# Is My Game OK Dr. Scratch?

## Exploring Programming and Computational Thinking Development via Metrics in Student-Designed Serious Games for STEM

Giovanni Maria Troiano[1], Sam Snodgrass[1], Erinç Argımak[1,5], Gregorio Robles[2], Gillian Smith[3], Michael Cassidy[4], Eli Tucker-Raymond[4], Gillian Puttick[4], and Casper Harteveld[1]

[1]Northeastern University, Boston, MA | {g.troiano, s.snodgrass, c.harteveld}@northeastern.edu

[2]Universidad Rey Juan Carlos, Madrid, Spain | grex@gsyc.urjc.es

[3]Worcester Polytechnic Institute, Worcester, MA | gmsmith@wpi.edu

[4]TERC, Cambridge, MA | {michael_cassidy, eli_tucker-raymond, gilly_puttick}@terc.edu

[5]argimak.e@husky.neu.edu

## ABSTRACT

Computational thinking (CT) is key to digital literacy and helps develop problem-solving skills, which are fundamental in modern school. As game design shows potential for teaching CT, metrics like Dr. Scratch emerge that help scholars systematically assess the CT of student-designed games, particularly with Scratch. Compared to other CT metrics, Dr. Scratch scores the CT of Scratch projects automatically and can be used to describe CT development. However, previous research using Dr. Scratch summatively assessed CT, but did not look at CT development. We use Dr. Scratch to assess the CT development of Scratch games designed by 8$^{th}$-grade students in STEM curricula. We show how CT proficiency in student-designed games develops differently in each CT dimension, where *parallelism*, *synchronization*, and *logic* develop proficiently, while developing *abstraction* seems hard. We discuss insights into game-based CT development for STEM, and suggest improvements for metric-based CT assessment.

## CCS CONCEPTS

• **Social and professional topics** → **Computing literacy**; **Student assessment**; **K-12 education**; • **Human-centered computing** → *User studies*; *Empirical studies in interaction design*;

## KEYWORDS

Computational thinking, game design, STEM, constructionist learning, assessment, metrics, Scratch, Dr. Scratch

## 1 INTRODUCTION

Computational thinking (CT) [79, 83, 84] is emerging in school curricula as a critical practice to be taught and leveraged in the 21st Century [17]. It involves using key learning outcomes in modern education, such as critical thinking [6, 56] and problem-solving [3, 26, 70, 71, 81], to support practices that lead to digital literacy [16, 23, 37], including programming [32–34, 54] and algorithmic thinking [20, 21, 38, 73]. As CT is essentially a problem-solving process (viewed from the perspective of how a computer would express problems and find solutions), the use of computing is now present in most school curricula to support problem-solving across various disciplines [75] (e.g., it has been increasingly implemented in STEM curricula [3, 25, 26, 59]).

Meanwhile, the increasing adoption of CT in modern schools is calling for tools that allow for effective teaching and assessment of CT [24, 49]. For *teaching CT*, game-based learning [70, 71], particularly game design [2, 5, 12, 13], shows promise. Game design has the benefit of introducing students to CT through creative practices [2, 13, 18, 22, 28, 39, 50, 59], and allows them to apply CT concretely while designing and creating digital artifacts [29, 30]. For *assessing CT*, various instruments are being developed, including surveys [10] and tests [8], but finding and validating CT measures that assess CT consistently across different disciplines remains challenging. A particularly promising area of development within CT assessment is the use of *metrics* [51, 84], that is, quantifiable, operationalized measures of CT based on observed coding practices, which relate to and deploy CT skills. Furthermore, CT metrics may be computationally automated for assessing CT without the need of human supervision. Such automated CT metrics may alleviate teachers' struggle with manual assessment of students' CT skills, as well as providing real-time feedback on how students develop CT competency over time. For that purpose, CT metrics like Dr. Scratch [49] have emerged recently, which automatically score CT competency on seven CT dimensions (e.g., flow control) that are based on observed coding practices in Scratch [63].

We use the automated CT metrics provided by Dr. Scratch to assess CT practices deployed by 8$^{th}$-grade students (i.e., age 13 to 14) via game design. We assess CT proficiency in 317 and explore CT development in 217 student-designed serious games for STEM. The games were designed by pairs of 8$^{th}$-grade students as part of a STEM curriculum taught by science teachers at middle schools in the greater Boston, MA area. In this newly developed curriculum, which aims to integrate the learning of CT, climate science, and systems thinking in parallel, students were tasked to design serious games on climate change using Scratch [63]. As such, compared to previous work that assessed CT in game design [1, 48, 50–52],



*IDC '19, June 12–15, 2019, Boise, ID, USA*



ACM ISBN 978-1-4503-6690-8/19/06.
https://doi.org/10.1145/3311927.3323152

our work provides unique insights on how CT develops via game design within integrated learning practices that involve multiple disciplines (i.e., STEM, CT, and game design).

However, as we focus on the promising use of metrics for assessing CT development, we do not analyze the qualitative aspects of the games (e.g., game genres, aesthetics) and curriculum (e.g., teacher knowledge and style, students' prior knowledge), or consider how and whether such games were effective as instrument to learn about climate science and systems thinking. Also, differently from earlier work [1, 48, 50–52], which focused their assessment only on the final CT scores given by Dr. Scratch, we analyze how CT scores develop over time.

In this paper, we focus on and explore the extent to which current CT metrics can be used to assess CT development, and provide insights on how students develop their CT skills across the entire duration of Scratch projects. In the context of Dr. Scratch, this means observing how seven CT dimensions develop over time, thereby indicating what dimensions develop early and more proficiently, and which seem harder to develop in student-designed serious games for STEM. We contribute the following to children uptake of CT via game design in constructionist learning and CT metrics: (1) insights into CT development and its assessment in the context of student-designed serious games with Scratch for STEM, (2) discuss lessons learned from using the Dr. Scratch metrics for exploring and assessing CT development, and (3) identify if game design and CT development align with specific learning goals of current STEM curricula, from the sole perspective of CT metrics.

## 2 RELATED WORK

We use Dr. Scratch [49] to explore and assess CT development in serious games designed by pairs of $8^{th}$-grade students for STEM. As such, we position our work in relation to (1) game-based learning [57], particularly game design for learning CT [80], and (2) metrics for assessing CT (e.g., [10, 11]). Next, we review (1) existing definitions of CT to provide context, (2) previous research on game design for CT learning, (3) Scratch and its use for developing CT, and (4) current metric-based approaches to CT assessment.

### 2.1 Computational Thinking

*Computational thinking* or CT [79] is generally defined as the process through which problems are formulated in a way that can be "understood" (and solved) by computers. The term CT is often associated with *algorithmic thinking* [20] and *programming thinking* [15], and it receives increasing attention from researchers [80, 83, 84] while emerging in modern education [42, 43]. Although research has not yet agreed on a universal definition, CT is widely recognized as a competence associated with problem-solving. For instance, CT is described as involving problem-solving using computer science knowledge, concepts, techniques, and perspectives [43, 78]. Weintrop et al. [80] explain that CT embodies preparing computational solutions for problems, developing modular solutions, creating abstractions and troubleshooting and debugging. Wing [83] proposes that CT is the thought processes involved in formulating problems and their solutions.

