Eliciting Student Scratch Script Understandings via Scratch Charades

Diana Franklin*, Jean Salac*, Cathy Thomas†, Zene Sekou*, & Sue Krause*
*University of Chicago, Chicago, IL
† Texas State University, San Marcos, TX, USA
{dmfranklin,salac,zene,sgkrause}@uchicago.edu;thomascat@txstate.edu

ABSTRACT

With many school districts nationwide integrating Computer Science (CS) and Computational Thinking (CT) instruction at the K-8 level, it is crucial researchers closely inspect the relationship between program expression and student understandings.

In this study, we propose and report on our use of Scratch Charades, a game in which students act out Scratch scripts while others build them. The purpose of Scratch Charades is to familiarize students with scripts and blocks without the cognitive overhead of the complex user interface. However, in this study, we also used it to elicit student understandings about Scratch blocks and scripts to design mnemonics to help students debug their code. We propose two building and/or debugging strategies based on our observations.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **Computational thinking**;

KEYWORDS

learning strategy, computational thinking, Scratch, elementary education

ACM Reference Format:
Diana Franklin*, Jean Salac*, Cathy Thomas†, Zene Sekou*, & Sue Krause*. 2020. Eliciting Student Scratch Script Understandings, via Scratch Charades. In *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20), March 11–14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366911

1 INTRODUCTION

Momentum has been building for integrating computer science into elementary school classrooms. Providing access to computing curricula is just one part of the solution. It is critical for computer science instruction to be effective for a broad spectrum of students. Diverse learners significantly underperform white peers with higher socio-economic status on important academic markers [4, 14, 15]. Recent work [18] has shown strong correlations between overall school academic performance and learning in a computer science curriculum built on open-ended projects designed using a Constructionist pedagogical approach [7].

Computational Thinking instruction in elementary school often uses Scratch, a visual block-based language with an integrated development environment[11]. Each command or instruction is called a block, and it can be dragged into code sequences called scripts. Each script begins with an event. For any Scratch learning experience, there are three distinct but related components: the Scratch language, the Scratch development environment, and the curriculum. In this paper, we focus on the Scratch language. In particular, this study has two purposes.

First, this study introduces our first experience with Scratch Charades, designed to allow students to learn the Scratch programming language outside of the Scratch development environment. While similar to existing methods of demonstrating how code works through human demonstration, it gamifies this to allow students to engage in the activity in a different way than a demonstration.

Second, observations of students playing the game were used to inform the design of instructional supports for struggling learners. In order to improve instruction for students who need more support, it is important to find out what aspects of Scratch are sources of confusion, defined as a mismatch between what students *think* a script should do and what a script *actually does*. If we can elicit students' early impressions of blocks and scripts, this can guide the development of activities, teacher materials, and debugging strategies. Activities can be developed that hone in on specific common issues to highlight how the scripts actually work. Teacher materials can highlight common student mistakes, allowing them to anticipate and recognize the source of specific problems. Finally, debugging strategies can be created that explicitly direct students to look for common mistakes. More precisely, we use Scratch Charades to answer this question: "For what Scratch blocks or code do novice students' interpretations diverge from the Scratch implementation? And in what ways?"

In this study, we use Scratch Charades, a Scratch-based Charades game, to elicit early interpretation of Scratch blocks and scripts. We focus both on students who act out given scripts and who construct the scripts for code they see acted out. Our experiments identified several common mismatches between student interpretations and Scratch implementation at a block, loop, and script level. And, while experienced instructors will be familiar with these mistakes, this identifies them in a methodical manner and uses those typical mistakes to create building / debugging strategies to help students.

The contributions of this paper are:

- The introduction and refinement of Scratch Charades, a formalized game students can play with each other with specific

SIGCSE '20, March 11–14, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6793-6/20/03.
https://doi.org/10.1145/3328778.3366911

student roles and materials to introduce new concepts prior to use in the Scratch programming environment.
- Identification of differences in interpretation of the meaning of Scratch blocks between students and Scratch.
- Identification of differences in interpretation of the meaning of Scratch repeat loops between students and Scratch.
- Mistakes that students make in building sequential and loop-based scripts.
- Creation of two building and/or debugging strategies based on the mistakes observed.

