • The child understands how to create programs that involve user's input. • The child can create multiple programs that interact with one another. • The child can debug multiple control structures. <p>This stage is characterized by the child being able to code in a rapid and efficient manner at high levels of abstraction requiring skill and flexibility and applying those skills to create a personally meaningful project. A child who reaches this stage has mastered all of the commands, grammar and syntax, of the programming language and has the ability to express herself through the project they create.</p>

development between four and seven years of age. Progression from one coding stage to another is independent of age, although a child's developmental level will influence how quickly they progress, as well as the kind of programming instruction the child receives. For example, while learning to code is possible by just tinkering and exploring with developmentally appropriate programming languages, in order to master complex skills there is a need of explicit teaching.

Bers (2019) describes how her choice of the term "stages" was influenced by Piaget's work on cognitive development stages (Piaget, 1963), but departs from it in an important way. While Piaget set out to explain a universal, naturally occurring phenomenon, Bers focuses on a taught skill. Coding stages describe a learning path that young children can

follow while learning to code with a developmentally appropriate programming language. The coding stages are levels or benchmarks of growth that represent distinct ways of mastering creative, expressive coding.

The developmental progression between coding stages is not always sequential, orderly, and cumulative. Coding stages are interconnected and not fixed or fully linear. Depending on the level of instruction received, and the degree of curiosity of a child to explore on her own, a child can jump stages quickly or never reach a particular stage. For example, using the ScratchJr programming language, children in the earlier stages might be able to put select motion blocks, but might not understand how to create an animation out connecting those blocks. Although the coding stages are not universal, they can be observed across different programming languages. The coding stages have some hierarchy based on syntax and grammar – mastery of simpler structures/commands (e.g. start/end) occurs before mastery of more complex structures (e.g. if statement or repeat loops). They thus capture computational thinking concepts such as representation, algorithms and modularization. Earlier stages are characterized by an understanding of representation and simple sequences (algorithms), later stages by an understanding of modules, more complex algorithms and the systematic use of debugging strategies. However, in line with the Coding as Another Language (CAL) approach, the Coding Stages framework also includes meaningfulness and expression at every stage, as well as in the final stage Purposefulness. Thus is, at each stage it is observed if and how children are able to use coding in expressive ways. [Table 1](#) presents the definitions for each of the stages.

While the Coding Stages theoretical framework applies to the ability of children to master computational ways of thinking and ways of expressing themselves with any programming language that is developmentally appropriate for young children, the CSA assessment is not universal, but specific to each programming language.

This paper presents the validation of the CSA instrument for the ScratchJr programming language, currently the most popular free introductory coding environment world-wide (Bers, 2018b; Bers & Resnick, 2015).

1.2 ScratchJr

ScratchJr is an introductory programming language for young children between five and seven, and currently the most popular free programming language world-wide (Bers, 2018b; Portelance et al., 2016; Sullivan & Bers, 2019) both in school settings and at home. At the time of the writing of this paper, ScratchJr was downloaded over 32 million times. It is available on different platforms and in different languages, and currently used in almost every country of the world.

In ScratchJr, children can create interactive stories and games by using graphical programming blocks representing different commands. They put together the blocks to make characters move, jump, dance, and sing. ScratchJr has a friendly graphical user interface with a main project editor, and tools for selecting and drawing characters, backgrounds and sounds.

Importantly, ScratchJr does not require children to be literate. All instructions and menu options are identifiable through symbols and colors. As shown in ([Figure 1](#)), at the



Figure 1. The ScratchJr interface.

center of the editor is the blue palette of programming instructions. One instruction category is selected at a time by clicking one of the selectors on the left. Instruction blocks are activated by dragging them from the palette into the scripting area below. The blocks snap together like jigsaw puzzles to create programs that are read and played from left to right. The “Green Flag” (“Play”) starts programmed animation, the red “Stop” interrupts it. The blocks are organized into six categories, which are represented by different colours: yellow Trigger blocks, blue Motion blocks, purple Looks blocks, green Sound blocks, orange Control flow blocks, and red End blocks. Through these programming blocks, young children learn the basic concepts and powerful ideas of coding while creating personally meaningful projects.

The design of ScratchJr was informed by the popular Scratch programming language for older children, eight years of age and up (Resnick et al., 2009). However, because ScratchJr is aimed at younger children, it differs from Scratch in important ways. It is designed to be a “coding playground” (Bers, 2018a) that reduces unnecessary low-level burdens, so programming can become another language of expression.

The approach of Coding as Another Language, which was described earlier, positions the learning of coding as a new literacy (Bers, 2019). As a new literacy, coding invites new ways of thinking and problem solving and enables the creation of computational interactive artifacts. ScratchJr’s design supports both the development of computational thinking and the ability for children to express themselves in creative ways by creating projects, without getting bogged down by frustrating syntax errors. One example of this design are the visual properties of the jigsaw puzzle pieces, which correspond to their

syntactic properties: The “Repeat Forever” block can only appear at the end of a program. Since nothing can follow a “Repeat Forever” command, the right side of this block is rounded so that no other block can be attached to that side, comparable to edge pieces in jigsaw puzzles. Design decisions like these keep the challenge at an appropriate level and help young children devote their cognitive resources to the many high-level thinking processes involved in imagining and creating a program.

Since the goal of ScratchJr is both expression (as in literacy) and computational thinking, an instrument able to assess children’s coding ability with ScratchJr needs to capture all of these dimensions.

2. Materials and methods

2.1 *The coding stages assessment*

2.1.1. *Purpose and use*

The main purpose of the new assessment presented in this paper was to develop a validated instrument that allows assessing children’s mastery of a particular programming language (in this case ScratchJr) in a reliable way. As such, it is distinct from already existing assessments that focus on the evaluation of Computational Thinking (CT) such as the Computational Thinking Test for Beginners (Zapata-Caceres et al., 2020) or TechCheck (Relkin et al., 2020). CT involves a range of analytical skills that are inherent to the field of CS, but applicable to many domains of life, such as thinking recursively, applying abstraction when figuring out a complex task, and using heuristic reasoning to discover a solution (Wing, 2006, 2011). CT is a cognitive ability, coding is the mastery of a particular programming language for problem-solving and creative expression, for which computational thinking is necessary. CT and mastery of a programming language are related, but they are not the same thing. CT can be measured in an “unplugged” way (Relkin & Bers, 2019), that is, without requiring coding knowledge, so that a lack of coding knowledge does not function as a barrier for a child to show their general ability to think computationally (see Relkin et al., 2020, for a discussion). In contrast, an assessment of the mastery of a programming language will always require a certain level of computational thinking, as the elements of the language reflect computational thinking concepts. For example, using the repeat block in ScratchJr necessitates some understanding of modularity.

In keeping with the CAL approach, the purpose of the CSA assessment is to capture not only children’s understanding of the “vocabulary” (i.e. the meaning of the various symbols) and the grammar of the language (i.e. in which way elements have to be combined to result in readable programs), but also their ability to use these skills to produce complex and meaningful projects. Using the analogy from natural language learning, children should not only be able to translate given sentences or spot grammatical errors but should also be able to use the language to communicate purposefully. In addition, 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 cross-domain skills (Lee & Ko, 2012; Lee et al., 2013; Wyeth, 2008). Thus, CSA offers children game-based tasks to solve.

2.1.2 Development process

The CSA is intended to be used both in educational and research settings. Thus, we used a development process that involved both working in a lab setting as well as school settings. The development process of CSA was first guided by design-based research (DBR), a dynamic and iterative methodology that employs cycles of intervention, analysis, and refinement of the intervention and the working theory of learning. Collins (1992) and Brown (1992) pioneered the notion of “design experiments” or “design-based research” with the aim of bridging the disconnect between education research and classroom practices. The paradigm was proposed in contrast to randomized control studies and in recognition that educational research must be rigorous and also be directly and immediately applicable to classrooms, teachers, and school systems (Barab, 2006; Barab & Squire, 2004; Van den Akker et al., 2006).

Over the past decade, the use of DBR as a research method has increased as it enables the linking of theory with practice (Anderson & Shattuck, 2012; Barab, 2006; Cobb et al., 2003). To a less extent, DBR has also been used to study the development of research instruments (Van Aalst & Chan, 2007). However, because of its dynamic nature, it is difficult to document the process of DBR using traditional measures of validity and reliability. However, as we show later, it is possible to assess the final product of DBR with respect to its validity and reliability, thus combining the strength of two approaches.

The DBR approach was adopted for the Coding Stages Assessment because it allowed us to refine both the Coding Stages theoretical framework as well as the early versions of the instrument as data was collected in classrooms. At a later phase of development, once the assessment was sufficiently piloted in classroom, we took it back to the lab setting.

