Article

Journal of Educational Computing Research
2023, Vol. 61(1) 178–207

DOI: 10.1177/07356331221102312
journals.sagepub.com/home/jec
SAGE

Types of Errors in Block Programming: Driven by Learner, Learning Environment

Anat Ben-Yaacov[1] and Arnon Hershkovitz[1]

Abstract

Block programming has been suggested as a way of engaging young learners with the foundations of programming and computational thinking in a syntax-free manner. Indeed, syntax errors—which form one of two broad categories of errors in programming, the other one being logic errors—are omitted while block programming. However, this does not mean that errors are omitted at large in such environments. In this exploratory case study of a learning environment for early programming (Kodetu), we explored errors in block programming of middle school students (N = 123), using log files drawn from a block-based online. Analyzing 1033 failed executions, we found that errors may be driven by either learners' knowledge and behavior, or by the learning environment design. The rate of error types was not associated with the learners' and contextual variables examined, with the exception of task complexity (as defined by SOLO taxonomy). Our findings highlight the importance of learning from errors and of learning environment design.

Keywords

block programming, computational thinking, student errors, error classification, log files, learning analytics

[1]Tel Aviv University, Tel Aviv, Israel

Corresponding Author:
Arnon Hershkovitz, School of Education, Tel Aviv University, P.O.Box 39040, Tel-Aviv 6997801, Israel.
Email: arnonhe@tauex.tau.ac.il

Introduction

Block programming has been suggested as a way to engage young children with the basic foundations of programming—and computational thinking at large—without the hassle of writing actual code (Sáez-López et al., 2016; Weintrop & Wilensky, 2015; Zhang & Nouri, 2019). Using a simple graphical interface, which is most commonly based on drag-and-drop interactions, learners can quickly and easily construct a computer program from basic blocks just like they would build a structure from little Lego bricks. Not surprisingly, the popularity of block programming has grown, and it is now used by millions of children and teens around the globe employing platforms like Scratch (https://scratch.mit.edu), Hour of Code (https://hourofcode.com), or Tynker (https://www.tynker.com). One of the prominent advantages—and promises—of using block programming is that learners are not prone to syntax errors, which characterizes novice and often also expert) programmers (Jackson et al., 2005; McCall & Kölling, 2015).

Errors may be beneficial for learners, in particular when they are responded with an effective corrective feedback, which allows students to reflect upon their attempts to solve a problem and to identify preconceptions (Metcalfe, 2017). Thus, error detection and error-driven teaching have been suggested as effective practices in teaching programming and computational thinking (Harrison & Hanebutte, 2018; Koehler, 2020; Nalaka & Edirisinghe, 2008). To support such methods, a deep understanding of novice programmers' errors should first be established; indeed, much has been studied in this context over the last decades (Chan Mow, 2012; Gladwin, 1987; Jackson et al., 2005; Ko & Myers, 2003; McCall & Kölling, 2015; Shooman, 1975). Despite the great importance of errors when learning to program, and despite the rising popularity of block programming, research in the field of block programming errors is still in its infancy. This is the gap we aimed to bridge in this exploratory research, that will serve as a first step towards furthering research in this field. We do so in a case study of a single learning environment for early programming (Kodetu).

Hence, the main purpose of the current study was to identify and classofy types of errors of young students in block programming, and to test for associations between these types and characteristics of the task and the learner. To meet this goal, we set up the following research questions:

1. What are the types of errors of young students in block programming and what is their distribution?
2. What are the associations between the types of errors and the following task characteristics?

a. Level;
b. Computational thinking concept;
c. Difficulty.

3. What are the associations between the types of errors and the following learner characteristics?

a. Gender;
b. Programming experience.

As this is—as far as we are aware of—the first study to focus on the identification and categorization of errors in block programming, we have no solid basis for setting up hypotheses for the types of errors, their frequencies, and their relationships with the various research variables. Therefore, we took an exploratory, bottom-up approach, using a large dataset of student errors. The rest of the paper is arranged as follows: In the section below, we will review the relevant literature about novices' errors in programming and about the ways in which block programming languages have been implemented in computer science education, and will lay down the theoretical framework for defining task difficulty in computer science education; we will then give full details about the methodological aspects of the reported study, followed by a description of our findings; finally, we will discuss the findings and their implications.

Literature Review

A computer program is a set of commands, written in a designated language, readable by a computer, and aimed at performing a specific task. Therefore, an error in programming normally takes one of two main forms: either preventing the computer from successfully running a code, or allowing it to run the code, however, with an undesired outcome. Broadly speaking, there are three common types of errors in programming: syntax, semantic, and logic (or logical) errors (Hristova et al., 2003; McCall & Kölling, 2015). Syntax errors are mistakes in the spelling, punctuation, and order of words in the program, e.g., using a capital letter when one is not needed, or omitting a semi-colon at the end of a command when it is required. Semantic errors refer to the meaning of the code, e.g., not declaring a variable before using it (in languages that require a declaration), or trying to assign a real number to a variable that is defined as integer only. The third type, logic errors, refers to mistakes in the logical structure of the program, e.g., using a wrong logical operation or calculation, sequencing operations in the wrong order, or repeating a command or a set of commands the wrong number of times.

Notably, syntax and semantic errors are strongly associated with the design specific to the programming language in use, hence their frequency is language-dependent (Grandell et al., 2005; McIver, 2000a), while logic errors are usually language-independent and have more to do with the programmer's reasoning. As our research is focused on block programming, which was originally introduced to reduce syntax and semantic errors, we are mostly interested in logic errors. The following subsections review the relevant background to logic errors and block programming.

Errors in Programming

Logic errors are the most common errors for novice programmers. Analyzing about 300,000 function implementations of learners of introductory programming MOOCs,

Smith and Rixner (2019) found that logic errors accounted for about half of the total errors in students' codes. Furthermore, while syntax and semantic errors usually cause compilation or runtime failure and maybe spotted and fixed using feedback from the development environment (Becker et al., 2016; Marwan et al., 2019; Qian & Lehman, 2019; Staub, 2021), logic errors are more difficult to debug, hence tend to remain in the code (Smith & Rixner, 2019).

Logic errors are commonly caused by faulty algorithmic design or with misconceptions regarding the very structure of programming tasks (Ettles et al., 2018). Therefore, reducing such errors usually involves engaging students with algorithmic thinking and with the fundamental concepts of programming, like the way loops are structured or the way variables represent specific values (Agbo et al., 2019; Grover & Basu, 2017). Brennan and Resnick (2012) defined such concepts—including sequences, loops, parallelism, events, conditionals, operators, and data—as the basic dimension of computational thinking which emerges when engaging with bock programming. Algorithmic thinking, in which problem-solving is done in a systematic, step-by-step manner, is considered as a cornerstone of computational thinking as a higher-order thinking skill (Shute et al., 2017). Some meta-cognitive practices are part of Brennan and Resnik's second dimension of computational thinking, specifically, being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing.

Therefore, identifying and correcting logic errors may involve meta-cognitive skills (Ginat & Shmallo, 2013), and following that notion, error-detecting has been suggested as a powerful pedagogical tool in programming teaching (Nalaka & Edirisinghe, 2008). Due to the unique cognitive and meta-cognitive skills involved in the process of debugging logic errors and due to the distinctive role these types of errors play in the field of computer science education, attempts have been made to focus on reducing as much as possible the burden of engaging with other types of errors. As a result, the use of block programming languages has become popular.

Block Programming

Learning to program is a complex task, which involves both general algorithmic thinking knowledge and particular knowledge of the specific programming language in use. In order to reduce the hassle of this task, visual programming languages—which have learners focus mostly on the logical foundation of programming—have been developed. In visual programming, users face a graphical interface that presents them with the available commands, with each command designed as a block (hence the term 'block programming'); using drag-and-drop, users can construct a program by creating a sequence (or sequences) of blocks. This way, writing a code is similar to building a Lego construction from individual blocks, and syntax errors are eliminated, as commands are a priori written and are capable of being joined together only when it makes sense (Kelleher et al., 2002; Maloney et al., 2010).

The most popular block programming languages are Scratch (https://scratch.mit.edu; developed in MIT Media Lab), Alice (http://www.alice.org; developed by the late

Prof. Randy Pausch of Carnegie Mellon University), and Google's Blockly (https://developers.google.com/blockly). Notable, block programming languages can be powerful tools for "real" programming; indeed, one can use them to build complex virtual or physical artifacts, e.g., with MIT's App Inventor (https://appinventor.mit.edu), BBC's micro:bit (https://microbit.org), or Lego Mindstorms.