We find that some of the most common sources of confusion are:

- What `point in direction` does
- Which blocks' actions persist and which "reset"
- The timing relationship between consecutive blocks
- How to interpret multi-block repeat loops

The rest of the paper is organized as follows. We next present Scratch Charades itself, followed by background, related work (Section 3) and theoretical framework (Section 4). Section 5 contains the methods. In results, Section 6, we present both the observations and the debugging strategies created from those results. Finally, we describe future work and conclusions in Section 7 and 8.

2 SCRATCH CHARADES

A version of Scratch Charades has been played in classrooms around the world. As a demonstration, one person acts out a script in order to help them better understand the connection between the blocks and the actions that result from them. The difference between our formalized Scratch Charades game and typical similar demonstrations is that during demonstrations, the audience can see the script being acted out. In this game version, only the actor sees the script - the audience members are active game participants, using blocks to build the scripts they see in order to make explicit any lack of understanding.

Like in Charades, one person is responsible for acting while others attempt to guess what they are acting out. Two roles were defined: actor, whose goal is to correctly act out a given Scratch script, and builders, whose goal is to recreate the script they saw acted out using LEGO Scratch blocks. Students were placed into groups of three, with the a single actor and two builders each round. The actor role rotated each round.

In each round, the actor draws the top card from a stack of cards ordered by difficulty (see Figure 1). The actor acts out the script on that card. The actor has a "Hello!" say bubble card, a green flag card, and their body for acting out commands. Four signs are attached to the four walls of the classroom, with 0°, 90°, 180°, and -90° hung up to correspond with the four directions on the `point in direction` blocks.

The builders have a set of LEGOs, each with a Scratch block sticker on it. They build the script and ask the actor to check it. The builders continue until their script matches the one on the actor's card. Actors can act the script out multiple times in order to assist the builders.

Blocks chosen for Scratch Charades are shown in Table 2. These are specifically chosen because they are very useful in simple Scratch projects and they can be acted out (unlike change costume). However, some do involve concepts not yet covered in 4th grade, so we chose what we believed would be the simplest form of each action. For example, turning is very useful. There are two turns to choose from: Turn a number of degrees or point to an absolute location. Because 4th grade students may not have learned degrees yet, we chose point to and placed signs on the wall indicating the location. To make a sprite smaller, there are also two choices: change size by (negative number) and set size to (percentage). Because neither negative numbers nor percentages have necessarily been covered by 4th grade, we chose change size by with light instruction on negative being smaller and positive being larger.

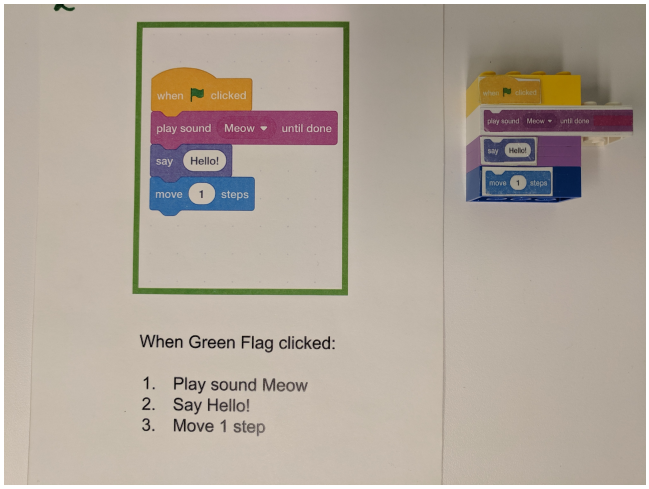

Figure 1: Scratch Charades card

3 BACKGROUND AND RELATED WORK

Computational Thinking instruction at the elementary-school level often uses Scratch, a visual block-based language with an integrated development environment. Scratch was designed with the philosophy of "low floors, high ceilings," which is intended to allow students with very little background to create projects quickly (low floors) but have the complexity to provide opportunities for deep learning through very complex projects (high ceilings). Designed to support a Constructionist philosophy [7], students in many different settings and age groups have successfully created projects with Scratch [17, 3]. We also know that some students are able to create projects but not yet understand the underlying code [1]. For any Scratch learning experience, there are three distinct but related components: the Scratch language, the Scratch development environment, and the curriculum.