The first iteration of the CSA was called ScratchJr Solve-Its (Strawhacker & Bers, 2015, 2018; Strawhacker et al., 2018) and was developed in two different versions. Version 1 involved the administration of Solve-Its at the end of the completion of a ScratchJr curriculum. Version 2 was integrated in the curriculum as formative measures of coding ability.

Solve-Its were designed to capture student learning in key areas of developing programming comprehension and were informed by qualitative results showing learning patterns from previous pilot studies of young children using ScratchJr (Flannery et al., 2013). Children were 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 of the ScratchJr programming blocks, they were asked to circle the blocks they believe were used to create the program they observed. Through this task, a child’s ability to observe the results of complete programs and reverse-engineer them from a larger set of isolated parts was required.

Solve-Its measured children’s programming ability using the following two metrics: block recognition and sequencing ability. However, they were not able to capture children’s ability to express themselves through programming. A total $N = 57$ Kindergarten through second grade children (27 male, 30 female) were tested with version 1 of ScratchJr Solve Its. Scoring rubrics and grading methods were based on prior research with robotics (Strawhacker et al., 2013). In summary, students generally performed well and consistently on Solve It tasks (Flannery et al., 2013). As expected, kindergarteners had difficulties with meta-level control flow blocks and with coordinating

multiple characters. First and second grade students found these concepts more accessible, and explored complex instructions and multi-step strategies.

There were issues observed in the administration format of Solve Its which might have impacted assessment. For example, researchers observed that some children were unable to maintain attention while the assessment was being administered which led to off-task play with the assessment materials as a diversion. Based on this experience, a new version of ScratchJr Solve Its was developed.

Version 2 involved students viewing a series of videos showing ScratchJr projects (e.g. characters performing actions) and asking them to respond to questions in multiple choice or fill-in format. These assessments could be administered to a large number of individuals at once, providing a flexible tool that could be integrated into the curriculum. Thus, the choice was made to divide the long assessment into chunks to be completed at different weekly intervals as the curriculum progressed. While this version was more engaging for children and ScratchJr Solve-Its captured children's ability to learn the syntax and grammar of the programming language, Version 2 not capture the children's ability to express themselves through programming. In addition, its integration with the curriculum, made it difficult to use in a variety of educational or research context.

While versions 1 and 2 were designed to be administered to an entire classroom at once to save time, in practice, teachers reported having a difficult time pacing them. They took longer than expected, as items could not be read aloud as a group until the preceding question had been completed by each student. Likewise, the group setting and read-aloud format inadvertently inhibited teachers' ability to prevent students from copying each other's work.

On top of that, the multiple-choice format presented its own problems. Coding is a complex construct that can yield multiple solutions to the same problem, yet the traditional forced choice format used in ScratchJr Solve-Its provided only one right answer. This design decision was due to the need to limit the number of choices offered to young children to avoid cognitive load. This format restricts students' capacity to express their thoughts and abilities to a finite list of options and does not shed light on students' design processes. In addition, seeing multiple, visually rich programs on the screen (or on paper) and having to compare them can be overwhelming especially younger children.

Based on this first iteration of version 1 and 2 of Solve-Its, we decided to move from group administration of multiple choice to one-on-one format involving interactive questions, as well as including opportunities to observe children purposefully creating their own programs. We also decided to change the name of the assessment as to not confuse teachers who still wanted to use the Solve-Its as a formative tool integrated into their curriculum.

The departure from a simple forced-choice format also presents an assessment style more fitting to the coding stages framework and the emphasis on coding as an expressive medium. The shift to a one-on-one setting allows for students' results to more closely reflect their individual progress.

2.1.3 Content and format

The CSA is an open question assessment. Questions are of two types. The first type of questions is verbal only. The administrator shows the child screenshots of the ScratchJr



Figure 2. A verbal question from the Emergent block. The child is shown this program and asked: “Look at this program. What is wrong with this program?”. Satisfactory responses include ‘it is backwards’, ‘the start needs to come first’, ‘the green flag needs to be at the beginning’.

interface and asks for example, why the shown program wouldn’t work (i.e. asking them to identify a bug) – see (Figure 2) for an example. The second type of question is task-based. The child is asked to complete a coding task in ScratchJr themselves, for example, making an object disappear (i.e. using control structures) – see (Figure 3) for an example. Each response is scored as either satisfactory or unsatisfactory. The questions are designed to be mapped onto each of the five coding stages (see above). That means that the questions are becoming increasingly complex with respect to the syntax used. In the case of the Purposefulness stage, they require creativity to solve them satisfactorily. More examples for questions from each stage can be found in the [Appendix A](#).

During testing with children, it became apparent that three questions out of the 30 were confusing to children, and they were removed from the assessment. As a result, there were 27 questions altogether: The first three stages (Emergent, Coding and Decoding, Fluency) had six questions each, New Knowledge had five questions, and Purposefulness four questions.

For the first four stages (Emergent, Coding and Decoding, Fluency and New Knowledge), the child needs to give at least four satisfactory responses out of six (three out of five in the case of New Knowledge) in order to move on to the next stage. We decided to use these criteria because we wanted children to be able to answer at least two thirds of the questions in a stage satisfactorily to be able to say that they have reached this coding stage.

If they give fewer than four satisfactory responses, the administrator will move directly to Purposefulness. An exception is the Emergent block. If a child does not receive at least three points in the first block, the assessment is stopped. In all other cases moving to



Figure 3. A task-based question from the Fluency block. The child is first instructed to recreate the program on their device. The they are asked: I want the Cat to go to the next page after moving. How can you change this program to make the Cat go to the next page at the end of this program?”. The response is scored satisfactory if the child correctly adds a page and changes the end block to go the next page.

Purposefulness allows the child to demonstrate their ability to create meaningful projects, even if their programming skills are still limited. Questions in the last block in particular give children more degrees of freedom in how to solve the problem. For example, in one question, children are instructed to program a “dance party”. This means they can choose the characters, the setting, but also how to implement “dancing” using the ScratchJr blocks they know. These questions test the children’s ability to translate an abstract idea (here the prompt) into concrete code.

A child receives a point for each satisfactory response. To calculate the overall score, points in each stage are weighted such that points received in higher stages receive more weight. That decision was made because the higher stages required more sophisticated knowledge. In addition, the weights for the last two stages (New Knowledge and Purposefulness) were adjusted to account for the fact that these stages had fewer questions than the others (five and four questions, respectively). Specifically, points in the first stage (Emergent) were multiplied by 1.1, points in the second stage (Coding and Decoding) by 1.2 and the scores in the third stage (Fluency) by 1.3. The scores in New Knowledge were multiplied with 1.68 and the scores in Purposefulness by 2.25. The weighted scores are summed to give the final score. The maximally achievable score is 39 points. The coding stage at which a child is in their development is then calculated using evenly spaced cut-off scores.

The CSA was designed to be administered one on one, either remotely or in person.

2.2 Field test

2.2.1. Participants

We tested the CSA with 118 children between five and eight years (mean age: 6.4 years; 24 Kindergarteners, 41 First Graders, 27 Second Graders, 26 Third Graders) recruited through the ScratchJr mailing list as well as schools who were working with ScratchJr or had plans to work with it. Of these, 67 were girls, 50 were boys, and one child preferred not to disclose their gender. The children had varying levels of familiarity and expertise with ScratchJr. Some had never worked with ScratchJr, others had participated in summer camps. However, none of the children had yet been exposed to a multi-week coding curriculum. Prior to testing, parental consent was obtained via an online consent form.

2.2.2. Materials

All children were tested on the Coding Stages Assessment version 1.0. Children either used their own iPad with ScratchJr installed or were given one by their teacher. In addition to the CSA, 23 children also completed the validated Computational Thinking assessment TechCheck (Relkin et al., 2020).¹ TechCheck is a platform-independent, “unplugged” CT assessment designed for children five to nine years of age. The term “unplugged” describes activities that can be used to teach programming principles without requiring the use of computers or actual coding (e.g. analyzing the ordered steps involved in brushing one’s teeth to illustrate the idea of an algorithm). In the context of CT assessment, an “unplugged” format allows administration to students who lack prior coding experience. TechCheck consists of 15 multiple choice questions that are designed to probe the “Seven Powerful Ideas” of computer science that are developmentally appropriate for young children (Bers, 2019) and it takes an average of 12–16 minutes to complete. We used TechCheck rather than another instrument for

measuring CT in children, the Computational Thinking Test for Beginners, BCTt (Zapata-Caceres et al., 2020), because TechCheck was available in English at the time of data collection, is targeting a narrower age range than the BCTt, and because it has a shorter administration time (an average of 15 minutes vs. 40 minutes for the BCTt). As discussed above, computational thinking ability is a necessary, but not sufficient condition for learning how to code. We therefore would expect there to be a positive relationship between children's ability to code, as measured by the CSA, and their computational thinking skills.