Most recently, in an attempt to "demystify" CT, the concept was categorized into six facets (i.e., decomposition, abstraction, algorithm design, debugging, iteration and generalization), based on the definition of CT as "a conceptual foundation required to solve problems effectively and efficiently, with solutions that are reusable in different contexts" [76]. Moreover, based on recent reviews on CT, problem-solving emerges as one the three terms that are most used to define CT practices [33]. As such, it is reasonable to see CT being used often as an instrument that supports the development of problem-solving skills, and across various disciplines [75]. In this paper, we follow earlier work [49], which defined CT on seven distinct dimensions that are based on observed coding practices (Table 1), specifically for assessing artifacts created with Scratch [63]. Based on the CT dimensions proposed by Dr. Scratch, we explore and assess CT development in student-designed serious games for STEM curriculum that integrate the learning of CT, climate science, and systems thinking.

### 2.2 Game Design as Tool for Learning CT

Game design is emerging in modern education for playful and creative learning [35, 41, 59, 62, 80]; it is the part of game-based learning [40, 77] that leverages *constructionism* [29, 30], and where students construct knowledge by designing digital artifacts [55, 61]. We focus on the use of game design as a tool for scholars to teach and learn CT [2, 24, 28, 50, 51, 85]. Earlier work argued that game design and CT are related and mutually beneficial [2, 28], and how game design can lead to effective CT learning [85]. For instance, previous work showed how middle-school students learning CT via game design score higher in CT tests [4, 53].

Moreno-León et al. have extensively investigated how game design allows for learning and mastery of CT [49–52], arguing how early exposure of students to CT may benefit their overall academic performance in the long run [50], and showing how designing games may lead to higher CT mastery compared to other design practices (e.g., design animations) [48]. However, previous work did not look at how CT and its underlying practices (e.g., logic, abstraction) develop as students design games. We assess the CT of student-designed games created in Scratch [63] via Dr. Scratch [49], and explore CT development.

### 2.3 Scratch

Scratch [63] (Figure 1) is a visual tool based on *block programming* [69], which was designed by MIT specifically to help children learn about programming and CT in a playful way, while creating digital artifacts; its design was inspired by the constructionist tool Logo [55, 60]. Scratch features a library that contains several premade programming blocks, which can be used to program and design interactive applications, including animations and video games [48]. Scratch is used worldwide by an ever growing online community of users, both in formal education [45, 49, 50] and extracurricular activities [30].

Previous work assessing CT in Scratch artifacts showed that the tool allows its users to develop proficiently in CT and easily learn how to program [49, 50, 52]. However, no previous work has investigated CT development in Scratch artifacts. We explore CT development in student-designed serious games with Scratch, to

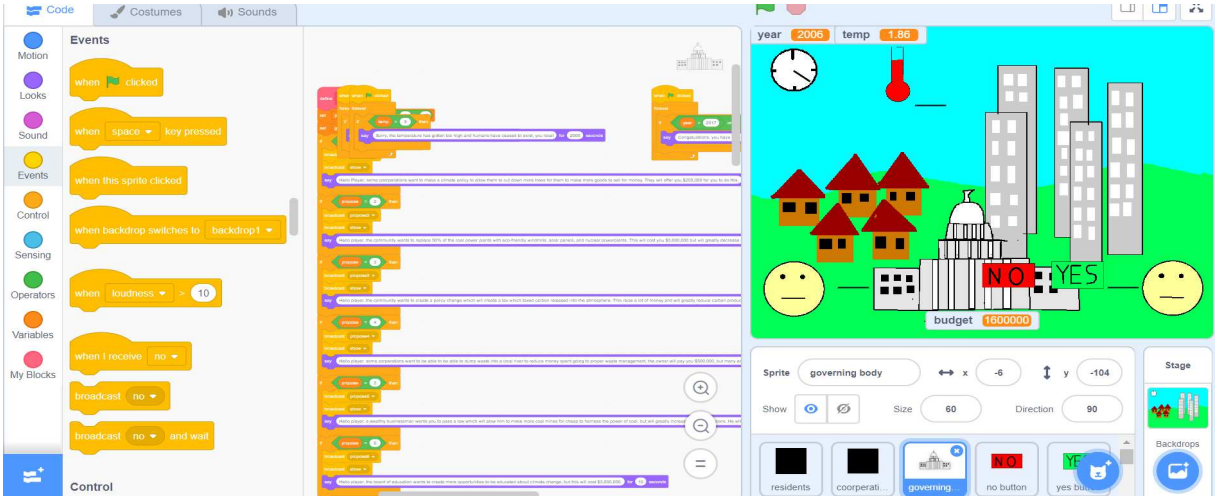


**Figure 1: *Government Simulator*, one of the serious games designed by students in Scratch for the STEM curriculum; the game asks players to take political decisions that may impact the environment either positively or negatively.**

provide insights on how students approach and develop CT in game design for STEM curricula.

## 2.4 Metrics for Assessing CT

Various metrics assess CT. Examples include Bebras [10, 11], the CT-t [66, 72], and Dr. Scratch [49]. Bebras [10, 11] defines a set of tasks or challenges, which assess CT skills based on a two-dimensional system characterized by five CT dimensions, namely *abstraction*, *algorithmic thinking*, *decomposition*, *evaluation*, and *generalization*. The CT-t [66–68] assesses CT skills through a multiple choice test that allows for one correct answer on various programming and CT topics; for instance, the test shows four pictures containing similar code snippets for looping [67], from which students identify the code that works correctly if executed.

We assess the CT of games designed with Scratch, thus we focus on metrics that were created to assess CT in artifacts created with Scratch. Wilson et al. [82] created a coding scheme adapted from Denner et al. [14] to assess students' programming competence in Scratch based on (1) *programming concepts* (e.g., use of conditional expressions), (2) *code organization* (e.g., naming sprites), and (3) *design for usability* (e.g., design user input). Seiter and Foreman [74] introduced the Progression of Early Computational Thinking (PECT) model, which assesses CT by mapping high-level abstractions of *computational thinking concepts* (e.g., problem decomposition), to low-level, measurable *evidence variables* (e.g., use of programming blocks in Scratch). Brennan and Resnick [9] proposed a framework based on (1) *computational concepts* (e.g., parallelism), (2) *computational practices* (e.g., debugging, remix), and (3) *computational perspectives* (i.e., developing a computational mind to understand the surrounding technology), to assess CT through portfolio-based and artifact-based analysis.

Although all viable, the approaches above mentioned rely on manual analysis and human supervision, which make CT assessment time-consuming and hardly scalable to big datasets. To compensate, Moreno-León et al. [49] created a web application based on Hairball [7] called Dr. Scratch[1], which automatically assess CT in Scratch artifacts.

## 2.5 Dr. Scratch

Dr. Scratch was used to assess CT with big datasets (e.g., 250K projects [1]), and has shown to correlate with the judgment of human experts [51]. The Dr. Scratch metrics score CT in Scratch on a scale from 0 to 3 on what the authors call seven "CT dimensions" ( Table 1). Each score defines a level of CT proficiency: 0 = *none*, 1 = *basic*, 2 = *developing*, and 3 = *proficient*, where the total score is the sum of the individual scores for each CT dimension (max 21). The same labels are applied to the overall score (i.e., 1 to 7 is basic, 7 to 14 is developing, and 15 to 21 is proficient). The scores are based on observed coding practices in Scratch. For instance, for *logic*, Dr. Scratch will score 1 point for the use of "if" blocks, 2 points for "if-else" blocks, and including logic operators (e.g., AND, OR, etc.) in these blocks will score 3 points.

# 3 METHOD

We use Dr. Scratch [49] to assess CT in student-designed serious games for STEM, and analyze the results using descriptive statistics, cluster analysis, and data visualization. Note that, unlike prior work [48], we consider both the final CT score given by Dr. Scratch (i.e., the "CT proficiency"), and how CT evolves over time (i.e., the "CT Development"). The serious games were designed by 8$^{th}$-grade students as part of a STEM curriculum focused on the learning in parallel of CT, climate science, and systems thinking.