For example, some challenges, like initialization, is affected by the programming environment itself, and the lack of a single "starting point" for a program as well as beginning in the same state as the prior execution ended [5] or the role of user input [20]. Some challenges may be conceptual, separate from Scratch, such as loops, sequence, or variables [6, 18, 20, 8]. Finally, the language itself can be challenging, with some blocks or aspects of the Scratch interface above grade level for students on the younger end of the recommended grade level [9].

In this study, we remove the Scratch programming environment in order to focus on the language itself and capture students' initial understandings through their actions.

4 THEORETICAL FRAMEWORK

In our theoretical framework, we draw from two bodies of work. First, we view the programming language as the learning interface for the student. Second, we utilize Scratch Charades as a window into students' thinking.

4.1 Language as Learning Interface

Programming language design has increasingly been viewed as an HCI endeavor, especially in the area of computer science learners. As Andy Ko posited in 2014, "Programming languages are the least usable, but most powerful human-computer interfaces ever invented" [10]. The Quorum programming language was specifically designed for the user, using empirical studies on novice students to choose syntax [19].

This view is supported by Constructivism, which posits that all learners interpret new knowledge through the lens of their own understanding [16]. This can sometimes be a benefit, utilizing prior knowledge to gain new knowledge, as with analogies. However, this can be a hindrance if the new construct is similar to existing knowledge but with important distinctions.

4.2 Manipulatives

Using manipulatives allows students to explore and express their thinking [13]. Diverse students may struggle to explain orally or express key ideas and concepts in writing [21, 22]. Manipulatives provide a response mode that is observable [2], thereby allowing students with limitations in expressive language, either oral or written, to show what they know. As teachers observe students' engagement with manipulatives, student thinking becomes overt, providing teachers with opportunities to scaffold instruction and enhance student learning. Manipulatives may also support students in recognizing and negotiating their own misconceptions [12].

Students	Grade(s)	Hisp	Af Am	Low-SES
14	4,5	N/A	91.6%	77.7%
24	4	95.6%	N/A	89.4%

Table 1: Participant School Demographics. The student column is the number of students who gave consent to be observed. Other information is at the school level.

5 METHODS

In this study, we used students playing Scratch Charades in order to elicit students' understandings or interpretations of Scratch scripts. In this section, we describe the study design, how observations were performed, and the analysis performed on the observational data.

move __ steps
point in direction __
change size by __
say __
play sound __ until done
repeat __
when green flag clicked
when sprite clicked
wait __ seconds

Table 2: Blocks included in Scratch Charades

5.1 Study Design

43 fourth and fifth grade students from two public schools in a major metropolitan school district participated in this study. The classrooms were drawn from schools with a high percentage of low-SES (Socioeconomic), underrepresented minority students, as shown in Table 1. There was no specific criteria students had to meet in order to be eligible for observation. In all five classes, all students were given the opportunity to participate but only those whose parents had signed consent forms were observed. Before the study took place, none of the students had instruction on Scratch in the classrooms. However, some students may have been introduced to Scratch in other settings.

5.2 Observations

Researchers served as observer-participants. That is, their main goal was to observe, but they also interacted with the students.

Observers focused on one or two groups and observed several types of interesting events such as interpersonal interactions, debugging strategies, and when actions did not match Scratch blocks. Researchers described each observation in prose, noted whether it related to the actor or builder, the script number, and details on the mismatch. Because observers were monitoring multiple groups, the list may not represent all instances of differences between student interpretations and script execution. Observers also explained concepts to students if the students did not self-correct their mistakes.

In this paper, we focus on two aspects of game play. First, we analyze the instances when the actor's actions did not match the script or when the builders' interpretation of the code did not match actors' actions. Second, we describe the revisions we have made to Scratch Charades as a result of these trials.

5.3 Analysis

Data was first filtered to retain only instances of mismatches between actions and scripts, removing observations about interpersonal interactions and debugging strategies. The first classroom of data, two researchers open coded the observations to create a coding scheme. A first-level categorization emerged, sorting observations into individual blocks, sequence, loop, and event categories. Block-level observations are misunderstandings about individual blocks, whereas sequence, loops, and event categories refer to misunderstandings about how blocks interact with each other. Instances within each category were further sorted and merged to create sub-categories. Once the coding was set, the two researchers

Figure 2: Some students would turn their bodies back to their original position before moving 1 step.

independently coded the second class of data, with a inter-rater reliability score of 93.10% (27/29). The final class was coded by a single researcher.