2.2.3 Procedure

The children were all tested remotely via Zoom, either from their homes or from their classrooms. The administrator and the child met in a Zoom conference room. The administrator shared their screen to show screenshots of ScratchJr programs for the verbal questions and the task-based questions in which the children had to recreate a program on their device. The administrator greeted the child and informed them that they were going to do some activities in ScratchJr, and asked to verify that their sound was working, and to alert the administrator if at any given point they were not able to see or hear them. To show their programs, children were asked to hold their iPad up to the webcam and run it. The administrator used a custom-made Qualtrics survey to enter children's scores. Administration took on average about 50 minutes (range = 1–125 minutes; standard deviation = ~21 minutes).

The children who completed TechCheck in addition to the CSA did this on a separate occasion. TechCheck administration lasted about eight minutes.

To assess interrater reliability, a subset of the CSA administration sessions ($N = 23$) was scored independently by a different research assistant from the one who had tested the children originally. The research assistant viewed video recordings of the Zoom sessions and scored the children's responses.

All materials and procedures, including obtaining parental consent, were approved by the Social, Behavioral, and Educational Research Institutional Review Board (IRB) of Tufts University under the protocols MOD-01-1,810,044 and MOD-03-1,105,019.

3 Results

As noted above, three questions were removed after about half of the children had been tested, as it became clear that the questions were taking very long and were also confusing to children. The results reported are the findings without those questions.

In this section, we first report descriptive statistics concerning the distribution of stages, average scores and distributions of stages and scores by grade and gender. After that, we report the psychometric analyses of the instrument. All analyses were conducted using R, version 3.6.2 (R Core Team, 2019) and R Studio Version 1.2.1335.

3.1 Descriptive statistics

The mean score was 12.25 points (standard deviation, $SD = 6.6$) out of a possible 39 points. The median was 12 points. Points achieved ranged from 0 to 39 points, thus spanning the

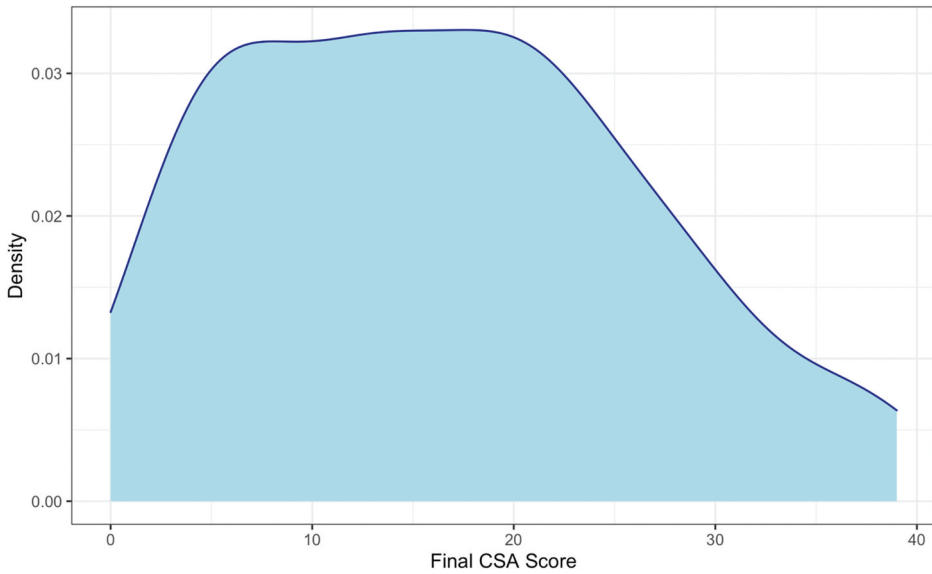


Figure 4. Density plot showing the distribution of final CSA scores. $N = 118$. Plot created using the “ggplot2” package in R (Wickham, 2016).

entire possible range. (Figure 4) shows the distribution in form of a density plot. The distribution is right-skewed, with more children achieving lower scores than children achieving higher scores.

This skew is also reflected in the distribution of Coding Stages (Figure 5). While 28 children were in the Emergent stage, only 11 children reached the Purposefulness stage.

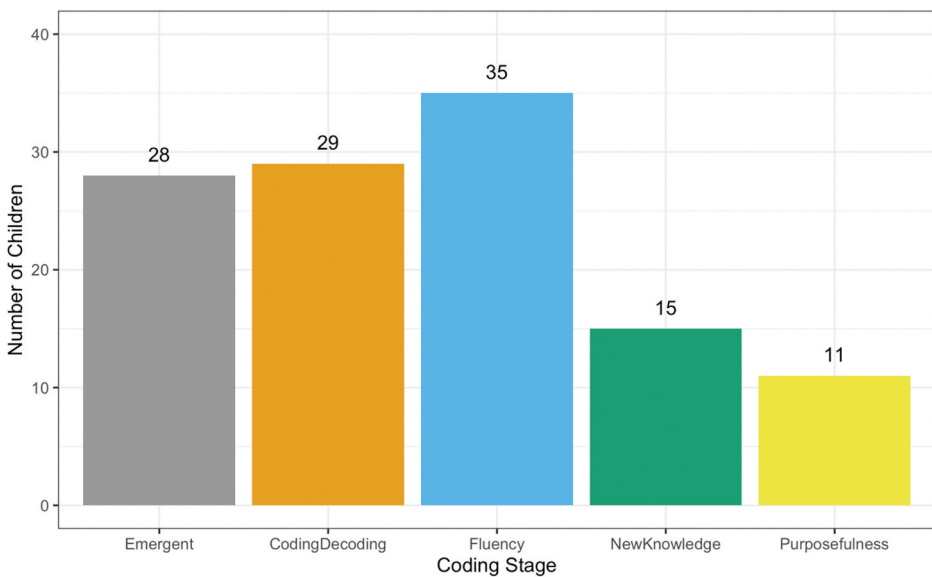


Figure 5. Bar plot showing the number of children in each Coding Stage. $N = 118$. Plot created using the “ggplot2” package in R (Wickham, 2016).

Table 2. Range of scores, mean scores, standard deviations, and highest achieved coding stage by grade.

Grade	N	Range	Mean	SD	Highest coding Stage
Kindergarten	24	0–19.2	7.8	5.6	Coding and Decoding
First Grade	41	2.2–37.7	14.3	9.1	Purposefulness
Second Grade	27	5.5–39	20.3	8.8	Purposefulness
Third Grade	26	3.3–37.3	24.3	7.4	Purposefulness

This is not surprising, given that many children had little to no experience with ScratchJr at the time of testing.

We also looked at the distribution of scores by school grade. As can be seen from [Table 2](#) and [\(Figure 6\)](#), both the minimum score and the mean score were higher for children in higher grades, and none of the Kindergarteners achieved the maximum score.

Notably, one third grader had a low score as well (3.3 points), and none of the third graders achieved the maximum score. The highest score among third graders was 37.3 points. ScratchJr is designed to have a low floor and a high ceiling, meaning that it is supposed to be easily accessible and understandable even for beginners and young children.

While boys' average score was one point higher than that of girls (see [Table 3](#)), the difference was not significant ($t = -0.49$, $df = 107.84$, $p = 0.624$). A non-significant result does not necessarily indicate that there is no difference, as non-significant results can also be due to small sample sizes. In contrast, Bayesian methods allow quantifying the evidence for or against the null hypothesis (i.e. no difference). We used a Bayesian t -test from the “BayesFactor” package (Morey et al., 2015) in R to do this. The results showed

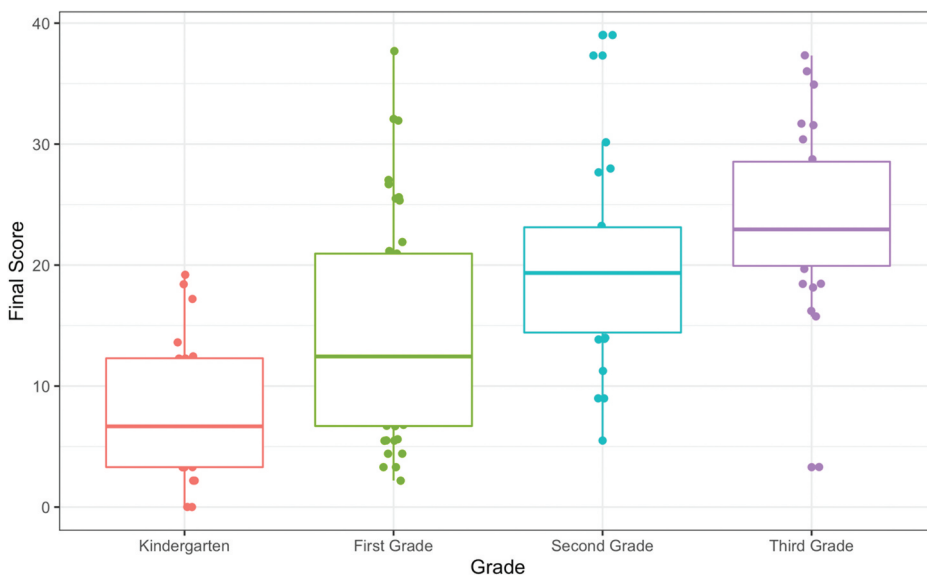


Figure 6. Box and whisker plot showing the distribution of CSA scores by grade. The box covers the interquartile range, the horizontal line in the box indicates the median, and the endpoints the highest/lowest values that are not outliers. $N = 118$. Plot created using the “ggplot2” package in R (Wickham, 2016).