## 3.1 STEM Curriculum for CT Learning

We assess serious games designed by 8$^{th}$-grade students for an innovative STEM curriculum. The curriculum uses game design as means to foster the learning of CT, climate science, and systems thinking [58]. The main task for students is to program and design

[1] http://www.drscratch.org/

**Table 1: *Dr. Scratch* metrics, showing competence level for each CT dimension and relative *Scratch* practices; adapted from [47].**

| CT Dimension | Competence Level | | | |
|---|---|---|---|---|
| | Null (0) | Basic (1) | Developing (2) | Proficient (3) |
| Abstraction | — | More than one script and more than one sprite | Definition of blocks | Use of clones |
| Data representation | — | Modifiers of sprite properties | Operations on variables | Operations on lists |
| Flow control | — | Sequence of blocks | Repeat, forever | Repeat until |
| Logic | — | If | If else | Logic operations |
| Parallelism | — | Two scripts on green flag | Two scripts on key pressed, two scripts on sprite clicked on the same sprite | Two scripts on when I receive message, create clone, two scripts when %s is >%s, two scripts on when backdrop change to |
| Synchronization | —- | Wait | Broadcast, when I receive message, stop all, stop program, stop programs sprite | Wait until, when backdrop change to, broadcast and wait |
| User interactivity | — | Green flag | Key pressed, sprite clicked, ask and wait, mouse blocks | When %s is >%s, video, audio |

serious games in Scratch, which represent gamified versions of climate change topics of their choice (e.g., CO2 emission).

The curriculum started with students and teachers exploring together the various phenomena that relate to climate change (e.g., greenhouse effect and CO2 emission, deforestation). After the students picked a climate change topic to explore via game design, the students were exposed to serious games; they were asked to play and critique existing serious games designed for educational purposes (e.g., *Offset*[2] and *Powerup*[3], two serious games designed by NASA), to understand how such games are designed and how they frame educational content. To help students structure the design of their serious games, the teachers also introduced them to the Triadic Game Design model [27].

Then, the students were introduced to Scratch through an activity called "10-Block Challenge", where they could practice how to code using only 10 given blocks. Once the teachers felt that the students had a sufficient understanding of climate science, designing serious games, and programming in Scratch, they let them develop their final projects in pairs.

The curriculum was implemented in 35 science classes over two years, at three different schools by nine teachers, and involved 8$^{th}$ grade students (13 to 14 years old, approximately 19 students per class). Prior to the curriculum implementation, all teachers participated in a professional-development program, which facilitated their understanding of programming in Scratch and CT. While the program aimed to address teachers' knowledge gaps and comfort level [46], it is still important to note that most teachers had no prior experience with either Scratch or game design.

The teachers were given the freedom to adjust the curriculum according to their available class time and objectives. The implementation took between four to six weeks, and while all teachers followed the curriculum closely, differences were noticeable among their implementations. Because the game design activity was open-ended, the range of games developed by students varied widely, from simple mini-puzzle games, to more elaborate simulation or narrative-based adventure games (e.g., Figure 1).

We focus on the analysis and assessment of CT development. Hence, we disregard contextual information about (1) curriculum implementation, (2) role of teachers, (2) genres of student-designed games, and (3) teamwork dynamics between student pairs. We acknowledge that such variables may have partly influenced the results, but consider their analysis out of scope in the present work.

## 3.2 Dr. Scratch Assessment and Data Collection

We use Dr. Scratch [49] to assess CT proficiency in 317 and CT development in 217 serious games designed by students in Scratch. However, differently from earlier work using Dr. Scratch [50, 52], the students who designed the games were not provided with real-time feedback on the status of their CT score, to avoid biasing their design choices. Upon teachers' supervision, students were instructed to upload the link to their Scratch projects on an external repository, from which we retrieved data for later analysis.

We tracked and collected the data contained in the Scratch compressed .sb2 files (e.g., changes in code, time-stamps) using the Scratch API. We kept track of the Scratch projects and their updates using a Python script, which took a "snapshot" of the projects every minute; we found such interval to be a good trade-off between capturing significant changes in code and limiting computational expense. We collected a total of 22,087 snapshots.

## 3.3 Data Cleaning

We collected a total of 435 Scratch projects. We found 45 corrupted or abandoned projects, which we filtered out. Additionally, we excluded 73 projects that students made for the "10-Block Challenge", as these were preparatory exercises and did not reflect mature CT development. The data filtering led to a final dataset of 317 Scratch projects, which we used to assess the overall CT proficiency, as

[2]http://climatekids.nasa.gov/offset/
[3]http://climatekids.nasa.gov/power-up/

well as CT proficiency in each of the seven CT dimensions defined by Dr. Scratch (Table 1).

Furthermore, we analyzed how CT proficiency developed in time as students designed their games in Scratch. To analyze CT development, we looked at sequences of snapshots over the life span of Scratch projects (from start to end), and observed how the Dr. Scratch score evolved. As we analyze CT development based on quartiles, we require a minimum of five snapshots per project. This further reduced the number of analyzable projects for CT development to 217 .

## 3.4 Data Analysis

We conducted two analyses: (1) a CT proficiency analysis and (2) a CT development analysis. The CT proficiency analysis looks only at the final CT score in Dr. Scratch for the 317 student-designed serious games. For the CT development analysis instead, we tracked the evolution of Dr. Scratch scores for 217 Scratch projects using the collected snapshots.

As explained earlier, we collected the projects snapshots by tracking changes in code at regular intervals (i.e., one minute), and stored snapshots only when changes in codes occured. Similar data extraction and collection process are used in software evolution [31], where changes are tracked by project *commits* or *saves*.

Since the numbers of snapshots varied greatly in Scratch projects, ranging from 5 to 375, to represent CT development across projects coherently, we normalized the number of snapshots by retrieving representative quartiles at evenly distributed positions. For instance, if a project has 8 snapshots, $Q_0$ would be snapshot 1, $Q_4$ would be snapshot 8, while $Q_1 - Q_3$ would be evenly distributed from the rest of the snapshots (i.e., snapshots 2, 4, and 6, respectively). Using the quartiles, we analyze CT development across 217 projects and explore how each CT dimension develops over time.

To further investigate trajectories of CT development, we perform cluster analysis using $k$-medoids with $k = 9$ ($k$ chosen via the silhouette method [36]), and the sum-squared distance between the total CT score at each quartile as the distance metric. Finally, for both the CT proficiency and the CT development analyses, we looked at the Dr. Scratch score for the overall CT proficiency (from 0 to 21), as well as the score for each CT dimension (from 0 to 3).

# 4 RESULTS

We first present results from the analysis of the final Dr. Scratch scores, which reflect the CT proficiency of 317 Scratch projects (i.e., student-designed serious games). Then, we explore the CT development of 217 student-designed games, and show results from the analysis of the Dr. Scratch scores evolving over time as students design and program serious games in Scratch.

## 4.1 CT Proficiency

We show the results from assessing CT proficiency in 317 student-designed serious games using Dr. Scratch. First we show the overall CT proficiency achieved in Scratch projects (i.e., from 0 to 21), then we show the proficiency of each CT dimension (i.e., from 0 to 3).

*4.1.1 Overall CT Proficiency.* Figure 2 shows the distribution and frequency for the total Dr. Scratch scores. As said earlier, Dr. Scratch divides CT proficiency into three levels (Table 1): (1) basic (from 1 to 7), (2) developing (from 8 to 14), and (3) proficient (from 15 to 21). We refer to those labels while reporting the results to describe different levels of CT proficiency.