Block programming, more than conventional text-based programming, may motivate students to learn "real" programming; furthermore, knowledge and experience gained with block programming may facilitate learning the more advanced material in text-based programming (Armoni et al., 2015; Ouahbi et al., 2015). As such, block programming languages have been suggested as a great means to promote learners' understanding of programming and of computational thinking at large (Fagerlund et al., 2021; Lye & Koh, 2014; Montiel & Gomez-Zermeño, 2021). Many learning environments have used block programming in a game-based design that adds some motivational aspects to the learning process (Tatar & Eseryel, 2019; Theodoropoulos & Lepouras, 2020).

Importantly, learning in block programming environments has been found to be associated with learner characteristics, in particular gender (Eguíluz et al., 2017; Funke & Geldreich, 2017; Hsu, 2014; Seraj et al., 2019)—however, not necessarily in terms of learning progress or success, but rather in different patterns of use or behavior—and prior coding experience, which was associated with either an appreciation of the block programming affordances or less enjoyment from practicing it (Bakali et al., 2018; Eguíluz et al., 2017; Menounou et al., 2019; Weintrop & Wilensky, 2015).

Certainly, using block programming languages does not fully eliminate errors, however, it keeps space mostly for logical errors (Socratous & Ioannou, 2020), which emphasizes the fact that algorithmic thinking is the core skill to be acquired while using such languages. As for the importance of learning from errors, for both students and teachers (Metcalfe, 2017; Rach et al., 2013; Tulis et al., 2016), it is important to understand which errors are most common in such programming languages. This is the focus of the current research.

Using Taxonomies of Learning in Computer Science Education

Taxonomies of learning are classifications of desired learning objectives or learning outcomes. Bloom's Taxonomy is probably the most known of them. The original taxonomy (Bloom et al., 1956) referred to six hierarchical levels of educational objectives sorted in increasing order by complexity, namely knowledge, comprehension, application, analysis, synthesis, and evaluation. Later, it was revised by one of the original taxonomy's designers, and it is now common to refer to the following six levels: remember, understand, apply, analyze, evaluate, and create (Anderson & Krathwohl, 2001). Both versions of Bloom's Taxonomy have been used in the field of computer science education, mostly for assessing learners' knowledge (Masapanta-Carrión & Velázquez-Iturbide, 2018). For example, Ullah et al. (2020) have defined the following six assessment criteria, based on the original Bloom's

Taxonomy: define the syntax of each structure used in a program (knowledge), explain each structure used in a program (comprehension), apply each structure used in a program as per the required output (application), breakdown the program by using nested structures or multiple structures of the same topic (analysis), synthesize the problem by using an appropriate structure of each topic of the program (synthesis), and, judge problem criteria and choose the most effective structure of the topic for problem solving (evaluation). Other proposed teaching and learning strategies, based on the revised taxonomy, are learning by typing (remember), learning by appreciating examples (understand), learning by modifying open sourced codes (apply), learning by partial coding (analyze), learning by debugging (evaluate), and learning by problem solving (create) (Lai et al., 2020).

Another common taxonomy of learning is the Structure of the Observed Learning Outcomes (SOLO) (Biggs & Collis, 1982). As its name suggests, SOLO Taxonomy classifies learning based on learners' responses. Assuming that learning becomes more complex as it progresses, this taxonomy refers to five phases. At its lowest level, it defines a pre-structural phase, in which a learner fails to demonstrate the required knowledge; next, during the unistructural level, a learner picks up only one or a few aspects of the task with which they are engaged; then, during the multi-structural level, they may pick up several aspects, however, without connecting them to each other; when integration of different aspects occur, the learner gets to the relational level; finally, when generalization and transfer of that acquired knowledge are achieved, the learner gets to the extended abstract level.

Just like Bloom's Taxonomy, SOLO is alo domain-independent and has been implemented in the field of computer science education. Whalley et al. (2011) have offered an implementation of SOLO taxonomy for the assessment of code writing; their framework includes the following stages that describe to students' codes: substantially lacking knowledge of programming constructs or is unrelated to the question (pre-structural), representing a direct translation of the specifications (unistructural), representing a translation that is close to a direct translation (multi-structural), providing a valid well-structured program that removes any redundancy and has a clear logic structure (relational), and finally, using constructs and concepts beyond those required in the exercise, that improve the solution (extended abstract). Since the development of that version of the SOLO Taxonomy, it has been used to assess students' solutions in programming in various contexts (Ginat & Menashe, 2015; Izu et al., 2016; Lister et al., 2006; Seiter, 2015). As our study is focused on classifying students' errors, that is, our data consists of students' learning outcomes, we preferred SOLO Taxonomy over others. Overall, as computational thinking and programming tasks involve cognitive processes that cover the full range described by SOLO, this taxonomy is suitable for studying such skills (Selby, 2015). Indeed, SOLO has been also used in the broader context of computational thinking, for designing and evaluation learning tasks (L. Lin et al., 2017; Parmar et al., 2022). Recently, Kaspersen et al. (2021) explicitly demonstrated how a computational thinking program can promote young learners across all the dimensions of Brennan and Resnick's (2012) framework by addressing

all levels of SOLO taxonomy. Importantly, Kasperson et al. emphasized that each of Brennan and Resnick's dimensions (concepts, practices, perspectives) could be promoted at each of SOLO levels (unistructural, multistructural, relational, and extended abstract).

Importantly, both Bloom's and SOLO taxonomies—although originally aimed at classifying learners' cognitive level or solution complexity—have also been used to classify learning tasks. This is usually done by mapping a task based on its desired solution, or as Petersen et al. (2011) put it, by "the characteristics of a response to which they would award full marks" (p. 632). This is similar to the case of the broader terms, "higher-order thinking" and "lower-order thinking", which traditionally refer to learner's skills or practices, and have also been used to characterize tasks based on the thinking level required to solve them (Haleva et al., 2021; Mohd Darby & Mat Rashid, 2017; Osadi et al., 2017; Rubin & Rajakaruna, 2015). We adopt this use, and will categorize the tasks in our studied learning environment based on SOLO Taxonomy; by doing so, we ignore the pre-structural level of the taxonomy, as each of the classified problems require some knowledge to correctly solve it.

Methodology

Our exploratory case study is based on a secondary analysis of log files drawn from students' use of an online learning environment for basic concepts in programming (Hershkovitz et al., 2019). Log files have been previously used to study errors in programming (Rodrigo et al., 2013; Seo et al., 2014; Thompson, 2006; Yarygina, 2020); mostly, such studies have referred to cases in which programs did not run completely successfully—either due to compilation error or run-time error—that is, generally focusing on syntax or semantic errors. Our approach is different, as we—while studying block programming, in which such errors are not possible—refer only to logic errors. Hence, our data include submitted codes that ran successfully but did not result in the expected outcome. Therefore, we had to manually classify the submitted codes for understanding the reason for not achieving the desired goal.

not true

Population and Data Collection

The data we analyzed were collected in April 2017 from a population of N = 123 primary school Spanish students, 10–12 years old (51% boys and 49% girls–63 and 60, respectively). About half of the participants did not have any previous experience with coding (63 of 123, 51%). The students arrived to an outreach activity organized by the Faculty of Engineering of the University of Deusto (Bilbao, Spain), and participated in a workshop about technology, programming, and robotics. During this workshop, the students used Kodetu for about 50 minute, engaging with a module consisting of 15 challenges, increasingly ordered by level of difficulty. Before playing this game, participants had completed a short online questionnaire, in which they self-reported on a few background variables (gender, age, coding experience).

Learning Environment: Kodetu

Kodetu (http://kodetu.org) is a web app built using Google's Blockly (https://developers.google.com/blockly) for teaching basic programming skills (Eguíluz et al., 2017). This platform has been recently used to assess the acquirement of computational thinking, as well as creativity in programming (Eguíluz et al., 2017; Guenaga et al., 2021; Israel-Fishelson et al., 2021). Each of Kodetu's challenges presents the user with a maze in which an astronaut should get to a marked destination. Guiding the astronaut to her destination is done via a block-based code which the user is editing. Moving to the next challenge is possible only upon completion of the current challenge.