Table 3. Mean CSA scores and standard deviations by gender. N = 118.

Gender	N	Mean	SD
Female	67	16.8	10.7
Male	50	16.98	9.69
Prefer not to say	1	28	N/A

that there was so-called anecdotal evidence for no difference between boys and girls (Bayes Factor = 0.23).

3.2 Psychometric analyses

We used both Classical Test Theory (CTT) and Item Response Theory (IRT) to analyze the CSA's psychometric characteristics.

3.2.1 Validity

We tested the CSA's concurrent criterion validity by correlating their CSA scores with their scores in the computational thinking assessment, TechCheck (Relkin et al., 2020). The CSA is a measure of children's coding ability with a specific programming language, ScratchJr. As described in the introduction, familiarity with programming languages is not a necessary prerequisite for thinking computationally, but we would expect children's coding ability to correlate with their computational thinking ability. This is because only if children have a good grasp of concepts such as representation, algorithms and modularity will they be able to make full and efficient use of a programming language. In other words, children with low computational thinking skills are unlikely to be good programmers. However, the correlation would not be expected to be perfect, as the CSA is assessing also children's knowledge of the ScratchJr vocabulary and grammar as well as expressivity.

The results show that there is a moderate positive correlation between children's CSA scores and their TechCheck scores ($r = .55$, $p < .01$; see Figure 7), indicating that both measures are tapping partially into the same construct (computational thinking).

3.2.2. Unidimensionality and reliability

The CSA is intended to measure one construct (latent trait), which is children's ability to program in ScratchJr. One index researchers have used to measure the dimensionality of an assessment is the inter-item correlation, which has been used in the development of a unidimensional assessment of coding ability before (Mühling et al., 2015). A range of other methods, such as principal component analysis or factor analysis have been put forward and discussed as well (Hattie, 1985). The mean inter-item correlation of the CSA was .24 (range: .07-.39). This puts the mean correlation in the ideal range between .20 and .40, which suggests "that while the items are reasonably homogenous, they do contain sufficiently unique variance so as to not be isomorphic with each other" (Piedmont, 2014, p. 3304). In addition to the inter-item correlation, we performed a confirmatory factor analysis on the different stages, which is described in the Appendix B. The factor analysis showed that for three stages (Coding and Decoding, New Knowledge, and Purposefulness) one factor was sufficient. For two stages (Emergent and Fluency), the

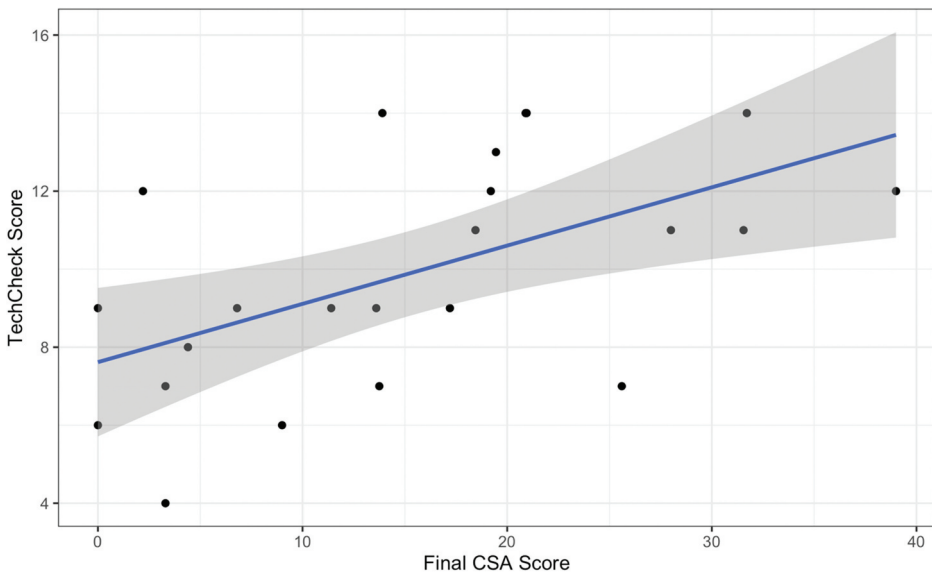


Figure 7. Scatterplot showing the relationship between children’s TechCheck scores and their CSA scores. The blue line is the regression line, the grey-shaded area indicates the 95% confidence level interval. $N = 23$. Plot created using the “ggplot2” package in R (Wickham, 2016).

factor analysis suggested that two factors were better. We return to this issue in the discussion.

The CSA’s internal consistency, measured using Guttman’s Lambda 6 (split-half reliability), was excellent with $\lambda_6 = .94$. Unlike Cronbach’s alpha, which increases as a function of the number of items in an assessment, Guttman’s Lambda 6 gives a more precise estimate as the number of items increases.

We measured inter-rater reliability for 23 children (559 individual questions, scored either “satisfactory” or “unsatisfactory” by two independent raters) using Cohen’s kappa. The agreement between the raters was substantial ($\lambda = .777$).

3.2.3 Item analysis

Recall that whether or not a child gets to respond to one of the items in the stages Coding and Decoding, Fluency, and New Knowledge depends on whether they answered a sufficient number of questions in the preceding stage(s) satisfactorily. This precludes the use of IRT analyses, which assume items to be independent from each other, for the assessment as a whole.²

In order to be able to give some impression of the items’ characteristics, we therefore conducted four different IRT analyses: one for the Emergent and Purposefulness items combined (explained next), and one each for the Coding and Decoding, Fluency and New Knowledge items. We combined Emergent and Purposefulness questions because of the large overlap between the samples: Of 118 participants, 102 were presented with both the Emergent item set and the Purposefulness item set. We then removed those 16 participants that had not been presented with the Purposefulness questions for the

combined Emergent/Purposefulness item set, which allowed us to compare the characteristics of the items in these two stages.

For all analyses, we used the “ltm” package (Rizopoulos, 2006) in R (R Core Team, 2019). For each set of items, we first fitted both a Rasch model (one-parameter model) and a 2PL model (two-parameter model) and then compared the model fit using a likelihood ratio test. Rasch models estimate the difficulty of individual items, 2PL models estimate both an item’s difficulty and its discrimination. For Coding and Decoding and New Knowledge item sets, the Rasch models were the better fit, for Emergent/Purposefulness and Fluency item sets, the 2PL models were the better fit. The results of these analyses can be illustrated using Item Characteristic Curves (ICC). These are S-shaped curves that show the probability of selecting the correct response for participants with a given level of ScratchJr coding ability. The x-axis represents the latent ability measured by the instrument (in our case ScratchJr coding ability); the y-axis represents the probability of selecting the correct response. Each item is represented by an ICC. The index of an item’s location is the point on the x-axis at which the curve crosses the 0.5 probability value on the y-axis. The more difficult an item is, the further this point is towards the higher end of the x-axis. In 2PL models, which also estimate item discrimination, the steepness of the curve indicates the question’s discrimination – the steeper the curve, the better the item is at discriminating participants with high ability from those with lower ability. It is desirable to have items of varying difficulty levels to accommodate participants with differing ability levels. Discrimination, on the other hand, should always be high. This is because items with low discrimination just make an assessment longer without providing any useful information about whether a participant has high or low ability. (Figure 8) shows the ICCs for the four different item sets. Note that it is not possible to compare difficulty *across* item sets, because the item sets were seen by different (sub-) samples of the participants. Still, the ICCs show that for the one combined item set, the Purposefulness questions have consistently higher difficulty indices than the Emergent questions, as intended by the design. Of the 12 items in that set, three (Q1.2, Q1.4 and Q1.5, i.e. items 2, 4 and 5) have shallow curves, indicating low discrimination, because almost all children who have minimal experience with ScratchJr were able to answer the initial questions correctly. These questions consequently also have lower difficulty indices.