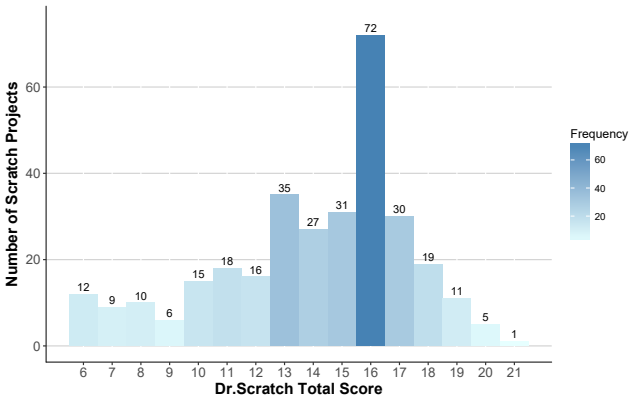


**Figure 2: Total Dr. Scratch scores of student-designed games.**

Overall, student-designed games showed developing CT proficiency ($M$ = 14.07, $SD$ = 3.33). Among 317 student-designed games, 72 reached a score of 16 and showed CT proficiency. More than half of the projects ($n$ = 179) scored between 14 ($n$ = 27) and 18 ($n$ = 19), showing also CT proficiency. The rest deployed either basic or developing CT ($n$ = 121), and few reached high CT proficiency ($n$ = 16, scoring 19-20). Only one student-designed serious game scored the maximum (i.e., 21 CT points). These results corroborate the underlying conjecture of the STEM curriculum that designing serious games allows students to become proficient in CT.

*4.1.2 Proficiency of Individual CT Dimensions.* Figure 3 shows the Dr. Scratch score frequency for each individual CT dimension. Abstraction ($M$ = 1.34, $SD$ = 0.72) and user interactivity ($M$ = 1.88, $SD$ = 0.37) show the highest frequencies at CT score 1 ($n$ = 251) and CT score 2 ($n$ = 267), respectively; this indicates that overall, student-designed games showed basic proficiency in abstraction (i.e., breaking a problem down into smaller parts) and developing in user interactivity (i.e., programming user input and interactive contents). As most games showed basic proficiency in abstraction, that means few of the Scratch projects in analysis defined blocks or made use of clones, which are considered "advanced coding practices" in Scratch for this particular CT dimension. However, in those few projects that showed developing or proficient in abstraction, we found that using clones was more frequent ($n$ = 46) than defining blocks ($n$ = 11). This shows discrepancy between the affordances of current game design tools and CT metrics scoring criteria. In fact, while Dr. Scratch scores using clones higher (i.e., 3 points) than defining blocks (i.e., 2 points), in Scratch defining blocks is harder, as it involves more than simple code cut-and-paste (i.e., cloning). This finding is in line with the results reported in previous research [64], where the practice of cut-and-paste instead of the more "appropriate" definition of blocks was frequently identified.

Regarding user interactivity, student-designed games showed developing proficiency in this CT dimension. That means most games were designed for user-input, and allowed for interaction

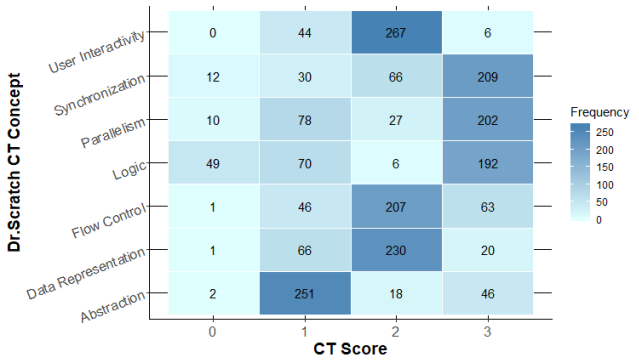


**Figure 3: Heatmap showing the Dr. Scratch scores for each individual CT dimension and their frequencies.**

through keyboard or mouse input. These results are not surprising, given that the Scratch projects made by students were games, which require interactivity by definition [27]. However, only few games ($n$ = 6) embedded video or audio content that is triggered by user input, which Dr. Scratch scores 3 in user interactivity. Interestingly, Scratch projects that score 3 in user interactivity were also uncommon in previous work [48].

Proficiency in flow control ($M$ = 2.05, $SD$ = 0.59) and data representation ($M$ = 1.85, $SD$ = 0.51) showed developing in student-designed games (CT score 2, $n$ = 207 and 230, respectively). For flow control (i.e., programming and handling sequential events), this means that students had an understanding of how to handle basic block sequences and were able to create loops that repeat forever, while they have had less understanding (or less need) of how to use loops that repeat until a certain condition is met (i.e., Boolean block is true).

For data representation (i.e., setting and retrieving information), the projects reflect students' understanding of basic mathematical operations that they used to modify sprite properties, creating, and handling variables that represent data (e.g., time counter). However, few projects showed competency with operations on lists, meaning that such data representation practices were hard to develop, or not always needed by students when designing their serious games.

Regarding logic ($M$ = 2.07, $SD$ = 1.2), parallelism ($M$ = 2.33, $SD$ = 0.95), and synchronization ($M$ = 2.49, $SD$ = 0.81), the games were proficient in all three CT dimensions (CT score 3, $n$ = 192, 202, and 209, respectively). For logic (i.e., use of conditional operations), this means that the projects reflected students' understanding of how to handle basic logic in Scratch (e.g., *if-then*). However, we observed a "jump" from score 1 to 3 in this dimension, suggesting that students may move directly from using simple *if* blocks, to using more complex logic operators (i.e., CT score of 3 for the use of binary operators such as AND and OR).

While the use of *if-else* blocks (i.e., CT score 2) may be comprised already in advanced logic operations, the use of binary operators does not ensure that learners have previously used *if-else* blocks. Therefore, the absence of *else* blocks should be signalled in some way by Dr. Scratch, as it hints to a lack of linear development in logic. In addition to the jumps in scores, it is important to notice that in logic and parallelism, although the highest frequency of scores are 3, there are many projects with scores of 0 ($n$ = 49 for logic) and 1 ($n$ = 70 for logic, $n$ = 78 for parallelism). This suggests that the metrics for these two CT dimensions may need re-calibrating, or that game design may not ensure that such dimensions are learned in a linear fashion.

For parallelism (i.e., a series of events occurring simultaneously), the student-designed games showed advanced use of multiple scripts and clones in Scratch; this may have been deployed in as changes in backgrounds, creation and multiplication of in-game objects (e.g., targets), or defining time constraints (e.g., when timer > 45, end game). Finally, for synchronization (e.g., creating and handling synchronized events), the games showed proficient CT, thus frequently using broadcast blocks and elements from the backdrops library (i.e., backgrounds). In sum, the student-designed games showed proficient CT in logic, parallelism, and synchronization, developing in data representation, flow control, and user interactivity, and basic in abstraction.

## 4.2 CT Development

In this section, we present the results of the CT-Development analysis for 217 student-designed serious games, and show how each CT dimension developed over time as students designed their games.

*4.2.1 Quartile Analysis.* Figure 4 shows the development of the project scores across the quartiles (see Section 3) for each of the seven Dr. Scratch CT dimensions. Note that, although we expected the first snapshots of projects at $Q_0$ to be always 0 points in all CT dimensions, we see that (1) user interactivity always starts above 0 (i.e., Dr. Scratch scores 1 in this dimension for *Green flag*, which block is added automatically by Scratch to newly created projects), and (2) 183 projects score above 0 in multiple CT dimensions already at the start. This may be due to one of the following:

(1) During the snapshot collection, initial progress in a project might have not been available. This could be because some students may have uploaded the link to their Scratch projects after some initial progress was already made.