In the module used by our participants there were 15 challenges, presenting the user with basic concepts of programming, namely, sequencing (Challenges 1–7), repetition (Challenges 8–10), and conditioning (Challenges 11–12). Within each concept, Challenges were sorted by an increasing order of difficulty. The Challenges were as follows: Challenge 1 introduced a very simple forward-path coding. Challenges 2–3 introduced a single rotation, using a "turn [right/left]" block, in different path points. Challenges 4–6 combined more than one rotation in different combinations, and Challenge 7 was a long maze intended to show the hard manual work required to lead the astronaut through the path step by step. Challenge 8 introduced the concept of loops, presenting the learners with a "repeat until destination" block, with the learners being limited to write a 2-block solution, in order to force them to use that block; code length was limited from that point on. Challenges 9–10 enhanced the use of loops. Challenges 11–12 presented the concept of conditionals, with a simple if-statement, using an "if path [ahead/to the left/to the right]" block, allowing to check whether a path existed before moving, and Challenges 13–14 introduced the concept of more complex conditionals, with an "if path [ahead/to the left/to the right]/else" block. Finally, Challenge 15 posed the classic problem of a general maze (difficult even for experienced coders). Those challenges are presented in Figure 1.

would that be the 3 rule problem?

Answer: yes

Classifying Kodetu Challenges Based on SOLO Taxonomy

For classifying Challenges based on SOLO taxonomy, we referred to both the complexity of the problem and of the solution. For doing so, we considered a few characteristics of the game Challenges and their expected solutions, specifically, maze complexity—in terms of length and turns required—and types and number of blocks in the expected solution. Following that, Challenges were classified to the four upper SOLO levels (Unistructural, Multistructural, Relational, Extended Abstract). Assuming that any Challenge of the game requires some knowledge, we did not use the Prestructural level. The classification was done based on the logic presented in Table 1.

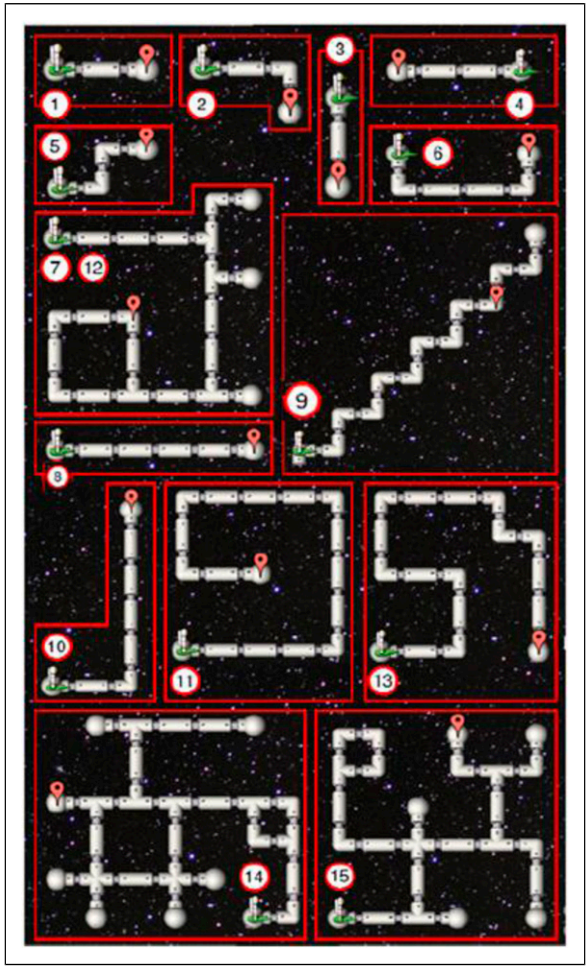

Figure 1. Kodetu challenges; figure taken from Eguíluz et al., 2017, p. 257.

Dataset and Preprocessing

The full log file included over 100,000 rows, each representing an action taken by a user, including its timestamp, the challenge in which it was taken [1–15], the code associated with this action, and a result indication. Logged actions included each block dropping in the editing area, either dragged from the blocks menu or from within the editing area. In cases where actions did not reflect code execution, the result field took the value "unset"; otherwise, it took one of the following values: "success" – the astronaut got to her destination; "failure" –the astronaut stopped before getting to her

Table 1. Classifying Kodetu challenges based on SOLO taxonomy.

SOLO Level	Maze Complexity	# Block Types	Solution Length	Classified Challenges
Unistructural	Straight path	1[a]	Very short	1, 8
Multistructural	Single path, includes a single turn (including a possible turn of the astronaut from her initial position)	2+ basic forward/ turn blocks	Short	2, 3, 4, 10
Relational	Single path, includes multiple turns	2+, may involve nested loop/ condition blocks	Long	5, 6, 9, 11, 13[b]
Extended abstract	Includes junctions, requires choosing the path to follow	2+, may involve nested loop/ condition blocks	Long	7, 12, 14, 15[b]

[a]The actual solution for that Challenge includes a "repeat until destination" loop block in which a single "forward" block is located; however, as the loop block is already located in the solution area and the user only has to drag the "forward" block into it—basically, there was nothing else the user could do—we considered this Challenge as a single block.
[b]Challenges 13–15 were eventually not included in our analysis due to the small number of students who completed them, see *Dataset and Preprocessing*.

destination; "timeout" – the astronaut got into an infinite loop; "error" – the astronaut fell off the path into the void. For the purpose of this study, we filtered out the logged actions that reflected non-executions, leaving us with 2679 total executions. This left us with 1033 failed executions (39%), that is, cases where the astronaut started moving and did not get to her destination.

It is important to highlight that our dataset included erroneous submissions only, that is, submissions that did not result in successfully complete the task. We chose to include successful submissions that may have been inefficient. For example, a code in which the sudent guided the astronaut to redundantly turn right and immediately left before following further steps. Such solutions are not considered as erroneous if the astronaut did get to her destination, and as was previously shown, ineffective solutions may have positive impact on student performance (Rangie et al., 2018)

Research Process

Error classification was done in a bottom-up approach while reading the codes submitted in the erroneous runs. This was done by the two authors in an iterative process and while keeping on full agreement. Each erroneous code could have included

multiple "atomic errors", where each one was classified into a single error category. After categorizing the errors observed in the log file, we ran statistical analyses to test for associations between the frequency of error types, the variables characterizing the task (level, computational thinking concept, difficulty) and the learner (gender, programming experience).

Findings

Types of Errors

does not apply to tasks 1-7

Classifying the erroneous codes resulted in seven types of errors, presented here in descending order by their overall frequency.

Decomposition (369 of 1033, 35.7%), refers to cases where the erroneous code was a result of decomposing the problem into smaller problems that were then wrongly recomposed, either by order or insufficiently. For example, in Challenge 11, a correct solution involves a "Repeat Until Destination" loop, an "If Path to Left" condition, and "Forward" steps, in the following order: "Repeat Until Destination (If Path to Left (Left), Forward)", the following erroneous solution would be classified under this category: "If Path to Left (Left), Repeat Until Destination (Forward)".

this probably could be avoided with RULER

Counting (288 of 1033, 27.9%), referring to cases where the number of the astronaut steps used in the code was smaller or larger than required. For example, in Challenge 2, where a correct code would be "Forward→Forward→Right→Forward", the following code falls under this category: "Forward→Right→Forward".

Repetition (276 of 1033, 26.7%), referring to cases where the code within a repetition block was wrong, or where the to-be-repeated code was not used within a repetition block. For example, in Challenge 9, where the loop should include "Forward→Left→Forward→Right", the following code falls under this category: "Repeat Until Destination (Forward→Left)". Note that loops were first presented in Level 8.

this probably could be avoided with RULER

Orientation (233 of 1033, 22.6%), referring to cases where the astronaut was guided to turn to a wrong angle. For example, in Challenge 3, where a correct solution could be: "Right→Forward→Forward", an erroneous code classified under this category is: "Left→Forward→Forward".

Premature (202 of 1033, 19.6%), referring to very short codes which formed the beginning of an expected solution. For example, in Challenge 4, where a correct solution could be: "Right→Right→Forward→Forward→Forward", the following code falls under this category: "Right→Right". These erroneous runs could be found by testing the solution step-by-step.

Conditionals (196 of 1033, 19.0%), referring to cases where a condition—or the code within it—was wrongly used. For example, in Challenge 11, where the condition should include "If Path to Left (Left), Otherwise Forward", a use of "If Path to Left (Forward), Otherwise Left" would fall under this category. Note that conditionals were first presented in Level 11.

residual

Redundant (56 of 1033, 5.4%), referring to longer-than-expected codes which were a result of keeping redundant blocks in the editing area. For example, in Challenge 3, where a correct solution could be: "Right→Forward→Forward", it is possible that a student had tried the following erroneous code: "Left→Forward→Forward" (which would be classified under "Orientation" below), and upon realizing their mistake, added a "Right" block at the beginning without omitting the "Left", to form the following erroneous code that falls under this category: "Right→Left→Forward→Forward".

Associations between Types of Errors and Task Characteristics

For each task, we computed the frequency of types of errors and then ran a few analyses to test for associations between these frequencies and level number, the difficulty of the task (based on SOLO taxonomy), and the computational thinking concept represented by that task.