In Coding and Decoding, the relative proximity of the ICC indicates that the items are of comparable difficulty levels. In Fluency, two questions (Q3.1 and Q3.6) have lower difficulty indices and have shallower slopes, indicating less good discrimination than the other questions. Both questions ask children to work with parameters (specifically with repeat blocks).

In the New Knowledge item set, one question (Q4.6) appeared to be relatively easy. Here children were asked to pretend that two characters are in different locations and create a telephone interaction between the two. This task required them to use the messaging blocks. We will return to these observations in the discussion.

We also conducted a Differential Item Function (DIF) analysis for gender and for age (younger children six years and younger, and older children) using ShinyItemAnalysis version 1.3.4 (Martinkova & Drabinova, 2018). Note that the child who preferred to not disclose their gender is not included in the gender analysis. A DIF analysis looks at whether there are any items that measure different abilities for different subgroups (e.g. gender, race, age group). If coding ability is assumed to be equally distributed among all

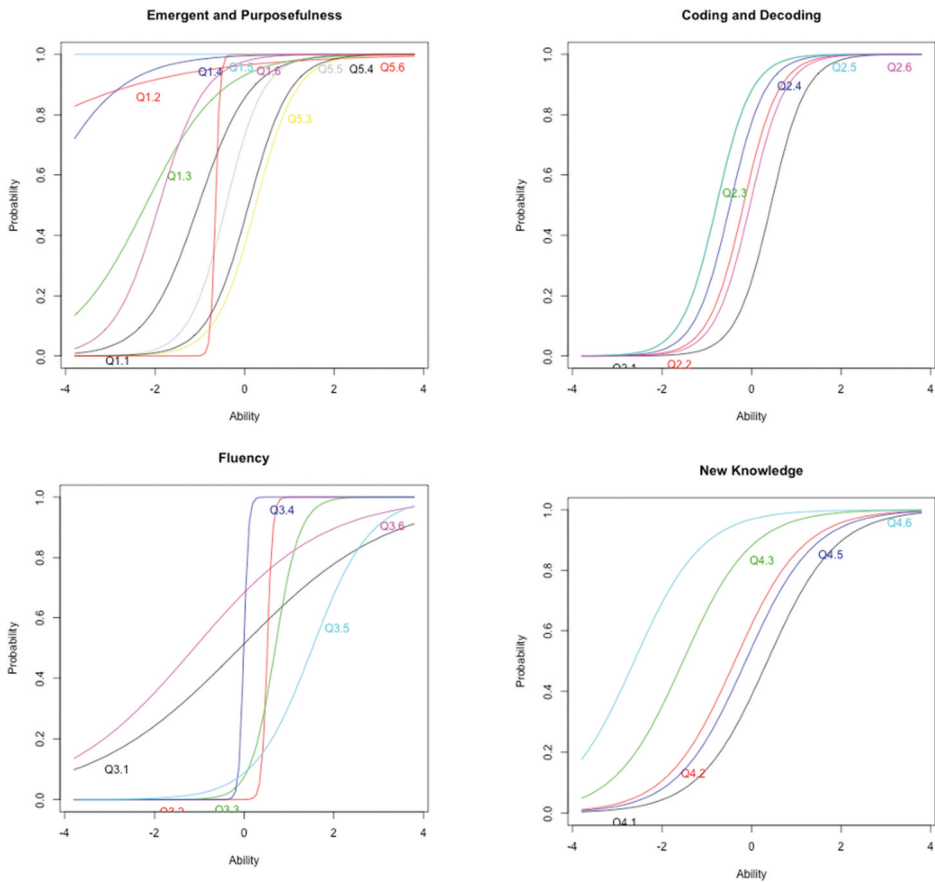


Figure 8. Item Characteristic Curves (ICC) for all 27 items (questions) in the CSA, grouped by sets of items that were presented to the same (sub-)sample of children. Note that the Emergent and Purposefulness and the Fluency ICCs are based on 2PL models, whereas the other ICCs are based on Rasch models. Plots created using the “Ictm” package (Rizopoulos, 2006).

children, irrespective of membership in a subgroup, then a child’s gender or age should not affect their probability of responding correctly to an item in the CSA. One popular method to identify potentially biased items is the delta plot method (Angoff & Ford, 1973). The method is using a scatterplot to visually compare item difficulty across different groups. (Figure 9) shows a delta plot of the CSA items for gender, and (Figure 10) shows the delta plot for age group. An item is “under suspicion of DIF” if its delta point departs considerably from the main axis. As can be seen from (Figure 9), item 5 (a question from the Emergent stage) appears to be more difficult girls compared to boys.

In (Figure 10), it appears that four items (2, 4, 5 and 27) are under suspicion of DIF for age group.

However, the delta plot method is suggested as a preliminary check before conducting further, more sophisticated DIF analyses. We followed up on the delta plots using the Mantel-Haenszel test (Mantel & Haenszel, 1959). For gender, no item was detected as functioning differentially. The DIF analysis thus shows that not only do boys and girls not score differently in the CSA overall, as shown above, in addition no

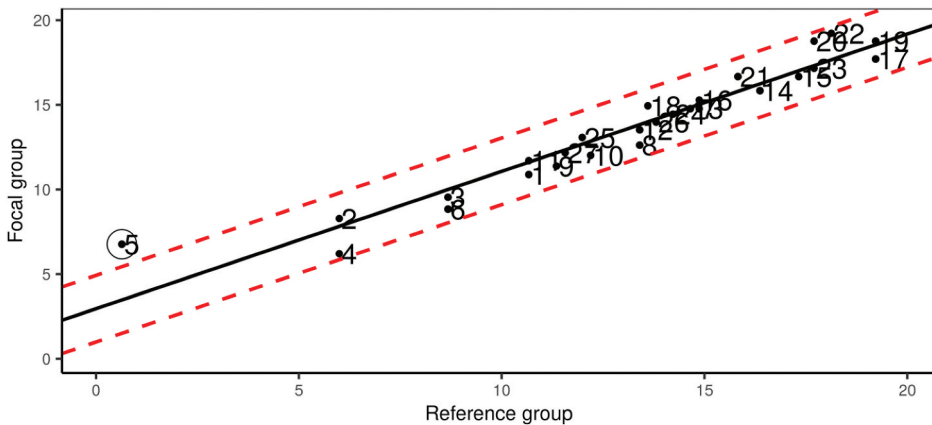


Figure 9. Plot showing the delta scores for each item for boys (reference group) and girls (focal group). $N = 117$. Plot created with ShinyItemAnalysis (Martinkova & Drabinova, 2018).

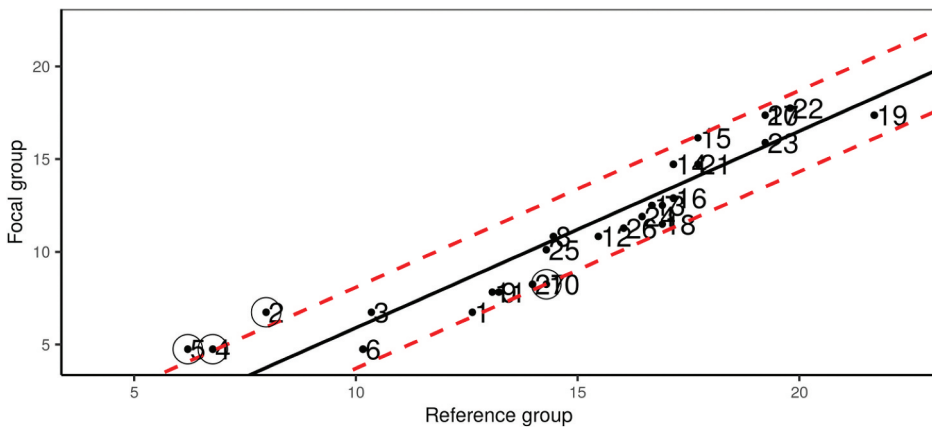


Figure 10. Plot showing delta scores for older children (reference group) and younger children (focal group). $N = 118$. Plot created with ShinyItemAnalysis (Martinkova & Drabinova, 2018).

individual item is biased against either gender. For age group, one item was flagged under the Mantel-Haenszel test as being biased against younger children: item 5 (Q1.4, Emergent stage). This question asks children to identify the one block in a given program that makes the character grow (see Figure 11). It's not immediately clear why this item may disadvantage younger children. A possible explanation may be that among children who do not know the answer, older children are able to guess it from looking at the symbols. They understand that the faded figures represent previous phases of a development, and that the development indicates a change in size. Younger children, on the other hand, may not be able to understand this and may be making random selections.