6 RESULTS

We present our results by high-level category - individual blocks, sequence, loop, and event categories. We present the counts to provide insight into how often different misunderstandings were observed. However, our study was not designed to imply that these are the rates at which misunderstandings of this nature occur in large populations.

Observation	Actor Instances	Builder Instances
`point in direction`		
Pointed but did not turn body	14	–
Did not hold position before next action	9	–
Pointed incorrect direction	4	–
`change size by`		
Did not hold position before next action	5	–
Mixed up -10 and 10	2	1
`other`		
Incorrect block	–	2

Table 3: Individual Block Observations

6.1 Individual Blocks

The two blocks that led to confusion were `point in direction` and `change size by`. The most common observation, with 14 instances, was that when acting the `point in direction` block, students pointed in the correct direction but did not turn their whole body to face that direction. The instructions directed the students to turn their whole body to model the behavior of sprites in Scratch.

Another common observation was that the student acting did not hold the action of the previous block before moving to the next block. For example, in the sequence in Figure 2, students would point to 90° and then turn back to the front of the room before moving one step. Instead, they should have pointed to 90° and taken one step *in that direction.* This occurred with the `change size by` block as well, in which students would "shrink" or "grow" as directed but then return to their normal height before the next action. In Scratch, some blocks have the format `do X for Y seconds` or `play __ until done`. Some students seemed to have attached this interpretation to the point and change size blocks even though it was not explicitly directed. It is worth noting that this misconception was only seen with these two blocks. For blocks such as `move 1 step`, there were no observed instances of a student stepping forward and then stepping back.

In the `change size by` and `point in direction` blocks, students also had difficulty interpreting the arguments. `point in direction` could direct the student to point to 0°, 90°, 180°, or -90°. `change size by` could direct the student to `change size by 10` (grow larger) or `change size by -10` (grow smaller).

Finally, there were two instances of builders using an incorrect block. In one case, the builders used a `move 1 step` block instead of the correct `change size by -10`. In the other case, a student used the block to close the repeat loop as a "stop," placing it at the end of the script.

6.1.1 Individual Blocks Discussion. There were three points of confusion related to blocks. We now provide suggestions on how instruction or activities that could help students with these concepts.

Meaning of point in direction. - in order to make clear that `point in direction` is rotation of the sprite, rather than pointing, a teacher could have students do a Scratch Charades activity, pointing out that you point with your nose, not with your finger. An alternative would be to use a 2-d picture without arms on a whiteboard instead of humans. They can have a discussion with students about what it means, with the four directions labeled on the four edges of the whiteboard. However, an important question is whether this misconception is *because* students acted out the blocks, and whether the interpretation would be clear within Scratch. We believe that this shows the imprecision of this command in general; it is not clear what it would mean for a sun sprite or a flower sprite to point in a specific direction. This is an instance in which the game *exposed* the misunderstanding, not *caused* it.

Change size by parameters. - in order to make it easier for students to act out different sizes, the class as a whole could decide on what "small", "medium", and "large" should look like. Then the teacher could explain that for this activity, change size by with a positive number would make the actor one bigger from where they began, and with a negative number would make them one smaller. They could practice this as a group, having the teacher say commands in different orders and having all students act it out together, one block at a time.

Resetting actions. - Activities could be created with different block sequences that depict blocks that do and do not reset each time, along with questions or observations that students can make. Students could then categorize instructions as those that stop automatically and those that need another instruction to stop (like `say __` vs `say for __ seconds`).

6.2 Sequence

Sequence relates to placing blocks together to form an ordered script. The majority of sequence errors were made by builders, with many instances of students placing the correct blocks in an

Observation	Actor Instances	Builder Instances
Out of order	3	9
Missing block	4	7
Concurrent actions	5	–
Opposite script order	–	2

Table 4: Sequence Observations

Debugging: Analyze your code to find A MESS!