Level Number. Level number represents learners' progression along the game. For each level, we calculated the frequency of types of errors, considering the total number of erroneous submissions in that level; this normalization was required in order to compare between levels that differ from each other by the total number of erroneous submissions. Then, we tested for Spearman's correlation between the level and frequency of the type of error. Findings are summarized in Table 2. As findings suggest, none of the error types is significantly statistically correlated with the level number, that is, there is no clear decreasing or increasing trend for any of them.

Computational Thinking Concept. Finally, we tested for correlations between the average of frequency of each type of error and the progression along the game by computationa thinking concept presented, while numbering concepts by their order of appearance (1 – sequences, 2 – loops, 3 – conditionals). No significant correlations were found. Findings are summarized in Table 3.

Task Difficulty. We continued by calculating frequencies of type of errors based on the four difficulty levels, as determined by SOLO taxonomy (see Table 1). Therefore, we averaged rate frequencies across levels of the same SOLO-based difficulty, and then tested for Spearman's correlation between frequency of type of error and SOLO-based difficulty levels, which were ranked from 1 (Unistructural) to 4 (Extended Abstract). Findings are summarized in Table 4. As findings suggest, Counting type of error demonstrates a marginally statistically significant decreasing trend, that is, the more difficult the level, based on SOLO Taxonomy, the less Counting errors are observed. On the contrary, three types of errors—namely, Premature, Conditionals, and Redundancy—demonstrate a marginally statistically significant increasing trend, that is, the more difficult the level, based on SOLO Taxonomy, the more Premature, Conditionals, and Redundancy errors are observed.

Table 2. Frequencies of error types by level of the game.

Level	Total	Decomposition	Counting	Repetition	Orientation	Premature	Conditionals	Redundancy
1	50	0.00	1.00	—	0.00	0.00	—	0.00
2	141	0.88	0.08	—	0.40	0.11	—	0.00
3	66	0.50	0.33	—	0.45	0.15	—	0.00
4	87	0.09	0.72	—	0.16	0.11	—	0.08
5	39	0.72	0.00	—	0.10	0.21	—	0.05
6	78	0.49	0.45	—	0.35	0.32	—	0.03
7	202	0.31	0.35	—	0.29	0.35	—	0.13
8	14	0.00	0.00	0.86	0.00	0.14	—	0.00
9	29	0.10	0.38	0.48	0.41	0.21	—	0.00
10	55	0.53	0.02	0.93	0.05	0.02	—	0.00
11	176	0.20	0.09	0.71	0.07	0.23	0.67	0.09
12	41	0.07	0.07	0.66	0.32	0.32	0.68	0.10
Spearman's correlation	−0.16 p = 0.62	−0.19 p = 0.55	−0.42 p = 0.18	−0.20 p = 0.78	−0.12 p = 0.72	0.49 p = 0.11	-[a]	0.39 p = 0.21

[a]Sample size is too small for correlation analysis.

Table 3. Frequencies of error types by computational thinking concepts.

Computational Concept	Decomposition	Counting	Repetition	Orientation	Premature	Conditionals	Redundancy
Sequencing	0.43	0.42		0.25	0.18		0.04
Loops	0.15	0.13	0.76	0.16	0.12		0.00
Conditions	0.20	0.08	0.68	0.20	0.28	0.68	0.09
Spearman's correlation	−1.00 p = 0.33	−1.00 p = 0.33	-[a]	−0.50 p = 1.00	0.50 p = 1.00	-[a]	0.50 p = 1.00

[a]Sample size is too small for correlation analysis.

Table 4. Frequencies of error types by SOLO-based difficulty.

SOLO Level	Decomposition	Counting	Repetition	Orientation	Premature	Conditionals	Redundancy
Unistructural	0.00	0.50	0.86	0.00	0.07		0.00
Multistructural	0.50	0.29	0.93	0.27	0.10		0.02
Relational	0.38	0.23	0.60	0.23	0.24	0.67	0.04
Extended abstract	0.19	0.21	0.66	0.30	0.33	0.68	0.11
Spearman's correlation	0.20 $p = 0.92$	-1.00 $p = 0.08$	-0.60 $p = 0.42$	0.80 $p = 0.33$	1.00 $p = 0.08$	-[a]	1.00 $p = 0.08$

[a]Sample size is too small for correlation analysis.

Associations between Types of Errors and Learner Characteristics

For each student, we calculated the rate of each type of error from their own total erroneous submissions. Then, we compared between female and male frequencies for each type of error separately, using t-test; similarly, we compared rates of type of error between participants with and without programming experience.

Rate of Decomposition type of error was found to be marginally higher for males than for females (M = 0.29, SD = 0.26, compared with M = 0.22, SD = 0.14), with t(110) = 1.78, at $p = 0.08$. Other types of errors did not show significant differences between female and male participants. Also, no differences in the rate of types of errors were found between participants with and without coding experience. Findings are summarized in Table 5.

Discussion

In this exploratory case study, we analyzed students' ($N = 123$) unsuccessful submissions in block-based programming. Based on an analysis of log files drawn from a single online learning environment (Kodetu), we identified seven types of errors. We will first point out that the overall rate of failed submissions in our study (37%) lies within the extremely wide range found in previous studies—often referred to the rate of failed compilations—which may get to as high as 90% (Blikstein, 2011; Ettles et al., 2018; Pettit, 2014; Rivers et al., 2016). Importantly, we should recall that the learning environment studied here is based on block programming, which was originally developed to decrease the complexity of coding and the number of potential errors (Jackson et al., 2005; McCall & Kölling, 2015). Indeed, syntax errors are amongst the most common errors for novice programmers (Ahadi et al., 2018; Fu et al., 2017; Hristova et al., 2003), while the type of errors we point out are non-syntactic. Reducing syntax errors is important, as they can substantially impact on the experience of learning to program, and choosing the right programming language is therefore important (Mciver, 2000b).

Two Categories of Errors: Driven by Learner, Learning Environment

Overall, we identified seven types of errors, which we can classify into two broad categories, as these errors were either mostly driven by the learner's knowledge and behavior or in the learning environment design.

Learner-Driven Errors. The first category, which is mostly learner-driven, relates to two dimensions in Brennan and Resnick's (2012) framework of studying CT, namely concepts and practices, and to their acquisition and implementation by the learners.

Concepts are the cornerstones of engaging with CT; in block-based programming, concepts are mapped to blocks. Indeed, in the learning environment we studied here, there are blocks that—when joined together—control an astronaut's path and guide her

Table 5. Frequencies of error types by gender, programming experience.

	Decomposition	Counting	Repetition	Orientation	Premature	Conditionals	Redundancy
Gender							
Female (N = 56)	0.22 (0.14)	0.17 (0.15)	0.17 (0.17)	0.14 (0.13)	0.11 (0.15)	0.11 (0.15)	0.07 (0.19)
Male (N = 56)	0.29 (0.26)	0.18 (0.21)	0.17 (0.21)	0.12 (0.14)	0.10 (0.13)	0.10 (0.15)	0.04 (0.14)
t-test (df = 110)	1.78 p = 0.08	0.42 p = 0.68	0.16 p = 0.87	1.08 p = 0.28	0.29 p = 0.77	0.58 p = 0.56	0.91 p = 0.36
Programming Experience?							
Yes (N = 52)	0.26 (0.25)	0.20 (0.20)	0.15 (0.18)	0.12 (0.11)	0.11 (0.16)	0.09 (0.14)	0.08 (0.18)
No (N = 60)	0.26 (0.19)	0.15 (0.15)	0.19 (0.20)	0.14 (0.15)	0.10 (0.13)	0.12 (0.15)	0.04 (0.16)
t-test (df = 110)	0.05 p = 0.96	1.43 p = 0.16	1.15 p = 0.25	0.83 p = 0.41	0.59 p = 0.56	1.22 p = 0.23	1.09 p = 0.28

to her destinations. Besides blocks that control movement—i.e., moving forward and turning—there are blocks that correspond to CT concepts of loops and conditionals. Indeed, we identified errors that correspond to incorrectly constructing these structures, which we classified under "repetition" and "conditional" types of errors. The prominence of these types of errors while using a rather intuitive programming interface, emphasizes students' misconceptions of basic CT concepts, such as loops and conditionals (Grover & Basu, 2017), although they are probably less prevalent in block programming compared to text-based programming (Mladenović et al., 2018). Therefore, it is recommended to explicitly teach about CT concepts using other tools and methods, including unplugged activities (Caeli & Yadav, 2020; Delal & Oner, 2020).