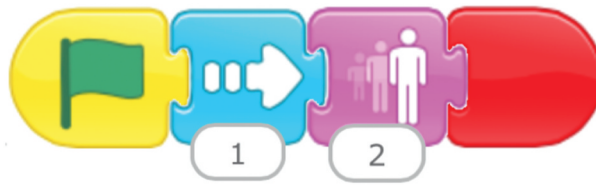


Figure 11. Item 5 (Q1.4) from the Emergent block. The child is shown this program and asked: “ Let’s look at this program! These blocks all help the Kitten move and get bigger. Can you show me the ONE block that tells Kitten to get bigger?”. Satisfactory responses include ‘the pink block’, ‘the grow block’, or pointing to the grow block.

4 Discussion

As coding becomes increasingly popular in early education, there is a need for reliable and valid assessments to gauge children’s learning. We have developed a new instrument that allows assessing young children’s coding ability in the visual programming language ScratchJr, the Coding Stages Assessment (CSA). The CSA is an interactive, developmentally appropriate assessment for children from Kindergarten (age five) through third grade (age eight). In line with the Coding as Another Language approach, the CSA captures not only the mastery of the ScratchJr syntax and grammar, but also the child’s ability to use this tool purposefully. The CSA assigns children to one of the five coding stages as laid out in the Coding Stages framework, which draws parallels with literacy development: Emergent, Coding and Decoding, Fluency, New Knowledge, or Purposefulness. In addition, the CSA provides a numeric score that allows for a more fine-grained quantification of skills.

We evaluated the CSA’s measurement properties in field test with 118 children. The results show that the CSA has good to very good psychometric properties. The analyses showed that the assessment has no gender or age bias, and that the level of difficulty is overall appropriate for the intended age group. In the Emergent stage, there were three items that had lower difficulty indices than the other items. This is intended by the design – we wanted to have some very easy questions in this block to avoid frustration for children with lower ability. Thus, while these items are easy and do not discriminate as well as the others, they have a function. Furthermore, the Emergent set of items still allows discriminating between children with very low ability and others, as 16 participants did not answer more than three questions correctly, which meant that the assessment was stopped after the first stage. In the Fluency stage, two items were easier and less good at discriminating than the others. As mentioned before, both questions involve the repeat block. This is an indication that in future developments of the CSA, it may be better to include different types of parameters (e.g. *grow*, *shrink*, *wait*). As for the easier item in the New Knowledge stage, we note that only 17 children progressed this far in the assessment and this question. More data is needed to make the estimate more accurate. Overall, the items in the stages are homogenous and the stages appear to be largely unidimensional, with the caveat that an additional factor was indicated for the Emergent and Fluency stage. As just mentioned, more data is needed to get more accurate estimates of the Fluency stage’s characteristics. For the Emergent stage, one possible explanation for the need for two factors may be that the Emergent stage captures some

very basic concepts which may be slightly removed from the core ability of programming, such as understanding that technologies are human-engineered or knowing where to find the program on a device.

Older children had higher scores on average, and a somewhat higher floor (i.e. the lowest scores in their group tended to be slightly higher than the lowest scores of the younger groups, although this did not hold for third grade). This is expected, as certain, more complex aspects of ScratchJr will be less accessible for younger coders. For example, even though ScratchJr is a visual programming language, there are textual elements such as parameters (e.g. number of repetitions of an action). Older children are more likely to have the literacy skills to fully understand that. That said, our results show that even first graders can reach the Purposefulness stage, and, conversely, that third graders without any experience with ScratchJr will be in the Emergent stage. In other words, the primary determining factor in children's outcomes is the ScratchJr proficiency, not their age.

The format and the administration are engaging, and children enjoyed the playful nature of the CSA, in particular the parts that allow them to realize their own ideas when they're asked to translate an abstract prompt into concrete code. These opportunities are essential for two reasons. First, we conceptualize coding as a means of expression, and as such children need to be able to demonstrate that in a comprehensive assessment of their coding ability. Second, preschool and young elementary school children have limited attention spans, so that it is imperative to have them enjoy the tasks such that they don't want to discontinue the assessment. Because of this open-ended nature of the CSA, both the administration and the scoring require some training. With respect to scoring, our analysis of a subset of sessions that were scored by independent raters shows that it can be scored reliably.

With an average administration duration of 50 minutes, the CSA is not a short assessment. It is an individual, in-depth assessment of children's coding ability. Large variation is to be expected, given that children vary in far into the assessment they get. Children who have never seen ScratchJr will finish in less than five minutes. Children with lots of programming experience and creativity may put a lot of effort into making their projects more complex, leading to sometimes long administration times. For example, children may want to add many different movements in their "dance" project or add a lot of detail to their backgrounds. While letting children go above and beyond what is required for a satisfactory response won't affect their score on that particular question, it adds to the overall duration and may lead to fatigue and potentially lower performance in the later questions. Administrators thus need to strike a balance between not dampening children's enthusiasm and keeping the assessment at a reasonable length. Learning to do this is one of the aspects that requires training, in addition to learning how to score children's responses. In its current format, the CSA should therefore not be administered without proper training. For use in educational settings, teachers and assessors would have to complete this training as well.

One challenge we noticed during virtual administration was that younger children need some practice of how to hold up the iPad and run the program so it can be seen by the administrator. However, this was not a major issue, and overall, the field test showed that virtual administration is possible. This is a positive finding, as we now know that there are no geographical restrictions on administering the CSA (provided that there is a stable internet connection).

Future versions of the CSA may include a mixture of closed, restricted option and open-ended questions to optimize administration time, while ensuring to maintain expressive aspects. Here, the design-based research approach will again help finding the appropriate formats. In addition, questions aimed at gauging expressivity are currently concentrated at the end of the assessment (in the “Purposefulness” block). In future versions, these may be distributed more equally throughout the assessment. Future work will use the CSA in studies that involve the teaching of the CAL curriculum to understand the rate of change before and after exposure to the learning condition. In addition, as with many other existing assessments, more work is needed to test the CSA with diverse populations (culturally, ethnically and socio-economically) to confirm that it is a suitable instrument across many different contexts. Finally, work is on the way to adapt CSA to other developmentally appropriate programming languages.

5 Conclusion

In this paper, we presented the development of a new instrument for assessing young children’s coding ability in the visual programming language ScratchJr, the Coding Stages Assessment (CSA). By using a combination two different research traditions, design-based research and psychometric methods, were able to develop a reliable and valid instrument for determining children’s stage in their coding development, as described in the Coding Stages framework. In this framework, coding is not seen as merely a problem-solving activity, but also as an expressive activity that allows children to create meaningful projects. We described how the assessment captures both technical skills in ScratchJr and expressivity through a combination of 25 open-ended verbal and task-based questions. We tested the instrument in a field test with 118 children between five and eight years of age. The test showed that the CSA is reliable and without age or gender bias, and that it can be administered remotely using video-conferencing software. Moderate positive correlations with a computational thinking test indicate that the CSA taps into computational thinking ability, which is a prerequisite for coding. We argue that the CSA fills a gap because it is the only summative assessment targeted at preliterate children. As such, it is suitable to be used in the evaluation of early computer science curricula, which are becoming increasingly more common world-wide. That said, assessment development is an ongoing process, and we expect the CSA to evolve further to ensure that it is a useful tool for the CS community.

Notes

1. Our goal had been to have at least half of all participants complete TechCheck. However, due to logistical challenges caused by the COVID-19 pandemic in 2020, only 23 children were able to take the test.
2. We thank an anonymous reviewer for pointing this out.

Acknowledgments

We would like to thank Riva Dhamala and Jessica Blake-West for their support in developing the CSA, Amanda Strawhacker and Madhumita Govindarajan for their work and feedback on previous

iterations of the instrument, and Jessica Blake-West for coordinating the data collection. Thanks also to all children, teachers and parents who made it possible to conduct this research in the middle of the COVID-19 pandemic.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the U.S. Department of Education under Grant number U411C190006.

Notes on contributors

Laura de Ruiter is a research assistant professor at the DevTech Research Group at the Eliot-Pearson Department of Child Study and Human Development at Tufts University. She studies language acquisition and cognitive development in young children. Her current research in developmental computer science focuses on the design and evaluation of interventions.

Marina Umaschi Bers is professor and chair at the Eliot-Pearson Department of Child Study and Human Development with a secondary appointment in the Department of Computer Science at Tufts University. She heads the interdisciplinary DevTech Research Group. Her research involves the design and study of innovative learning technologies to promote children's positive development.