(2) Some Scratch projects might have been remixes, namely they were based on already existing projects, and developed further based on previously existing coding. This would mean that their initial CT scores would include pre-implemented Scratch blocks and codes, and thus deploying already some degree of CT proficiency at the start.

Overall, all CT dimensions show development in CT across quartiles. However, they differ in how and when they develop. The least development by far is noticeable with abstraction. It starts basic and does not show observable improvement throughout. The distribution ranges from null to proficient, suggesting that only at $Q_4$ ($Mdn$ = 1, $IQR$ = 1-2) CT development in abstraction can be observed, and that this widely differs per project. User interactivity, data representation, and flow control have similar but distinct trajectories. User interactivity ranges from null to proficient CT in $Q_0$ ($Mdn$ = 2, $IQR$ = 1-2), but does not show further development in the other quartiles, with the exception of a few outliers (see Figure 4), while the rest achieve a score of 2 in $Q_1$ and keep that score onward.

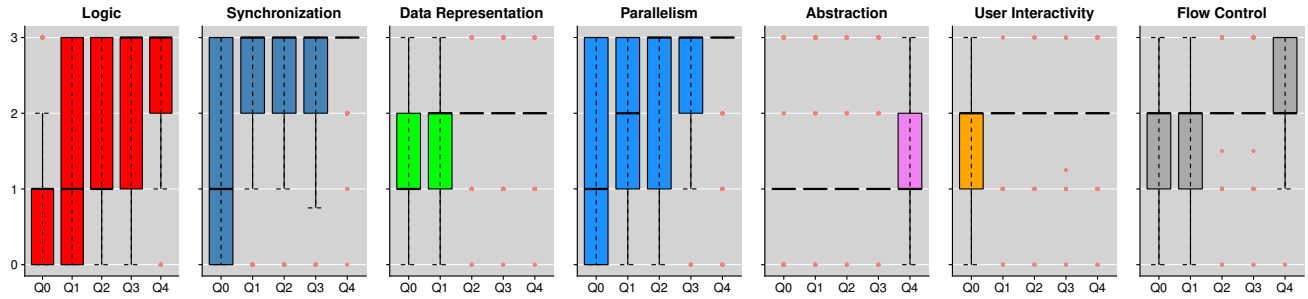


**Figure 4: Distribution of Dr. Scratch CT scores for individual CT dimensions across Scratch project quartiles; dashed lines represent the error bars; red dots are the outliers.**

Data representation shows development trajectories similar to user interactivity. However, it does not show development nor any distribution beyond $Q_2$, suggesting that this dimension consolidates or is approached later than user interactivity in student-designed games. Flow control develops similarly to data representation, but continues improving slightly until $Q_4$ (*Mdn* = 2, *IQR* = 2-3). A common element among these four dimensions is that, while development is noticeable, they mostly remain developing, and hardly reach high CT proficiency. This suggests that (1) there may be barriers to developing these CT dimensions proficiently, (2) the student-designed games did not need to develop proficiently in those dimensions because of specific design choices, or (3) the Dr. Scratch metrics need re-calibrating for the context in which CT practices are deployed.

Then, by and large, synchronization, parallelism, and logic have similar but yet distinct patterns. Synchronization shows the fastest improvement; it develops from basic to proficient already at $Q_1$ (*Mdn* = 3, *IQR* = 2-3), and while showing possibility for improvement in $Q_1$–$Q_3$, in most projects it sets on proficient CT early, and shows no further development in $Q_4$. Parallelism, instead, develops somewhat similarly but slower and more gradually: it develops from basic in $Q_0$ (*Mdn* = 1, *IQR* = 0-3), to developing in $Q_1$ (*Mdn* = 2, *IQR* = 1-3), then to proficient in $Q_2$ (*Mdn* = 3, *IQR* = 1-3), while $Q_4$ shows no further development. Logic develops even slower than parallelism and at $Q_4$ shows still chances of development. It is still basic at $Q_2$ (*Mdn* = 1, *IQR* = 1-3), but then "jumps" to proficient at $Q_3$ (*Mdn* = 3, *IQR* = 1-3).

*4.2.2 Cluster Analysis.* In the previous section we identified development trends for the individual CT dimension across 217 projects. However, although these trends provide useful insights into CT development, they do not show the extent to which different projects develop CT with similar trajectories, which may have been lost when averaging CT development across projects. Hence, we further inquire trajectories of CT development through cluster analysis (see Section 3). Table 2 shows the number of projects belonging to each of the nine clusters we identified. Figure 5 shows how the different clusters evolve across the quartiles on the seven CT dimensions. Below, we describe the CT development trajectories that emerge from the cluster analysis.

Cluster 1 (Figure 5a) shows steep improvement in all CT dimensions from the first snapshot to $Q_1$, going from 0 to 2 in five CT dimensions, while abstraction and logic stay on 1. Between $Q_1$ and $Q_4$, logic shows the steepest curve of improvement, which reaches high proficiency in $Q_4$, while abstraction is the only CT dimension that stays below 2. The way in which each CT dimension develops within Cluster 3 (Figure 5c) has similar trends with Cluster 1. However, in Cluster 3 most CT dimensions start from above the score of 0 already at $Q_0$. Regardless, these two clusters are marked by a steep initial increase and then a gradual improvement across the remaining quartiles towards proficiency, with the exception of logic, which increases rather linearly.

Clusters 2 and 6 (Figures 5b and 5f, respectively) show CT development trajectories similar to Clusters 1 and 3 (mean snapshots: 150.5 and 141.3), but seem to improve more rapidly (mean snapshots: 73.1 and 80.1). However, when comparing Cluster 2 to Cluster 6, the latter shows higher competency in logic, synchronization, and parallelism, and more development in abstraction throughout. Furthermore, Cluster 6 (but also Cluster 7) develop abstraction and logic more proficiently than other clusters, suggesting that in these we may be assessing projects that were designed by students with prior knowledge of Scratch, or that such projects may have been remixes of previously existing projects. Despite these differences, Clusters 2 and 6 illustrate CT development trajectories

| Cluster # | Projects in Cluster | Mean Snapshots/Project |
|---|---|---|
| 1 | 25 | 150.5 |
| 2 | 57 | 73.1 |
| 3 | 31 | 141.3 |
| 4 | 31 | 85.1 |
| 5 | 24 | 86.3 |
| 6 | 36 | 80.1 |
| 7 | 9 | 52.4 |
| 8 | 2 | 9.5 |
| 9 | 2 | 43.0 |

**Table 2: This table shows the number of projects belonging to each cluster and the average number of snapshots for the projects within those clusters.**