CT practices refer to the ways in which learners think and learn, therefore depicting the "how?" of the learning process, rather than just the "what?" (Brennan & Resnick, 2012). The "decomposition", "premature", and "redundant" types of errors fall under this broad category, as they most probably derived from implementing (or incorrectly implementing) problem-solving practices. Specifically, "decomposition" errors may be related to the practice of modularizing, that is, building something large by first building its smaller parts; "premature" errors may be a result of being incremental, that is, trying to run codes prematurely in order to follow the resulting path step by step; and "redundant" errors may be associated with either of these practices.

Notably, the characteristics of a programming language and the design of a learning environment for programming may shape novices' practices of coding (Weintrop & Wilensky, 2018). This should be emphasized to educators, who could in turn explicitly help students foster such practices (Kong et al., 2017). Furthermore, CT practices should be continuously assessed, alongside with CT concepts, in order to have a more comprehensive understanding of students' progress; only this will allow supporting them. In that sense, our approach of focusing on student errors may be taken as a metaphor to a pedagogical practice of learning from errors; that is, educators should encourage a culture where it is ok to make mistakes, as such mistakes would reveal misconceptions, knowledge gaps, and unfruitful programming practices from which students could grow (Basu et al., 2020).

Learning Envirovnment-Driven Errors. The second category of errors has mostly to do with the design of the learning environment, and holds those errors related to counting and orientation. These errors can be seen as inherent to the given learning environments, as guiding the astronaut to her destination—a task that by its very definition requires counting and turn-making—is the learners' goal. That they were evident in over 50% of the erroneous codes we analyzed, demonstrates the crucial impact of the user interface in block-based programming. To put it simply, a large portion of student errors, when being presented with the foundations of programming, was a result of misunderstanding or misinterpreting visual cues, that have little to do with programming concepts or practices per se.

these findings echo previous studies of teaching young children to program by means of using friendly and seemingly-intuitive environments—either virtual or physical—that demonstrated how visual-spatial cues may hinder learning (Simões Gomes et al., 2018; Tony Andrew Lowe, 2018). Therefore, the obstructing mechanism may be related to visual-spatial skills, which were previously found to be positively associated with learning (Mathewson, 1999; Morrison, 2004; Salmerón & García, 2012). Within the context of teaching programming to young learners, it is common to engage them with problem-solving that requires visual-spatial abilities (e.g., in Code.org, CodeMonkey, or CodeCombat), hence these abilities should be taken into consideration while designing such learning environments and while using them for teaching. Identifying these types of errors has a large contribution to the understanding of novices' errors in block programming, and extends previous attempts to classify such errors (e.g., Emerson et al., 2020).

this is precisely what RULER does

Associations Between Error Type and Personal and Task Characteristics

We found no associations between error type and learner characteristics, specifically gender and prior coding experience. Previous studies indicated on differences in the ways by which children with prior coding experience engage with programming or computational thining tasks, and that such experience may impact their preferences towards certain types of programming language (Bakali et al., 2018; Menounou et al., 2019; Weintrop & Wilensky, 2015). To these, our findings add the important notion that types of errors do not necessarily correlate with prior coding experience, which highlights the importance of the very design of the learning environment.

As for gender, previous studies had demonstrated how girls and boys manifest differently the way they construct block-based programs (Funke & Geldreich, 2017; Hsu, 2014). When measuring achievementsin programming tasks, boys and girls do not necessarily differ from each other (Abdullah et al., 2021; Vasilopoulos & van Schaik, 2018; Zha & Billingsley, 2021). Indeed, a large-scale, log-based study of the same system used here (i.e., Kodetu), covering 3355 primary- and middle-school students, has shown some complex associations of behavior within the system and gender, prior coding experience (Eguíluz et al., 2017). Therefore, our findings of no associations between gender and error types may not be surprising.

Also, we pointed out the lack of associations between error types and game level or concept taught, contrary to previous large-scale analyses that suggest a performance decrease in both block-based and text-based learning environments for programming as the game progresses (Eguíluz et al., 2017; Israel-Fishelson & Hershkovitz, 2020); That is, our study adds to the existing literature the notion that as learners progress in a learning process, they may find it more difficult and hence submit more erroneous solutions, but the very mechanisms that drive these failed submissions do not necessarily change profoundly. This makes us reiterate the notion of learning from mistakes, which we highlighted above, as a means to change learners' behavior

meaningfully. Therefore, while using self-paced learning environments, learners may benefit from some human- or machine-led guidance.

The only attribute that was found to be associated with the type of error was the task complexity, as defined by SOLO taxonomy (Biggs & Collis, 1982). While counting errors decrease as tasks become more complex, premature and redundancy errors increase. Importantly, the tasks classified at the higher levels of SOLO taxonomy, compared with those at the lower levels, are not necessarily characterized by shorter paths of the puzzles they present, hence the decrease in counting-related errors is not explained by the maze length, and may be easily explained by their familiarity with the learning environment in which all tasks are designed similarly.

The increase in premature and redundant types of errors—which, as we explained above, are related to CT practices—may indicate that students modify their problem-solving strategies as tasks become more complex, probably working incrementally to a larger extent and trying to break down the problem to smaller, simpler ones. As previously suggested, students implement various strategies of problem solving in programming, either textual of block-based, and it is the changes in these strategies, and not merely the strategies themselves, that are more indicative on learning (Blikstein et al., 2014; Kesselbacher & Bollin, 2019). In a sense, our findings echo Breland et al.'s (2013) ideitnfication of learning pathways of novice programmers; their findings demonstrated a shift from exploring to tinkering to refinement, which may be associated with the increase we suggest in the use of a systemic way of problem-solving.

Therefore, identifying these changes sheds new light on the mechanisms that help students transfer from visual to procedural programming (Yetunde, 2018), and emphasizes the importance role of block-based programming in strengthening the very foundations of CT (Repenning, 2017; Sáez-López et al., 2016). Achieving this desired goal was probably a result of the learning environment made of a series of seemingly-similar tasks, which made the learners first familiar with the environment and its user interface, and then led them towards the learning goal in small steps. This is indeed a desired design principle of digital learning games, and at large of any learning experience (Diniz et al., 2015).

Nevertheless, for increasing the impact of this supportive design, it is advised to make explicit the implementation and development of CT practices. This may be achieved by guiding students through reflecting upon their problem-solving strategies, as was previously demonstrated in various domains (Cengiz & Karataş, 2015; C. H. Lin & Liu, 2012; Mason & Singh, 2016). Within the context of self-paced online learning environments, such a reflective process could be guided by a virtual agent (Egbert et al., 2021; Moechammad Sarosa et al., 2021).

Limitations

This study is not without limitations. We analyzed data from a single learning platform (Kodetu), and therefore our findings may be influenced by unique characteristics pertaining to this platform. Our analysis also focused on a dozen tasks

which refer to a limited set of CT concepts, namely, sequence, loops, and conditionals. It is of great importance to study all CT concepts (Barr & Stephenson, 2011; Brennan & Resnick, 2012). Moreover, the analysis was focused on students from a single country (Spain)—which has specific educational, cultural, and technological characteristics—and on a specific age group. Therefore, it is advised to replicate this study in other learning environments, and in other geographical and cultural settings. Another limitation lies in the mechanism of progressing in the learning environment studied, that is, the need to successfully complete a task before moving on to the next one. This design principle may reduce errors that would otherwise arise because students have had training and have gained experience in previous tasks. It is possible, therefore, that more open, exploratory learning environments would yield other types of errors, or at least other frequencies of the types we found, and therefore encourage replicating this study is such environments. Still, we believe that we add an important unique contribution to the literature, and that our study has valuable implications.

Conclusions and Implications

In this study, we characterized students' errors in block-based programming. We found two broad categories of errors, driven by either learners' knowledge and behavior or by the learning environment design. Testing for associations between error types and personal and task characteristics, we only found such a link between error types and SOLO-defined task complexity, suggesting that it may indicate the development of problem-solving strategies as a result of the design of the learning environment.

This study has a few important implications, of which we will highlight three. First, identifying types of errors may lead to the improvement of CT acquisition processes while using block-based programming. With no syntax involved, educators could use student errors as learning opportunities to strengthen students' understanding of the foundations of CT, that is, to promote a culture of learning from errors. Second, identification of error types may help support learners with formative feedback, and if errors are detected and recognized automatically in real-time, this could serve to offer students insightful, helpful support. Third, design characteristics may have a crucial impact on learners' progress—hence, being a key component in assessment processes—and should be taken into consideration by developers and designers of learning environment, as well as by educators and policy makers.