ORCID

Laura E. de Ruiter  <http://orcid.org/0000-0002-6590-4073>

Marina U. Bers  <http://orcid.org/0000-0003-0206-1846>

References

- Anderson, T., & Shattuck, J. (2012). Design-based research: A decade of progress in education research? *Educational Researcher*, 41(1), 16–25. <https://doi.org/10.3102/0013189X11428813>
- Angoff, W. H., & Ford, S. F. (1973). Item-race interaction on a test of scholastic aptitude 1. *Journal of Educational Measurement*, 10(2), 95–105.
- Barab, S. (2006). Design-based research. In *The cambridge handbook of the learning sciences* (pp. 153–169).
- Barab, S., & Squire, K. (2004). Design-based research: Putting a stake in the ground. *The Journal of the Learning Sciences*, 13(1), 1–14. https://doi.org/10.1207/s15327809jls1301_1
- Bers, M. U. (2018a). *Coding as a playground: Programming and computational thinking in the early childhood classroom*. Routledge.
- Bers, M. U. (2018b). Coding and computational thinking in early childhood: The impact of ScratchJr in Europe. *European Journal of STEM Education*, 3(3), 3. <https://doi.org/10.20897/ejsteme/3868>
- Bers, M. U. (2019). Coding as another language: A pedagogical approach for teaching computer science in early childhood. *Journal of Computers in Education*, 1–30.
- Bers, M. U. (2020). Playgrounds and microworlds: Learning to code in early childhood. In *Designing constructionist futures: The art, theory and practice of learning designs*. MIT Press.
- Bers, M. U. (2021). *Beyond coding: How children learn human values through programming*. MIT Press.
- Bers, M. U., & Resnick, M. (2015). *The official ScratchJr book: Help your kids learn to code*. No Starch Press.

- Brown, A. L. (1992). Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *The Journal of the Learning Sciences*, 2(2), 141–178. https://doi.org/10.1207/s15327809jls0202_2
- Chall, J. S. (1983). *Stages of reading development*. New York: McGraw-Hill.
- Clements, D. H., & Sarama, J. (2004). Learning trajectories in mathematics education. *Mathematical Thinking and Learning*, 6(2), 81–89. https://doi.org/10.1207/s15327833mtl0602_1
- Cobb, P., Confrey, J., DiSessa, A., Lehrer, R., & Schauble, L. (2003). Design experiments in educational research. *Educational Researcher*, 32(1), 9–13. <https://doi.org/10.3102/0013189X032001009>
- Collins, A. (1992). Toward a Design Science of Education. In: Scanlon E., O'Shea T. (eds) *New Directions in Educational Technology* (pp. 15–22). Springer: Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-77750-9_2
- Flannery, L. P., Silverman, B., Kazakoff, E. R., Bers, M. U., Bontá, P., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. *Proceedings of the 12th International Conference on Interaction Design and Children - IDC '13*, New York, USA, 1–10. <https://doi.org/10.1145/2485760.2485785>
- Hattie, J. (1985). Methodology review: Assessing unidimensionality of tests and IteInls. *Applied Psychological Measurement*, 9(2), 139–164. <https://doi.org/10.1177/014662168500900204>
- 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. <https://doi.org/10.1111/j.1540-5826.2010.00321.x>
- International Society for Technology in Education. (2007). *Standards for technological literacy*. <https://www.iteea.org/File.aspx?id=67767&v=b26b7852>
- Lee, M. J., & Ko, A. J. (2012). Investigating the role of purposeful goals on novices' engagement in a programming game. *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Innsbruck, 163–166.
- Lee, M. J., Ko, A. J., & Kwan, I. (2013). In-game assessments increase novice programmers' engagement and level completion speed. *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, San Diego, 153–160.
- Lonigan, C. J., Schatschneider, C., & Westberg, L., & others. (2008). *Identification of children's skills and abilities linked to later outcomes in reading, writing, and spelling*. Developing Early Literacy: Report of the National Early Literacy Panel, 55–106.
- Magnuson, B. (2010). *Building blocks for mobile games: A multiplayer framework for App inventor for Android* [PhD Thesis]. Massachusetts Institute of Technology.
- Maiorana, F., Giordano, D., & Morelli, R. (2015). *Quizly: A live coding assessment platform for App Inventor*. *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, Atlanta, 25–30. <https://doi.org/10.1109/BLOCKS.2015.7368995>
- Mantel, N., & Haenszel, W. (1959). Statistical aspects of the analysis of data from retrospective studies of disease. *Journal of the National Cancer Institute*, 22(4), 719–748.
- Martinkova, P., & Drabinova, A. (2018). ShinyItemAnalysis for teaching psychometrics and to enforce routine analysis of educational tests. *The R Journal*, 10(2), 503–515. <https://doi.org/10.32614/RJ-2018-074>
- Moreno-León, J., & Robles, G. (2015). Dr. Scratch: A web tool to automatically evaluate Scratch projects. *Proceedings of the Workshop in Primary and Secondary Computing Education*, London, 132–133.
- Morey, R. D., Rouder, J. N., & Jamil, T. (2015). *BayesFactor: computation of Bayes factors for common designs*. <https://cran.r-project.org/package=BayesFactor>
- Mühling, A., Ruf, A., & Hubwieser, P. (2015). Design and first results of a psychometric test for measuring basic programming abilities. *Proceedings of the Workshop in Primary and Secondary Computing Education*, London, 2–10. <https://doi.org/10.1145/2818314.2818320>
- NAEYC, & Fred Rogers Center for Early Learning and Children's Media. (2012). *Technology and interactive media as tools in early childhood programs serving children from birth through age 8. Joint position statement*. www.naeyc.org/files/naeyc/file/positions/PS_technology_WEB2.pdf

- Paciga, K., & Donohue, C. (2017). Technology and interactive media for young children: A whole child approach connecting the vision of Fred Rogers with research and practice. In *Latrobe, PA: Fred Rogers center for early learning and children's media at Saint Vincent College*.
- Piaget, J. (1963). *La naissance de l'intelligence chez l'enfant* (Vol. 968). Delachaux et Niestlé Neuchatel-Paris.
- Piedmont, R. L. (2014). Inter-item Correlations. In A. C. Michalos (Ed.), *Encyclopedia of quality of life and well-being research* (pp. 3303–3304). Springer Netherlands. https://doi.org/10.1007/978-94-007-0753-5_1493
- Portelance, D. J., Strawhacker, A. L., & Bers, M. U. (2016). Constructing the ScratchJr programming language in the early childhood classroom. *International Journal of Technology and Design Education*, 26(4), 489–504. <https://doi.org/10.1007/s10798-015-9325-0>
- R Core Team. (2019). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. <https://www.R-project.org/>
- Relkin, E., & Bers, M. U. (2019). Designing an Assessment of Computational Thinking Abilities for Young Children. In Cohen, L. A., Waite-Stupiansky, S. (eds) *STEM for early childhood learners: how science, technology, engineering and mathematics strengthen learning* (pp. 85–89). Routledge.
- Relkin, E., de Ruiter, L., & Bers, M. U. (2020). TechCheck: Development and validation of an unplugged assessment of computational thinking in early childhood education. *Journal of Science Education and Technology*, 29(4), 482–498. <https://doi.org/10.1007/s10956-020-09831-x>
- Resnick, M. (2013). *Learn to code, code to learn*. EdSurge. <https://www.edsurge.com/news/2013-05-08-learn-to-code-code-to-learn>
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y., & others. (2009). Scratch: Programming for all. *Communications of the ACM*, 52(11), 60–67. <https://doi.org/10.1145/1592761.1592779>
- Rizopoulos, D. (2006). ltm: An R package for latent variable modelling and item response theory analyses. *Journal of Statistical Software*, 17(5), 1–25. <https://doi.org/10.18637/jss.v017.i05>
- Smith, M. (2016, January 30). *Computer science for all* [https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all]. Obama White House.
- Strawhacker, A., & Bers, M. U. (2015). “I want my robot to look for food”: Comparing Kindergartner’s programming comprehension using tangible, graphic, and hybrid user interfaces. *International Journal of Technology and Design Education*, 25(3), 293–319. <https://doi.org/10.1007/s10798-014-9287-7>
- Strawhacker, A., & Bers, M. U. (2018). What they learn when they learn coding: Investigating cognitive Domains and computer programming knowledge in young children. *Educational Technology Research and Development*, 67(3), 541-575. <https://doi.org/10.1007/s11423-018-9622-x>
- Strawhacker, A., Lee, M., & Bers, M. U. (2018). Teaching tools, teachers’ rules: Exploring the impact of teaching styles on young children’s programming knowledge in ScratchJr. *International Journal of Technology and Design Education*, 28(2), 347–376. <https://doi.org/10.1007/s10798-017-9400-9>
- Strawhacker, A., Sullivan, A., & Bers, M. U. (2013). TUI, GUI, HUI: Is a bimodal interface truly worth the sum of its parts? *Proceedings of the 12th International Conference on Interaction Design and Children - IDC '13*New York, 309–312. <https://doi.org/10.1145/2485760.2485825>
- Sullivan, A., & Bers, M. (2019). Computer science education in early childhood: The case of ScratchJr. *Journal of Information Technology Education: Innovations in Practice*, 18(1), 113–138. <https://doi.org/10.28945/4437>
- U.S. Department of Education. (2010). *Transforming American education: Learning powered by technology. National Educational Technology Plan 2010*. <https://www.ed.gov/sites/default/files/netp2010.pdf>.
- U.S. Department of Education, & U.S. Department of Health and Human Services. (2016). *Early learning and educational technology policy brief*. <https://tech.ed.gov/earlylearning>
- van Aalst, J., & Chan, C. K. K. (2007). Student-directed assessment of knowledge building using electronic portfolios. *Journal of the Learning Sciences*, 16(2), 175–220. <https://doi.org/10.1080/10508400701193697>