Arguments: Check inputs in white circles

Missing: Check for missing blocks

Extra: Check for extra blocks

Scrambled: Check block order

Swapped: Check for wrong block

Figure 3: A MESS debugging strategy for sequence errors

incorrect order or leaving out a block. For scripts that were ordered incorrectly, the most common occurrence was that students switched the ordering of two blocks. For both categories, the errors did not appear to be related to a specific type of block. There were two instances of students building scripts with the correct sequence of blocks but starting from the bottom rather than the top. This could be due to the use of LEGOs for building the scripts as LEGOs are traditionally built from the bottom up.

Amongst the actors, there were also instances of incorrect ordering and missing blocks, although these occurred less frequently. Like with the building instances, there was no clear pattern to which blocks were involved in these errors. Performing two actions concurrently was the most common actor mistake. There were three observed instances of students making a sound and changing size at the same time. In another instance, a student changed sized while pointing to 180°. There was only one instance not involving changing size in which a student lifted the green flag and stepped forward simultaneously.

6.2.1 Sequence Discussion. These results have identified several different types of errors students can make in sequential code. The biggest misconception was the belief that two instructions could occur at the same time, which could affect how they build programs or expect them to work. The different sequential mistakes could be made explicit to students and presented as a debugging strategy to give students a more methodical approach and more concrete mistakes to look for when they encounter bugs in their own code. We propose A MESS (See Figure 3), which provides a checklist of common mistakes to look for in sequential code.

6.3 Loops

While students understood the general concept of loops, they had different interpretations of the loops syntax, as shown in Figure 4. The biggest source of confusion was when multiple instructions

Observation	Actor Instances	Builder Instances
Treated multiple items in repeat loop as multiple loops	5	4
Repeated block was outside loop	2	2
Did not repeat all blocks in loop	2	–
Bottom of repeat incorrectly / not placed	–	6

Table 5: Loop Observations

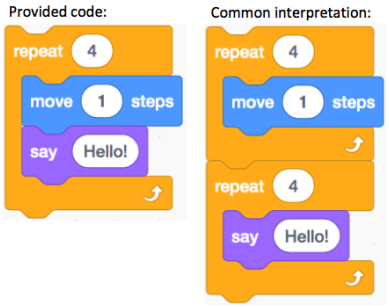

Figure 4: Common misinterpretation of loops

were placed within the loop. If the loop on the left of Figure 4 was the one given, the most common alternate interpretation by the actor was that the first instruction should be repeated followed by the second instruction being repeated rather than repeating the whole iteration. A similar (but opposite) phenomenon was seen in builders - when the actor acted out the loop correctly (with the two instructions alternated four times), some builders created separate loops.

Students also got confused about what instructions were inside versus outside the loop. Some students repeated instructions that were after the loop, and others repeated only the first instruction, effectively placing the second instruction out of the loop.

Builders had a challenge that does not exist in the interface with Scratch - the LEGO top and bottom of the repeat block were separate LEGOs, so they needed to place both of them for a single conceptual instruction. Many students forgot to place the bottom piece or placed it in the wrong location (above the beginning).

6.3.1 Loops Discussion. These results have identified several different types of errors students can make when constructing loops. Because of the complexity of loops, it might be worth not only presenting a debugging strategy but also having this be a building or planning strategy for loops. We propose Loop BASICs (See Figure 5), which provides a checklist of common mistakes to look for.

6.4 Events

Event misconceptions are those involving two Scratch event blocks included in this activity: `when green flag clicked` and `when this sprite clicked`. Both actors and builders had cases where a starting event was not included, or where the wrong starting event

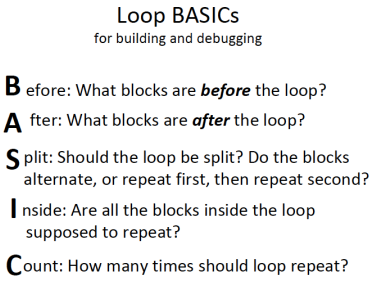

Figure 5: Loop BASICs building and debugging strategy for loops

was used. Instances where an event was not included possibly indicates a lack of understanding among the students that an event is necessary or, more generally, that actions in programs need something to trigger them.

6.4.1 Events Discussion. The original Scratch Charades rules do not properly handle events. In Scratch, a sprite does not do its own starting event. Instead, a user performs the starting event. In order to make the role of events more clear, we are changing the game play rules.