We recommend to further explore patterns of error in block programming and the relationship between them and content- and task-characteristics.

Acknowledgments

The authors would like to thank to the members of DeustoLearningLab at the University of Deusto for their ongoing collaboration and their continuous support. Special thanks to Prof. Mariluz Guenaga (Director), Dr Pablo Garaizar, and Andoni Eguíluz.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

ORCID iD

Arnon Hershkovitz https://orcid.org/0000-0003-1568-2238

References

Abdullah, O., Kamaludin, A., & Rahman, N. S. A. (2021). Gender differences in computational thinking skills among Malaysian's primary school students using visual programming. *Proceedings of the International Conference on Software Engineering and Computer Systems and 4th International Conference on Computational Science and Information Management*, 655–660. https://doi.org/10.1109/ICSECS52883.2021.00125

Agbo, F. J., Oyelere, S. S., Suhonen, J., & Adewumi, S. (2019). A systematic review of computational thinking approach for programming education in higher education institutions. *ACM International Conference Proceeding Series*, 1–10. https://doi.org/10.1145/3364510.3364521

Ahadi, A., Lal, S., Lister, R., & Hellas, A. (2018). Learning programming, syntax errors and institution-specific factors. *ACM International Conference Proceeding Series*, 90–96. https://doi.org/10.1145/3160489.3160490

Anderson, L. W., & Krathwohl, D. R. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Longman.

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to "real" programming. *ACM Transactions on Computinig Education*, *14*(4), 1–15. https://doi.org/10.1145/2677087

Bakali, I., Fourtounis, N., Theodoulidis, A., Chatzis, D., Soulountsi, M., Papanastasiou, P., Kavallieratou, E., Manos, N., & Mavropoulos, N. (2018, July 9). Control a robot via internet using a block programming platform for educational purposes. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3200947.3201063

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

Basu, S., Rutstein, D., Xu, Y., & Shear, L. (2020). A principled approach to designing a computational thinking practices assessment for early grades. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 7. https://doi.org/10.1145/3328778

Becker, B. A., Glanville, G., Iwashima, R., McDonnell, C., Goslin, K., & Mooney, C. (2016). Effective compiler error message enhancement for novice programming students. *Computer Science Education*, *26*(2–3), 148–175. https://doi.org/10.1080/08993408.2016.1225464

Berland, M., Martin, T., Benton, T., Petrick Smith, C., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences*, *22*(4), 564–599. https://doi.org/10.1080/10508406.2013.836655

Biggs, J., & Collis, K. F. (1982). Evaluating the quality of learning: The SOLO taxonomy. In *A handbook for teaching and learning in higher education: Enhancing academic practice*. Academic Press.

Blikstein, P. (2011). Using learning analytics to assess students ' behavior in open-ended programming tasks. *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, 110–116. https://doi.org/10.1145/2090116.2090132

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., & Koller, D. (2014). Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, *23*(4), 561–599. https://doi.org/10.1080/10508406.2014.954750

Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H., & Krathwohl, D. R. (1956). *Taxonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain*. David McKay Company.

Brennan, K., & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking* (pp. 1–25). 2012 Annual Meeting of the American Educational Research Association. https://doi.org/10.1.1.296.6602

Caeli, E. N., & Yadav, A. (2020). Unplugged approaches to computational thinking: A historical perspective. *TechTrends*, *64*(1), 29–36. https://doi.org/10.1007/s11528-019-00410-5

Cengiz, C., & Karataş, F. Ö. (2015). Examining the effects of reflective journals on pre-service science teachers' general chemistry laboratory achievement. *Australian Journal of Teacher Education*, *40*(10), 125–146. https://doi.org/10.14221/ajte.2015v40n10.8

Chan Mow, I. T. (2012). Analyses of student programming errors in Java programming courses. *Journal of Emerging Trends in Computing and Information Sciences*, *3*(5), 739–749.

Delal, H., & Oner, D. (2020). Developing middle school students' computational thinking skills using unplugged computing activities. *Informatics in Education*, *19*(1), 1–13. https://doi.org/10.15388/INFEDU.2020.01

Diniz, A., Santos, D., & Fraternali, P. (2015). A comparison of methodological frameworks for digital learning game design. *International Conference on Games and Learning Alliance, 9599*, 111–120.

Egbert, J., Shahrokni, S. A., Abobaker, R., & Borysenko, N. (2021). It's a chance to make mistakes": Processes and outcomes of coding in 2nd grade classrooms. *Computers & Education*, *168*, 104173. https://doi.org/10.1016/J.COMPEDU.2021.104173

Eguíluz, A., Guenaga, M., Garaizar, P., & Olivares-Rodríguez, C. (2017). Exploring the progression of early programmers in a set of computational thinking challenges via clickstream analysis. *IEEE Transactions on Emerging Topics in Computing*, *8*(1), 256–261. https://doi.org/10.1109/tetc.2017.2768550

Emerson, A., Smith, A., Rodriguez, F. J., Wiebe, E. N., Mott, B. W., Boyer, K. E., & Lester, J. C. (2020). Cluster-based analysis of novice coding misconceptions in block-based programming. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 825–831. https://doi.org/10.1145/3328778.3366924

Ettles, A., Luxton-Reilly, A., & Denny, P. (2018). Common logic errors made by novice programmers. *ACM International Conference Proceeding Series*, 83–89. https://doi.org/10.1145/3160489.3160493

Fagerlund, J., Häkkinen, P., Vesisenaho, M., & Viiri, J. (2021). Computational thinking in programming with Scratch in primary schools: A systematic review. *Computer Applications in Engineering Education*, *29*(1), 12–28. https://doi.org/10.1002/CAE.22255

Funke, A., & Geldreich, K. (2017). Gender dierences in scratch programs of primary school children. *ACM International Conference Proceeding Series*, 57–64. https://doi.org/10.1145/3137065.3137067

Fu, X., Shimada, A., Ogata, H., Taniguchi, Y., & Suehiro, D. (2017). Real-time learning analytics for C programming language courses. *ACM International Conference Proceeding Series*, 280–288. https://doi.org/10.1145/3027385.3027407

Ginat, D., & Menashe, E. (2015). SOLO taxonomy for assessing novices' algorithmic design. *SIGCSE 2015 - Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 452–457. https://doi.org/10.1145/2676723.2677311

Ginat, D., & Shmallo, R. (2013). Constructive use of errors in teaching CS1. *SIGCSE 2013 - Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, 353–358. https://doi.org/10.1145/2445196.2445300

Gladwin, L. A. (1987). Intention-based diagnosis of novice programming errors. *IEEE Expert-Intelligent Systems and Their Applications*, *2*(3), 162–168. https://doi.org/10.1109/MEX.1987.4307101

Grandell, L., Peltomäki, M., & Salakoski, T. (2005). High school programming - A beyond-syntax analysis of novice programmers' difficulties. *Koli Calling 2005 Conference on Computer Science Education*, 17–24.

Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic. *Proceedings of the Conference on Integrating Technology Into Computer Science Education, ITiCSE*, 267–272. https://doi.org/10.1145/3017680.3017723

Guenaga, M., Eguíluz, A., Garaizar, P., & Gibaja, J. (2021). How do students develop computational thinking? Assessing early programmers in a maze-based online game. *Computers & Education*, *31*(2), 259–289. https://doi.org/10.1080/08993408.2021.1903248

Haleva, L., Hershkovitz, A., & Tabach, M. (2021). Students' activity in an online learning environment for mathematics: The role of thinking levels. *Journal of Educational Computing Research*, *59*(4), 686–712. https://doi.org/10.1177/0735633120972057

Harrison, W. S, & Hanebutte, N. (2018). Embracing coding mistakes to teach computational thinking. *Journal of Computing Sciences in Colleges*, *33*(6), 52–62.

Hershkovitz, A., Sitman, R., Israel-Fishelson, R., Eguíluz, A., Garaizar, P., & Guenaga, M. (2019). Creativity in the acquisition of computational thinking. *Interactive Learning Environments*, *27*(5–6), 628–644. https://doi.org/10.1080/10494820.2019.1610451

Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). Identifying and correcting Java programming errors for introductory computer science students. *SIGCSE Bulletin*

(Association for Computing Machinery, Special Interest Group on Computer Science Education, *35*(1), 153–156. https://doi.org/10.1145/792548.611956

Hsu, H.-M. J. (2014). Gender differences in scratch game design. *3rd International Conference on Information, Business and Education Technology*, 100–103.