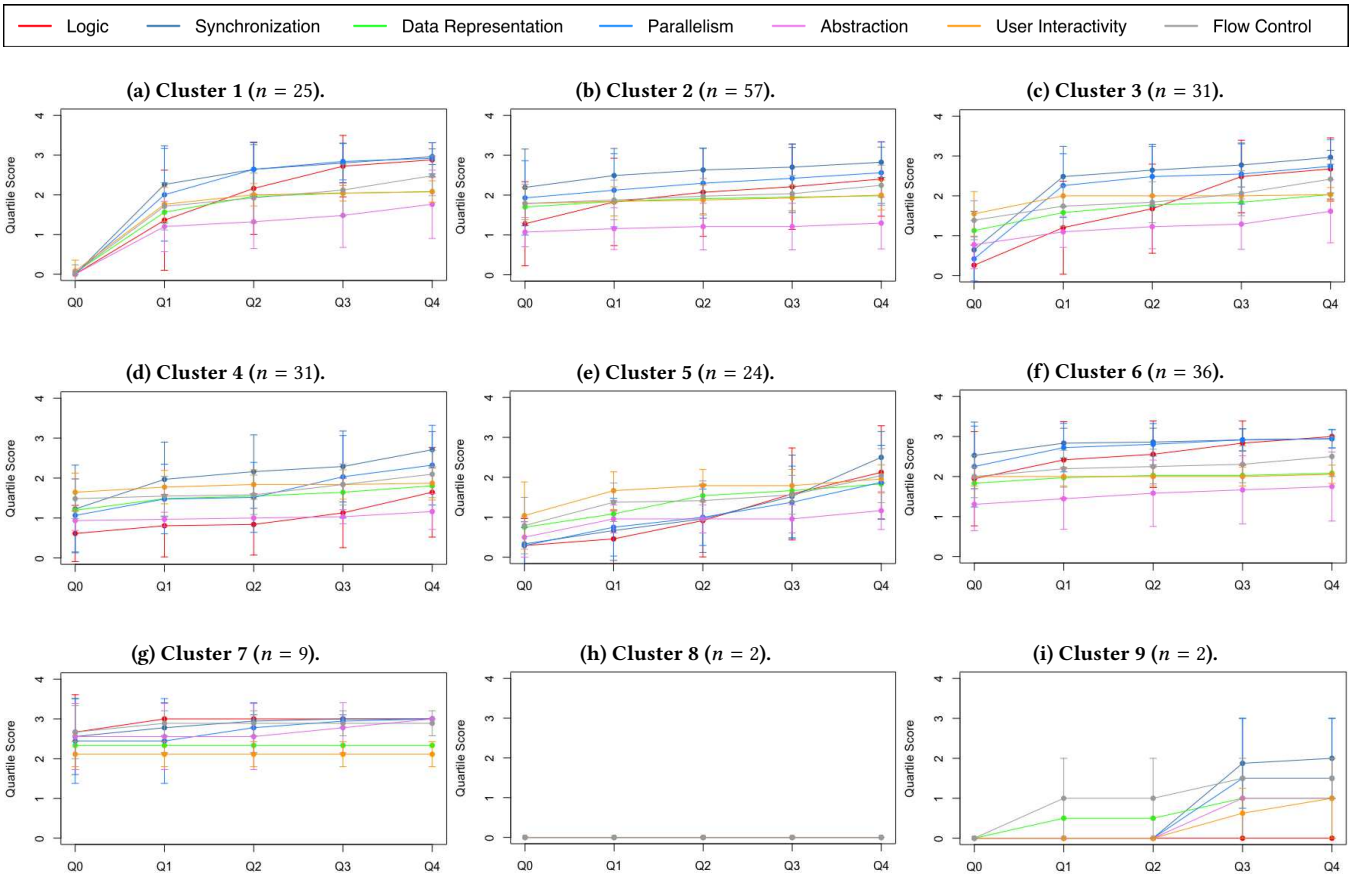


**Figure 5: The average scores over the quartiles for the projects in each cluster.**

that go from basic to developing fast in the first two quartiles and gradually improve towards proficiency in the last quartile.

Cluster 4 (Figure 5d) is the cluster where logic shows the least development and where parallelism develops less steadily and rapidly compared to other clusters. Interestingly, projects from Cluster 4 developed over slightly more iterations (mean snapshots: 85.1) than Clusters 2 and 6 (mean snapshots: 73.1 and 80.1), but still show less development in logic and parallelism compared to those. This means that, although projects in Cluster 4 had more time to develop compared to Clusters 2 and 6, students designing games in this cluster may have found difficult, or may have not need to use (1) developing Scratch practices for logic as defined by Dr. Scratch (i.e., using if-else and logic operators like AND and OR), or (2) creating clones for parallelism. While it may be that students designing games in Cluster 4 did not need to use advanced Scratch practices for the aforementioned CT dimensions, overall the cluster displays little development on all CT dimensions.

Cluster 5 (Figure 5e) shows unique CT development trajectories compared to other clusters. For instance, logic, abstraction, parallelism, and synchronization struggle to develop within the first two quartiles (i.e., score $< 1$ for $Q_0$–$Q_2$) and it is the only cluster where logic and flow control develop slightly more proficiently than parallelism; it is also the only cluster that shows one of the CT dimensions (i.e., abstraction) slightly decreasing in proficiency between $Q_1$ and $Q_3$. Similarly to Cluster 4, Cluster 5 does not show progress towards CT proficiency.

The remaining clusters (i.e., Clusters 7, 8, and 9) contain fewer projects compared to the other clusters (see Table 2). In particular, Cluster 8 contains only two projects and no CT development is observable; these are likely projects that were started and immediately abandoned. Cluster 9 contains two projects too, but shows some CT development compared to Cluster 8. This is because, projects from Cluster 9 may have been initially developed by students with the intent to complete them, but were then abandoned or moved to new Scratch projects.

Finally, Cluster 7 (Figure 5g) also contains few projects ($n = 9$), but shows CT development throughout, which eventually leads to high CT proficiency; this is also the cluster where we see the highest overall CT development. Considering the few number of snapshots, Cluster 7 may include Scratch projects that were designed by students with prior knowledge of the tool, or that developed their Scratch code based on preexisting Scratch projects (i.e., they remixed code that was already implemented by other Scratch users).

## 5 DISCUSSION

We assessed and analyzed student-designed serious games with Scratch for STEM curricula using Dr. Scratch. We looked at both the global (overall CT scores) and the atomic (scores for each CT dimension) levels of CT practices in students' games, and reported results on their final product (CT Proficiency) as well as their development over time (CT Development). Here, we reflect on the insights from the assessment and analysis of CT development in student-designed serious games for STEM, outline the lessons learned from metric-based CT assessment using Dr. Scratch, and consider limitations to our work.

### 5.1 Insights into CT Development

Previous work using Dr. Scratch looked at how CT develops towards proficiency within different design practices in Scratch (e.g., art design, music composition, game design), and showed that CT develops most in game design [48]. Our assessment of CT proficiency reinforces such evidence and argues that game design may be effective for supporting students in developing CT proficiently in STEM curricula (i.e., most student-designed serious games in analyses showed developing and high CT proficiency).

In breaking down the overall score, we showed that student-designed games deployed CT proficiency in synchronization, parallelism, and logic, while abstraction was basic; prior investigations assessing Scratch games with Dr. Scratch showed similar trends [48]. Therefore, while the present work focuses on CT practices deployed in the context of STEM curricula integrating game design, climate science, and systems thinking, our results highlight that from a Dr. Scratch perspective, the results are higher compared to other non-game design practices and at least equivalent to other CT practices involving game design. This means that while improvements are possible, the newly developed STEM curriculum was effective in teaching CT, despite teachers' limited knowledge and experience with Scratch and CT and the additional demands and constraints placed both on teachers and students in such an integrative curriculum as presented in this paper.

Compared to previous assessment that considered only the final Dr. Scratch scores [48], we add the analysis and assessment of CT development over time, as students created and designed serious-games in Scratch. We observed that synchronization, parallelism, data representation, flow control, and user interactivity developed early in student-designed games, but that data representation and user interactivity soon stopped improving and settled on developing CT proficiency (i.e., CT score 2) before students were even halfway through their Scratch projects. By contrast, logic shows a slow start but continues developing until the end to eventually reach CT proficiency (i.e., CT score 3).

Additionally, we observed various trajectories of CT development, which can be described in four ways: (1) *quick-then-steady*: those projects that make a short burst at the onset, then gradually improve their CT until they reach high proficiency (26%); (2) *steady-all-the-way*: such projects gradually and steadily improve CT throughout, with no observable steep curves or sudden changes in score (42%); (3) *slow-and-still-developing*: projects that slowly develop CT proficiency and do not manage to reach the high CT proficiency in any of the seven dimensions, but remain developing (25%); and (4) *no-improvement-necessary*: those projects that show no observable improvement, as their CT is already proficiency at the start (4%). Further investigation, however, is necessary to better understand such CT trajectories.