- Van den Akker, J., Gravemeijer, K., McKenney, S., & Nieveen, N. (2006). *Educational design research*. Routledge.
- Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. <https://doi.org/10.1145/1118178.1118215>
- Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366 (1881), 3717–3725
- Wing, J. M. (2011). Research notebook: Computational thinking—What and why. *The Link Magazine*, 20–23. Carnegie Mellon University's School of Computer Science.
- Wyeth, P. (2008). How young children learn to program with sensor, action, and logic blocks. *The Journal of the Learning Sciences*, 17(4), 517–550. <https://doi.org/10.1080/10508400802395069>
- Yu, J., & Roque, R. (2018). A survey of computational kits for young children. *Proceedings of the 17th ACM Conference on Interaction Design and Children*, Trondheim, Norway, 289–299.
- Zapata-Caceres, M., Martin-Barroso, E., & Roman-Gonzalez, M. (2020). Computational thinking test for beginners: design and content validation. *2020 IEEE Global Engineering Education Conference (EDUCON)*, [online], 1905–1914. <https://doi.org/10.1109/EDUCON45650.2020.9125368>

Appendix A

Below we provide additional examples of questions from each stage in the CSA. Interested researchers who would like to use the CSA can request access using this online form: <http://bit.ly/CSAScratchJr>

Emergent
Coding and Decoding
Fluency
New Knowledge
Purposefulness

Appendix B

As described in section 3.2.3, there is a lot of missing data, because not all children saw all questions. It was therefore not possible to conduct a single factor analysis on the entire data set. Instead, we conducted confirmatory factor analyses for each of the stages. For this, we used the “factanal” function in R. For each stage, we tested the hypothesis that one factor is sufficient to capture the dimensionality of the data, using varimax rotation. When the analysis reported a p-value smaller than .01, this meant that the null hypothesis (“one factor is sufficient”) should be rejected. We then increased the number of factors to two and tested again. According to the analyses, the stages



Figure A12. The first question of the CSA, a task-based question. The prompt is: “Using these blocks, make a program with all THREE blocks that will make Cat move”. For the response to be scored as satisfactory, the child needs to arrange the blocks correctly in this order: *start on green, move right, end*.



Figure A13. The third question in the Coding and Decoding stage, a task-based question. The scripting area is empty, the Cat is the only character on an empty stage. The prompt is: “Now, I want you to make a friend for the Cat so they can play together. Can you add a friend and then program the Cat move to its friend?”. For the response to be scored as satisfactory, the child needs to first use the Characters menu to select and add a different character from the library, and then create a program for this character that moves it to the Cat. This will usually just involve the *start on green*, *move right*, *end* commands, but other commands are also accepted, as long as the two characters end up close to each other.

Coding and Decoding, New Knowledge, and Purposefulness were captured by one factor (dimension). The Emergent stage and the Fluency stage needed two factors. Figures A17 and A18 show the factor loadings of the items on the two factors.



Figure A14. The third question from the Fluency stage, a verbal question. The child is shown a slide with the two different start blocks. The prompt is: “Let’s look at these blocks. What is the difference between these blocks?”. For a satisfactory response, the child needs to explain that *start on bump* makes the character’s program start after getting in contact (“bumping”) into another character and that *start on message* makes the character’s program start only if another character sent that same-colored message.

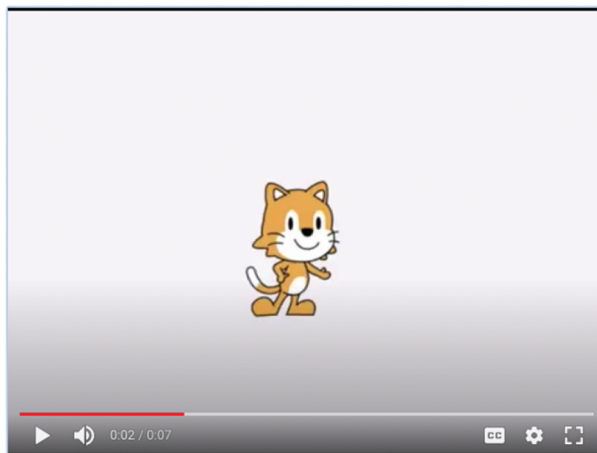


Figure A15. The fifth question from the New Knowledge stage, a task-based question. The child is shown a video of Cat moving to the right across the screen and jumping when tapped. The prompt is: “Program the Cat to forever move to the side. At the same time, when you tap on the Cat, program it to jump. After jumping, it continues to forever move right”.



Figure A16. The third question from the Purposefulness stage, a task-based question. The child is shown this slide. The first prompt is: “Looking at this picture, add the background with the bedroom and move the Cat to the bed and a friend by the door. It doesn’t have to be this character, it can be any character!” After the child has recreated the scene, the second prompt is: “Program the friend to send a message to Cat that helps Cat get out of bed using motion blocks”.

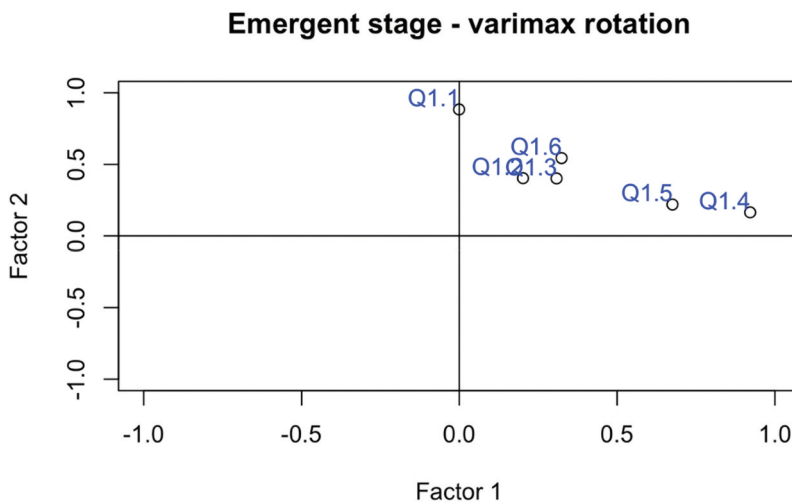


Figure A17. Scatter plot showing the factor loadings of each item in the Emergent stage.

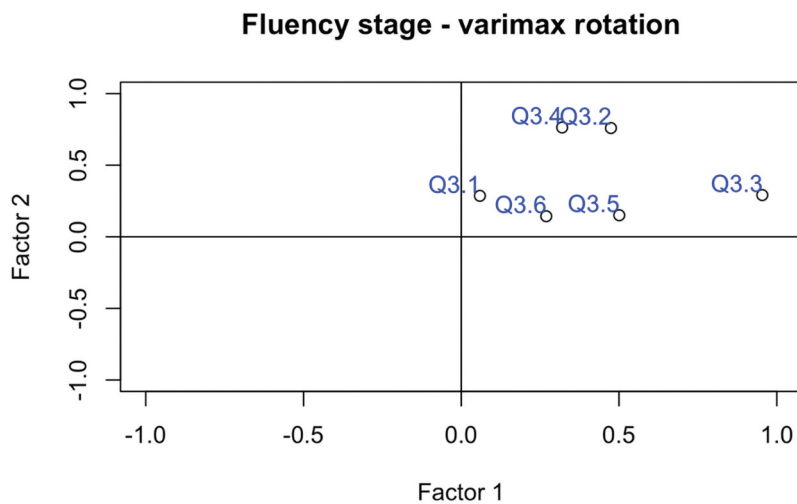


Figure A18. Scatter plot showing the factor loadings of each item in the Fluency stage.