Israel-Fishelson, R., & Hershkovitz, A. (2020). Shooting for the stars: Micro-persistence of students in game-based learning environments. In D. Glick, A. Cohen, & C. Chang (Eds.), *Early warning systems and targeted interventions for student success in online courses* (pp. 239–258). IGI Global. https://doi.org/10.4018/978-1-7998-5074-8.ch012

Israel-Fishelson, R., Hershkovitz, A., Eguíluz, A., Garaizar, P., & Guenaga, M. (2021). The associations between computational thinking and creativity: The role of personal characteristics. *Journal of Educational Computing Research*, *58*(8), 1415–1447. https://doi.org/10.1177/0735633120940954

Izu, C., Weerasinghe, A., & Pope, C. (2016). A study of code design skills in novice programmers using the SOLO taxonomy. *ICER 2016 - Proceedings of the 2016 ACM Conference on International Computing Education Research*, 251–259. https://doi.org/10.1145/2960310.2960324

Jackson, J., Cobb, M., & Carver, C. (2005). Identifying top java errors for novice programmers. *Proceedings - Frontiers in Education Conference, FIE, 2005*, 24–27. https://doi.org/10.1109/fie.2005.1611967

Kaspersen, M. H., Graungaard, D., Bouvin, N. O., Petersen, M. G., & Eriksson, E. (2021). Towards a model of progression in computational empowerment in education. *International Journal of Child-Computer Interaction*, *29*, 100302. https://doi.org/10.1016/J.IJCCI.2021.100302

Kelleher, C., Cosgrove, D., & Culyba, D. (2002). *Alice2: Programming without syntax errors* (pp. 3–4). User Interface Software and Technology - UIST 2002.

Kesselbacher, M., & Bollin, A. (2019). Discriminating programming strategies in scratch - Making the difference between novice and experienced programmers. *Proceedings of the 14th Workshop in Primary and Secondary Computing Education*, 1–10. https://doi.org/10.1145/3361721.3361727

Koehler, A. T. (2020). *A methodology for teaching from student errors in computer science education*. Unpublished Ph.D. Dissertation. University of California Riverside.

Ko, A. J., & Myers, B. A. (2003). Development and evaluation of a model of programming errors. *Proceedings - 2003 IEEE Symposium on Human Centric Computing Languages and Environments HCC 2003*, 7–14. https://doi.org/10.1109/HCC.2003.1260196

Kong, S.-C., Abelson, H., Sheldon, J., Lao, A, Tissenbaum, M., Lai, M., Lang, K., & Lao, N. (2017). Curriculum activities to foster primary school students' computational practices in block-based programming environments. In S. C. Kong, J. Sheldon, & K. Y. Li (Eds.), *Conference Proceedings of International Conference on Computational Thinking Education* (pp. 84–89). Hong Kong: The Education University of Hong Kong.

Lai, G. C. H., Kwok, R. C. W., & Kong, J. S. L. (2020). Teaching computational thinking and python programming for business students: A preliminary study of the alignment of teaching and learning strategies with bloom's taxonomy of learning outcomes. *Proceedings of International Conference on Computational Thinking Education*, 116–120.

Lin, C. H., & Liu, E. Z. F. (2012). The effect of reflective strategies on students' problem solving in robotics learning. *Proceedings 2012 4th IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning, DIGITEL 2012*, 254–257. https://doi.org/10.1109/DIGITEL.2012.67

Lin, L., Parmar, D., Babu, S. v., Leonard, A. E., Daily, S. B., & Jörg, S. (2017, September 16). How character customization affects learning in computational thinking. *Proceedings - SAP 2017, ACM Symposium on Applied Perception*, 1–8. https://doi.org/10.1145/3119881.3119884

Lister, R., Simon, B., Thompson, E., Whalley, J. L., & Prasad, C. (2006). Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education 2006*, *38*(3), 118–122. https://doi.org/10.1145/1140124.1140157

Lowe, T. A. (2018). *Misconceptions and the notional machine in very young programming learners*. American Society for Engineering Education Annual Conference & Exposition.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education*, *10*(4), 1–15. https://doi.org/10.1145/1868358.1868363

Marwan, S., Jay Williams, J., & Price, T. (2019). An evaluation of the impact of automated programming hints on performance and learning. *Proceedings of the 2019 ACM Conference on International Computing Education Research*, 61–70. https://doi.org/10.1145/3291279

Masapanta-Carrión, S., & Velázquez-Iturbide, J. Á. (2018). A systematic review of the use of Bloom's taxonomy in computer science education. *SIGCSE 2018 - Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 441–446. https://doi.org/10.1145/3159450.3159491

Mason, A. J., & Singh, C. (2016). Impact of guided reflection with peers on the development of effective problem solving strategies and physics learning. *The Physics Teacher*, *54*(5), 295–299. https://doi.org/10.1119/1.4947159

Mathewson, J. H. (1999). Visual-spatial thinking: An aspect of science overlooked by educators. *Science Education*, *83*(1), 33–54. https://doi.org/10.1002/(sici)1098-237x(199901)83:1<33::aid-sce2>3.0.co;2-z

McCall, D., & Kölling, M. (2015). Meaningful categorisation of novice programmer errors. *Proceedings - Frontiers in Education Conference, FIE, 2015-Febru(February)*. https://doi.org/10.1109/FIE.2014.7044420.

McIver, L. (2000a). The effect of programming language on error rates of novice programmers. *12th Workshop of the Psychology of Programming Interest Group*, 181–192.

Mciver, L. (2000b). The effect of programming language on error rates of novice programmers. In A.F. Blackwell & E. Bilotta (Eds.), *Proceedings of the 12th workshop of the psychology of programming interest group* (pp. 181–192). www.ppig.org.

Menounou, G., Pantelopoulou, S., Karaliopoulou, M., & Kanidis, E. (2019). Students' perceptions on using a dual modality programming environment. *European Journal of Engineering Research and Science*, 19–27. https://doi.org/10.24018/ejers.2019.0.cie.1292

Metcalfe, J. (2017). Learning from errors. *Annual Review of Psychology*, *68*(1), 465–489. https://doi.org/10.1146/annurev-psych-010416-044022

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

Mohd Darby, N., & Mat Rashid, A. (2017). Critical thinking disposition: The effects of infusion approach in engineering drawing. *Journal of Education and Learning*, *6*(3), 305. https://doi.org/10.5539/jel.v6n3p305

Montiel, H., & Gomez-Zermeño, M. G. (2021). Educational challenges for computational thinking in k–12 education: A systematic literature review of "scratch" as an innovative programming tool. *Computers*, *10*(6), 69. https://doi.org/10.3390/computers10060069

Morrison, J. B. (2004). The effect of spatial ability on learning from text and graphics. *Proceedings of the Annual Meeting of the Cognitive Science Society*, *26*(26), 1608.

Nalaka, E. M., & Edirisinghe, S. (2008). Teaching students to identify common programming errors using a game. *SIGITE'08: Proceedings of the 9th ACM SIG-Information Technology Education Conference*, *14*, 95–98. https://doi.org/10.1145/1414558.1414586

Osadi, K. A., Fernando, M. G. N. A. S., & Welgama, W. V. (2017). Ensemble classifier based Approach for classification of examination questions into Bloom's Taxonomy cognitive levels. *International Journal of Computer Applications*, *162*(4), 1–6. https://doi.org/10.5120/ijca2017913328

Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., & Lahmine, S. (2015). Learning basic programming concepts by creating games with Scratch programming environment. *Procedia - Social and Behavioral Sciences*, *191*, 1479–1482. https://doi.org/10.1016/j.sbspro.2015.04.224

Parmar, D., Lin, L., Dsouza, N., Joerg, S., Leonard, A. E., Daily, S. B., & Babu, S. (2022). How immersion and self-avatars in VR affect learning programming and computational thinking in middle school education. *IEEE Transactions on Visualization and Computer Graphics*, 1. https://doi.org/10.1109/TVCG.2022.3169426

Petersen, A., Craig, M., & Zingaro, D. (2011). Reviewing CS1 exam question content. *SIGCSE'11 - Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 631–636. https://doi.org/10.1145/1953163.1953340

Pettit, M. A. M. (2014). *A case study of the implementation of iPads with high school students at two charter high schools in Southern California*. [unpublished Ph.D. Dissertation]. Vol. *3630494*. http://pepperdine.contentdm.oclc.org/cdm/ref/collection/p15093coll2/id/469

Qian, Y., & Lehman, J. D. (2019). Using targeted feedback to address common student misconceptions in introductory programming: A data-driven approach. *9*(4), 215824401988513https://doi.org/10.1177/2158244019885136

Rach, S., Ufer, S., & Heinze, A. (2013). Learning from errors: Effects of reachers training on students' attitudes towards and their individual us of errors. *Proceedings of the National Academy of Sciences*, *8*(1), 21–30. https://doi.org/10.30827/pna.v8i1.6122

Rangie, J., Obispo, C., Enrique, F., Castro, V. G., Mercedes, M., & Rodrigo, T. (2018). Incidence of einstellung effect among programming students and its relationship with achievement.