Interestingly, we observed that the majority of games we assessed developed poorly in abstraction and data representation; these results were similar in previous CT assessments based on Dr. Scratch [48, 50]. However, we find these results somehow surprising, when considering the context in which students designed their games. One main goal of the STEM curriculum was to teach young students about systems thinking, which involves developing problem-solving skills (e.g., [44]). In that respect, it is interesting that the two CT dimensions that mostly relate to problem-solving (i.e., abstraction and data representation) are the ones that developed the least. This may be due to an imbalance in the implementation of the STEM curriculum or the requested task (i.e., game design), which might have shifted students' focus more towards designing the actual game rather than reflecting on system thinking and climate science, and favored a higher development of CT dimensions like user interactivity.

### 5.2 Lessons Learned from Metric-Based CT Assessment using Dr. Scratch

While there are broadly accepted definitions on what CT means [79], measuring and assessing CT remains challenging. Dr. Scratch proposes a set of metrics that can be used to automatically assess CT proficiency and CT development in Scratch artifacts, and has demonstrated its reliability [52]. This system can be critiqued or changed (see also [28]) in several ways, the most relevant one being the inherent limitations of any automatic assessment approach; however, our work found a number of specific lessons learned.

First, we found discrepancies between how Dr. Scratch assesses CT competencies in Scratch, and how much CT proficiency is actually needed for the associated coding practices in Scratch. For instance, defining blocks in Scratch (which Dr. Scratch scores 2 CT points in abstraction) is harder than using clones (which is scored 3 CT points in the same dimension). Once a clone is created (which Dr. Scratch scores 3 in parallelism), using clones is relatively easy to implement; as such, creating a clone may automatically result in getting 3 points in abstraction for simply using clones. Instead, defining custom blocks in Scratch assumes higher programming skills, as students need to have a more solid and deeper understanding of how Scratch works, to be able to create blocks that do not already exist in the Scratch environment, rather than using blocks that are already available.

Furthermore, the Dr. Scratch metrics for abstraction may inadvertently promote what Robles et al. [65] defined as *bad smells* (i.e., coding practices that are not recommended), such as copy-and-paste of code chunks, as they score high. We observed that logic developed straight from basic to high CT proficiency, in fact "jumping" directly from 1 to 3. This may be due to imbalance in the Dr. Scratch metrics for logic, which vaguely define "logic operators" as the highest level of CT proficiency. Finally, as seen in previous work [48], we showed that data representation and user interactivity "saturate" when reaching developing CT competency. This trend should be further investigated in future work with Dr.

Scratch, to understand why the use of advanced coding practices for such CT dimensions are uncommon, and whether the reason lies in how Dr. Scratch defines their metrics.

### 5.3 Limitations and Generalization

We see limitations and a number of potential threats to validity in our work, for which we aim to compensate in future work. First, we purposely focused on assessing CT proficiency and CT development of student-designed serious games via metrics. As such, we did not include in our analysis the qualitative aspects of such games and the information related to the curriculum in which they were developed (i.e., influence of different teaching styles on the development of CT practices). For instance, we did not consider the genre of the games in question (e.g., narrative, action). The serious games were designed by students with different prior knowledge of Scratch and programming, under the supervision of different teachers, whose different teaching methods might have in part influenced the way CT was developed and deployed by students throughout the projects. Future work should include such contextual information, to further understand how particular game design choices, different teacher's approaches, and different levels of prior knowledge of programming and CT in students, influence the development of CT trajectories that lead to CT proficiency.

Another limitation to our approach is represented by how we collected the data for our analysis. As noted in Section 4.2, a considerable number of projects showed already a degree of CT development from the first snapshot, which hints to potential missing information or projects that started as a remix (which, notably, is information that cannot be retrieved from the .sb2 Scratch project files). However, at present our data scraping and collection system does not allow us to identify whether a project is a remix of a previously existing Scratch project.

Furthermore, it is possible that teachers implementing the curriculum did not always ensure that students (correctly) uploaded links to their Scratch projects in our repository before starting to develop the coding. We should compensate for that shortcoming in future work by (1) ensuring that teachers implementing the curriculum promptly instruct students to upload the link to their Scratch projects as they develop them, and (2) finding a systematic way to identify and distinguish Scratch projects that are remixes from the ones that start from scratch. Previous work has identified remixes among Scratch projects; however, it is unclear how this was achieved [1]. Third, the student-designed games of which we assessed CT proficiency and CT development were created and designed for specific educational goals (i.e., teaching young students about CT, climate science, and systems thinking via game design). This may represent both a strength and a weakness.

On the one hand, our student-designed games provide insight on CT development that are grounded in specific design practices (i.e., designing educational games on climate change). Therefore, researchers that are particularly interested in the context in which our results were produced may benefit from our effort. On the other hand, because we investigated CT development in such a specific context, our results may be hard to generalize. We offer the results of a case study, with its particularities and peculiarities. We thus cannot claim that our results and observations can be generalized to other types of projects (e.g., animations, story-telling) and contexts (e.g., other programming languages or learning environments).

However, the value of case studies should not be underestimated; Flyvbjerg [19] provides several examples of individual cases that contributed to discoveries in areas such as physics, economics, and social sciences. As a case study, our work serves as an example that, from a CT perspective, game design represents a valid tool that can effectively integrate the teaching of CT along with other topics (i.e., climate science), particularly in STEM curricula. Future work should go beyond and assess the CT development of student-designed games produced in diverse contexts, to see if the trends are common across different disciplines and educational goals, and compare the results to produce a broader overview of CT development via game design.

## 6 CONCLUSION

We assessed CT in student-designed games with Dr. Scratch and showed that game design in STEM curricula lead to CT proficiency. We provided insights into CT development in game design for STEM, showing that synchronization, parallelism, flow control, and logic develop proficiently in such context when assessed by Dr. Scratch. Furthermore, we identified potential flaws in the Dr. Scratch metrics and suggested improvements. Finally, we showed how existing CT metrics can be used to explore and assess CT development in student-designed games, but that they need recalibrating and further improvements.

## ACKNOWLEDGMENTS

This project is supported by the NSF under Grant No. 1542954. We also thank Ángela Vargas Alba for her help with the data analysis, and our collaborating teachers, school districts, and students.

## SELECTION AND PARTICIPATION OF CHILDREN

Children participated as students of a STEM curriculum developed together by a team of teachers and experts on constructionist learning and game design. All the data we have collected for our assessment and analysis used consent of teachers and student parents, and children assent. The consent form described (1) the tasks related to the curriculum (i.e., designing games in Scratch, learning climate change, and systems thinking), (2) what kind of research we would conduct using the student-designed games, and (3) clearly stated that the data would be treated with discretion and children identities would be protected (i.e., the student pairs used teachers-assigned pseudonyms when loading their projects on the Scratch website).

## REFERENCES

[1] Efthimia Aivaloglou, Felienne Hermans, Jesús Moreno-León, and Gregorio Robles. 2017. A dataset of scratch programs: scraped, shaped and scored. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 511–514.