6.5 Scratch Charades challenges

We now present informal feedback from observers that was outside of the formal protocol but, nonetheless, led to revisions for Scratch Charades for this academic year.

Dexterity insufficient for LEGOs. Students had great difficulty taking LEGOs apart. Some figured out that if they placed the LEGOs in a stair-step fashion, it was easier to take them apart. Unfortunately, this counteracts the goal of making the scripts look like Scratch scripts. Scratch uses offsetting the blocks to visually represent placing blocks inside loops and conditionals (like most type-written, commercially-used languages). Using offsetting for sequential scripts removes meaning from offsetting blocks.

Building script upwards, not downwards. As described in sequence, one group built the script upwards, just as one would build a house in LEGOs. This results in a script that is the opposite order of a Scratch script.

Confusion about events. As described in the previous section, some students forgot about the event block or used the wrong one.

Heavy reliance on observers. Students interacted with observers more than anticipated. This most often consisted of helping students put together and take apart LEGOs as well as helping the actor act out blocks correctly.

As a result of these observations, we have made three major changes to Scratch Charades.

Magnets, not LEGOs. Instead of using LEGOs, we are using magnets. These are stiff enough to hold up to lots of play, can stick to the tins in which we store the pieces for easy building, and are easy to place and remove.

Added User role. There are now three different roles: Sprite, User, and Builder. We suggest groups of 4-5 students. At any given time, there is one sprite, one user, and at least two builders. This is for two reasons. First, this makes the role of the event explicit. Second, this allows every role to have a partner. The user and sprite can discuss the card and how it should be / was acted, and the builders can discuss how to build it. The hope is that less teacher intervention will be necessary if students can discuss.

Separate play sessions. The scripts for Scratch Charades have been split into two card sets - sequential cards and loop cards. This way, the sequential cards can be used prior to using Scratch the first time, and the loop cards can be saved until just prior to loops. Loops were substantially more challenging than sequential code. Splitting the cards would allow the introduction of loops to be delayed until students' concepts of sequential code were solid.

7 FUTURE WORK

There are two major areas for future work. The first is to experiment with Scratch Charades as a learning strategy to understand whether students learn Scratch more quickly or with less confusion if they play Scratch Charades prior to using the Scratch development environment. The second is to evaluate the use of the A MESS and Loop BASICs debugging / building strategies. Do students use these when looking for mistakes? Can they be used as a script to help each other, allowing them to collaboratively problem solve rather than needing individual help from the teacher?

8 CONCLUSIONS

We have found that while students understand the concept of sequential and loop-based code very easily, a few commonly-used Scratch blocks can lead to confusion, and students may not understand the details of loop syntax. In this paper, we presented interpretations observed in student enactments of simple Scratch scripts that diverge from Scratch implementation, along with proposed guidance for mitigating the common mistakes. We presented two debugging strategies that students can use, one for sequential errors and one for loop-based errors. Finally, we introduced Scratch Charades, a game that can be used to introduce students to these concepts.

Future work could evaluate the effectiveness of these two debugging strategies, as well as investigate the use of Scratch Charades as an educational intervention for getting students started in Scratch.

9 ACKNOWLEDGEMENTS

We would like to thank all of the teachers who opened their classrooms and the students who played Scratch Charades while we observed. This material is based upon work supported by the National Science Foundation under Grant Nos. 1660871, 1738758, and 1760055.

REFERENCES

[1] John B Biggs and Kevin F Collis. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome).* Academic Press, 2014.
[2] CAST. *Universal Design for Learning Guidelines version 2.2.* 2018. URL: http://udlguidelines.cast.org.
[3] *Community Statistics At a Glance.* URL: https://scratch.mit.edu/statistics/.

[4] Lara M Talpey Ludger Woessmann Eric A Hanushek Paul E Peterson. *The unwavering SES achievement gap: Trends in US student performance*. URL: https://www.hks.harvard.edu/publications/unwavering-ses-achievement-gap-trends-us-student-performance.

[5] Diana Franklin et al. "Initialization in Scratch: Seeking Knowledge Transfer". In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. SIGCSE '16. Memphis, Tennessee, USA: ACM, 2016, pp. 217–222. ISBN: 978-1-4503-3685-7. DOI: 10.1145/2839509.2844569. URL: http://doi.acm.org/10.1145/2839509.2844569.