Repenning, A. (2017). Moving beyond syntax: Lessons from 20 years of blocks programing in agentsheets. *Journal of Visual Languages and Sentient Systems*, *3*(1), 68–91. https://doi.org/10.18293/vlss2017-010

Rivers, K., Harpstead, E., & Koedinger, K. (2016). Learning curve analysis for programming: Which concepts do students struggle with? *ICER 2016 - Proceedings of the 2016 ACM Conference on International Computing Education Research*, 143–151. https://doi.org/10.1145/2960310.2960333

Rodrigo, M. M. T., Andallaza, T. C. S., Castro, F. E. V. G., Armenta, M. L. V., Dy, T. T., & Jadud, M. C. (2013). An analysis of java programming behaviors, affect, perceptions, and syntax errors among low-achieving, average, and high-achieving novice programmers. *Journal of Educational Computing Research*, *49*(3), 293–325. https://doi.org/10.2190/EC.49.3.b

Rubin, J., & Rajakaruna, M. (2015). Teaching and assessing higher order thinking in the mathematics classroom with clickers. *International Electronic Journal of Mathematics Education*, *10*(1), 37–51. https://doi.org/10.12973/mathedu.2015.103a

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, *97*, 129–141. https://doi.org/10.1016/J.COMPEDU.2016.03.003

Salmerón, L., & García, V. (2012). Children's reading of printed text and hypertext with navigation overviews: The role of comprehension, sustained attention, and visuo-spatial abilities. *Journal of Educational Computing Research*, *47*(1), 33–50. https://doi.org/10.2190/ec.47.1.b

Sarosa, M., Kusumawadhani, M., Suyono, A., & Mulyani Azis, Y. (2021). The effectiveness of chatbot as an online learning method on active and reflective learning styles. *Multicultural Education*, *7*(9), 92–100. https://doi.org/10.5281/zenodo.5484411

Seiter, L. (2015). Using SOLO to classify the programming responses of primary grade students. *SIGCSE 2015 - Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 540–545. https://doi.org/10.1145/2676723.2677244

Selby, C. C. (2015). Relationships: Computational thinking, pedagogy of programming, and Bloom's taxonomy. *ACM International Conference Proceeding Series, 09-11-November-2015*, 80–87. https://doi.org/10.1145/2818314.2818315

Seo, H., Sadowski, C., Elbaum, S., Aftandilian, E., & Bowdidge, R. (2014). Programmers' build errors: A case study (at Google). *International Conference on Software Engineering*, *1*, 724–734. https://doi.org/10.1145/2568225.2568255

Seraj, M., Katterfeldt, E. S., Bub, K., Autexier, S., & Drechsler, R. (2019). Scratch and google blockly: How girls' programming skills and attitudes are influenced. *PervasiveHealth:*

Pervasive Computing Technologies for Healthcare, 1–10. https://doi.org/10.1145/3364510.3364515

Shooman, M. L., & Bolsky, M. I. (1975). Types, distribution, and test and correction times for programming errors. *ACM SIGPLAN Notices*, *10*(6), 347–357. https://doi.org/10.1145/800027.808457

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*(1), 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Simões Gomes, T. C., Pontual Falcão, T., & Cabral de Azevedo Restelli Tedesco, P. (2018). Exploring an approach based on digital games for teaching programming concepts to young children. *International Journal of Child-Computer Interaction*, *16*, 77–84. https://doi.org/10.1016/j.ijcci.2017.12.005

Smith, R., & Rixner, S. (2019). The error landscape: Characterizing the mistakes of novice programmers. *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 538–544. https://doi.org/10.1145/3287324.3287394

Socratous, C., & Ioannou, A. (2020). Common errors, successful debugging, and engagement during block-based programming using educational robotics in elementary education. *14th International Conference of the Learning Sciences*, 991–998.

Staub, J. (2021). *Programming in k-6: Understanding errors and supporting autonomous learning*. Unpublished Ph.D. Dissertation. ETH Zürich.

Tatar, C., & Eseryel, D. (2019). A literature review: Fostering computational thinking through game-based learning in K-12. *The 42nd Annual Convention of The Association for the Educational Communications and Technology*, 288–297.

Theodoropoulos, A., & Lepouras, G. (2020). Digital game-based learning and computational thinking in P-12 education: A systematic literature review on playing games for earning programming. In M. Kalogiannakis & S. Papadakis (Eds.), *Handbook of research on tools for teaching computational thinking in p-12 education* (pp. 159–183). IGI Global. https://doi.org/10.4018/978-1-7998-4576-8.CH007

Thompson, S. M. (2006). *Exploratory study of novice programming experiences and error*. Unpublished Master's thesis. University of Victoria.

Tulis, M., Steuer, G., & Dresel, M. (2016). Learning from errors: A model of individual processes. *Frontline Learning Research*, *4*(4), 12–26. https://doi.org/10.14786/flr.v4i2.168

Ullah, Z., Lajis, A., Jamjoom, M., Altalhi, A., & Saleem, F. (2020). Bloom's taxonomy: A beneficial tool for learning and assessing students' competency levels in computer programming using empirical analysis. *Computer Applications in Engineering Education*, *28*(6), 1628–1640. https://doi.org/10.1002/CAE.22339

Vasilopoulos, I. v., & van Schaik, P. (2018). Koios: Design, development, and evaluation of an educational visual tool for Greek novice programmers. *Journal of Educational Computing Research*, *57*(5), 1227–1259. https://doi.org/10.1177/0735633118781776

Weintrop, D., & Wilensky, U. (2015). To block or not to block, that is the question: Students' perceptions of blocks-based programming. *Proceedings of IDC 2015: The 14th International Conference on Interaction Design and Children*, 199–208. https://doi.org/10.1145/2771839.2771860

Weintrop, D., & Wilensky, U. (2018). How block-based, text-based, and hybrid block/text modalities shape novice programming practices. *International Journal of Child-Computer Interaction*, *17*, 83–92. https://doi.org/10.1016/j.ijcci.2018.04.005

Whalley, J., Clear, T., Robbins, P., & Thompson, E. (2011). Salient elements in novice solutions to code writing problems. *Conferences in Research and Practice in Information Technology Series*, *212*(2839), 37–45.

Yarygina, O. (2020). Learning analytics of CS0 students programming errors: The case of data science minor. *ACM International Conference Proceeding Series*, 149–152. https://doi.org/10.1145/3377290.3377319

Yetunde, T. F. (2018). *Taking first year college students' programming skills from the visual to the procedural*. Unpublished Ph.D. dissertation.

Zha, S., & Billingsley, J. (2021). Understanding students' pre-existing computational thinking skills and its relationship with their block programming performance. *The 42nd Annual Convention of The Association for the Educational Communications and Technology*, *59*(8), 265–268. https://doi.org/10.1177/07356331211004048

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers and Education*, *141*(2018), 103607. https://doi.org/10.1016/j.compedu.2019.103607

Authors Biographies

Anat Ben-Yaacov is a computer science education expert. She earned an MA in Computer Science Education (2021) from Tel Aviv University's School of Education (Israel), an MBA (2007) from Heriot-Watt University (Edinburgh, Scotland) and a BA in Computer Science (1999) from The Academic College of Tel Aviv-Yaffo (Israel). She works as a computer science educator at The Academic College of Tel Aviv-Yaffo and at an Israeli high school. She serves as a pedagogy developer for ed-tech companies, and leads a teachers community. Prior to her educational career, she worked as a product manager and solution engineer in the high-Tech industry.

Arnon Hershkovitz, PhD, is a senior lecturer at Tel Aviv University's School of Education (Israel). He earned his Ph.D. in Science Education from Tel Aviv University, Israel, in 2011, after completing his MA in Applied Mathemathematics (2003), and BA in Mathematics and Computer Science (1996) at the Technion Israel Institute of Technology. He was a post-doctoral research associate in the Department of Human Development at Teachers College Columbia University (NYC), and had spent a visiting professor fellowship at Northwestern University's School of Education and Social Policy (Evanston, IL). He had served as a co-editor of Technology, Instruction, Cognition and Learning (TICL) and of the Journal of Learning Analytics (JLA), and has served as chair and reviewer for many peer-reviewed conferences. He is an alumni of the European Commission's Marie Curie Career Integration Grant. His main research interests are the application of learning analytics methods to studying the skills that learners and instructors require in today's digital age.