[6] Shuchi Grover and Satabdi Basu. "Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic". In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '17. Seattle, Washington, USA: ACM, 2017, pp. 267–272. ISBN: 978-1-4503-4698-6. DOI: 10.1145/3017680.3017723. URL: http://doi.acm.org/10.1145/3017680.3017723.

[7] Idit Ed Harel and Seymour Ed Papert. *Constructionism*. Ablex Publishing, 1991.

[8] Felienne Hermans et al. "Thinking out of the Box: Comparing Metaphors for Variables in Programming Education". In: *Proceedings of the 13th Workshop in Primary and Secondary Computing Education*. WiPSCE '18. Potsdam, Germany: ACM, 2018, 8:1–8:8. ISBN: 978-1-4503-6588-8. DOI: 10.1145/3265757.3265765. URL: http://doi.acm.org/10.1145/3265757.3265765.

[9] Charlotte Hill et al. "Floors and Flexibility: Designing a Programming Environment for 4Th-6th Grade Classrooms". In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. SIGCSE '15. Kansas City, Missouri, USA: ACM, 2015, pp. 546–551. ISBN: 978-1-4503-2966-8. DOI: 10.1145/2676723.2677275. URL: http://doi.acm.org/10.1145/2676723.2677275.

[10] Andrew Ko. *Programming languages are the least usable, but most powerful human-computer interfaces ever invented*. 2014. URL: https://medium.com/bits-and-behavior/programming-languages-are-the-least-usable-but-most-powerful-human-computer-interfaces-ever-invented-7509348dedc.

[11] John Maloney et al. "The Scratch Programming Language and Environment". In: *Trans. Comput. Educ.* 10.4 (Nov. 2010), 16:1–16:15. ISSN: 1946-6226. DOI: 10.1145/1868358.1868363. URL: http://doi.acm.org/10.1145/1868358.1868363.

[12] S. C. Marley and K. J. Carbonneau. "Theoretical Perspectives and Empirical Evidence Relevant to Classroom Instruction with Manipulatives". In: *Educational Psychology Review* 26.1 (2014), pp. 1–7.

[13] B. M. Moskal and M. E. Magone. "Making Sense of What Students Know: Examining the Referents, Relationships and Modes Students Displayed in Response to a Decimal Task". In: *Educational Studies in Mathematics* 43.3 (2000), pp. 313–335.

[14] *NAEP Nations Report Card*. URL: https://nces.ed.gov/nationsreportcard/glossary.aspx#basic.

[15] *NAEP Nations Report Card - National Assessment of Educational Progress - NAEP*. URL: https://nces.ed.gov/nationsreportcard/?src=ft.

[16] Jean Piaget. "Piaget's theory". In: *Piaget and his school*. Springer, 1976, pp. 11–23.

[17] Mitchel Resnick et al. "Scratch: Programming for all." In: *Commun. Acm* 52.11 (2009), pp. 60–67.

[18] Jean Salac et al. "An Analysis Through an Equity Lens of the Implementation of Computer Science in K-8 Classrooms in a Large, Urban School District". In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. SIGCSE '19. Minneapolis, MN, USA: ACM, 2019, pp. 1150–1156. ISBN: 978-1-4503-5890-3. DOI: 10.1145/3287324.3287353. URL: http://doi.acm.org/10.1145/3287324.3287353.

[19] Andreas Stefik and Susanna Siebert. "An Empirical Investigation into Programming Language Syntax". In: *ACM Transactions on Computing Education* 13.4 (2013).

[20] Alaaeddin Swidan, Felienne Hermans, and Marileen Smit. "Programming Misconceptions for School Students". In: *Proceedings of the 2018 ACM Conference on International Computing Education Research*. ICER '18. Espoo, Finland: ACM, 2018, pp. 151–159. ISBN: 978-1-4503-5628-2. DOI: 10.1145/3230977.3230995. URL: http://doi.acm.org/10.1145/3230977.3230995.

[21] C. N. Thomas et al. "Applying a Universal Design for Learning framework to mediate the language demands of mathematics". In: *Reading and Writing Quarterly* 31.3 (2015), pp. 207–234.

[22] C. N. Thomas et al. *Applying a Universal Design for Learning framework to meet the language demands of science*. Leiden, 2018.