[2] Jackie Barnes, Amy K. Hoover, Borna Fatehi, Jesus Moreno-Leon, Gillian Smith, and Casper Harteveld. 2017. Exploring Emerging Design Patterns in Student-made Climate Change Games. In *Proceedings of the 12th International Conference on the Foundations of Digital Games (FDG '17)*. ACM, New York, NY, USA, Article 64, 6 pages. https://doi.org/10.1145/3102071.3116224

[3] Valerie Barr and Chris Stephenson. 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 (Feb. 2011), 48–54. https://doi.org/10.1145/1929887.1929905

[4] Ashok R. Basawapatna, Kyu Han Koh, and Alexander Repenning. 2010. Using Scalable Game Design to Teach Computer Science from Middle School to Graduate School. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '10)*. ACM, New York, NY, USA, 224–228. https://doi.org/10.1145/1822090.1822154

[5] Ahmet Baytak and Susan M. Land. 2010. A case study of educational game design by kids and for kids. *Procedia - Social and Behavioral Sciences* 2, 2 (Jan. 2010), 5242–5246. https://doi.org/10.1016/j.sbspro.2010.03.853

[6] Linda Behar-Horenstein and Lian Niu. 2011. Teaching Critical Thinking Skills In Higher Education: A Review Of The Literature. *Journal of College Teaching and Learning* 8, 2 (02 2011), 25–41. http://ezproxy.neu.edu/login?url=https://search.proquest.com/docview/857923396?accountid=12826 Copyright - Copyright Clute Institute for Academic Research Feb 2011; Document feature - Charts; ; Last updated - 2011-03-21; SubjectsTermNotLitGenreText - United States–US.

[7] Bryce Boe, Charlotte Hill, Michelle Len, Greg Dreschler, Phillip Conrad, and Diana Franklin. 2013. Hairball: Lint-inspired Static Analysis of Scratch Projects. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. ACM, New York, NY, USA, 215–220. https://doi.org/10.1145/2445196.2445265

[8] Christian P. Brackmann, Marcos Román-González, Gregorio Robles, Jesús Moreno-León, Ana Casali, and Dante Barone. 2017. Development of Computational Thinking Skills Through Unplugged Activities in Primary School. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education (WiPSCE '17)*. ACM, New York, NY, USA, 65–72. https://doi.org/10.1145/3137065.3137069

[9] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *In AERA 2012*.

[10] Valentina Dagiene and Gabriele Stupuriene. 2016. Bebras–A Sustainable Community Building Model for the Concept Based Learning of Informatics and Computational Thinking. *Informatics in Education* 15, 1 (2016). https://eric.ed.gov/?id=EJ1097494

[11] Valentina Dagienė, Gabrielė Stupurienė, and Lina Vinikienė. 2016. Promoting Inclusive Informatics Education Through the Bebras Challenge to All K-12 Students. In *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016 (CompSysTech '16)*. ACM, New York, NY, USA, 407–414. https://doi.org/10.1145/2983468.2983517

[12] Jill Denner, Shannon Campe, and Linda Werner. 2019. Does Computer Game Design and Programming Benefit Children? A Meta-Synthesis of Research. *ACM Trans. Comput. Educ.* 19, 3, Article 19 (Jan. 2019), 35 pages. https://doi.org/10.1145/3277565

[13] Jill Denner, Linda Werner, and Eloy Ortiz. 2012. Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education* 58, 1 (Jan. 2012), 240–249. https://doi.org/10.1016/j.compedu.2011.08.006

[14] Jill Denner, Linda Werner, and Eloy Ortiz. 2012. Computer Games Created by Middle School Girls: Can They Be Used to Measure Understanding of Computer Science Concepts? *Comput. Educ.* 58, 1 (Jan. 2012), 240–249. https://doi.org/10.1016/j.compedu.2011.08.006

[15] Anna Eckerdal, Michael Thuné, and Anders Berglund. 2005. What Does It Take to Learn 'Programming Thinking'?. In *Proceedings of the First International Workshop on Computing Education Research (ICER '05)*. ACM, New York, NY, USA, 135–142. https://doi.org/10.1145/1089786.1089799

[16] Yoram Eshet. 2004. Digital Literacy: A Conceptual Framework for Survival Skills in the Digital era. *Journal of Educational Multimedia and Hypermedia* 13, 1 (January 2004), 93–106. https://www.learntechlib.org/p/4793

[17] Charles Fadel and Bernie Trilling. 2010. 21st Century Skills: Learning for Life in Our Times. *Education Review // Reseñas Educativas* 0, 0 (2010). https://doi.org/10.14507/er.v0.1296

[18] Taciana Pontual Falcão, Tancicleide C. Simões Gomes, and Isabella Rocha Albuquerque. 2015. Computational Thinking Through Children's Games: An Analysis of Interaction Elements. In *Proceedings of the 14th Brazilian Symposium on Human Factors in Computing Systems (IHC '15)*. ACM, New York, NY, USA, Article 29, 10 pages. https://doi.org/10.1145/3148456.3148485

[19] Bent Flyvbjerg. 2006. Five misunderstandings about case-study research. *Qualitative inquiry* 12, 2 (2006), 219–245.

[20] Gerald Futschek. 2006. Algorithmic Thinking: The Key for Understanding Computer Science. In *Informatics Education – The Bridge between Using and Understanding Computers (Lecture Notes in Computer Science)*. Springer, Berlin, Heidelberg, 159–168. https://doi.org/10.1007/11915355_15

[21] Gerald Futschek and Julia Moschitz. 2011. Learning algorithmic thinking with tangible objects eases transition to computer programming. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer, 155–164.

[22] Alex Games and Luke Kane. 2011. Exploring Adolescent's STEM Learning Through Scaffolded Game Design. In *Proceedings of the 6th International Conference on Foundations of Digital Games (FDG '11)*. ACM, New York, NY, USA, 1–8. https://doi.org/10.1145/2159365.2159366

[23] Paul Gilster. 1997. *Digital Literacy*. John Wiley & Sons, Inc., New York, NY, USA.

[24] Lindsey Ann Gouws, Karen Bradshaw, and Peter Wentworth. 2013. Computational Thinking in Educational Activities: An Evaluation of the Educational Game Light-bot. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13)*. ACM, New York, NY, USA, 10–15. https://doi.org/10.1145/2462476.2466518

[25] Brian E. Gravel, Eli Tucker-Raymond, Kaitlin Kohberger, and Kyle Browne. 2017. Navigating worlds of information: STEM literacy practices of experienced makers. *International Journal of Technology and Design Education* (Sept. 2017), 1–18. https://doi.org/10.1007/s10798-017-9422-3

[26] Shuchi Grover and Roy Pea. 2013. Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher* 42, 1 (2013), 38–43. https://doi.org/10.3102/0013189X12463051 arXiv:https://doi.org/10.3102/0013189X12463051

[27] Casper Harteveld. 2011. *Triadic Game Design: Balancing Reality, Meaning and Play* (1st ed.). Springer Publishing Company, Incorporated.

[28] Amy K. Hoover, Jackie Barnes, Borna Fatehi, Jesús Moreno-León, Gillian Puttick, Eli Tucker-Raymond, and Casper Harteveld. 2016. Assessing Computational Thinking in Students' Game Designs. In *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts (CHI PLAY Companion '16)*. ACM, New York, NY, USA, 173–179. https://doi.org/10.1145/2968120.2987750

[29] Yasmin B. Kafai. 1996. *Constructionism in Practice: Designing, Thinking, and Learning in a Digital World*. Routledge.

[30] Yasmin B. Kafai and Quinn Burke. 2015. Constructionist Gaming: Understanding the Benefits of Making Games for Learning. *Educational Psychologist* 50, 4 (Oct. 2015), 313–334. https://doi.org/10.1080/00461520.2015.1124022

[31] Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. 2007. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of software maintenance and evolution: Research and practice* 19, 2 (2007), 77–131.

[32] Filiz Kalelioğlu. 2015. A new way of teaching programming skills to K-12 students: Code. org. *Computers in Human Behavior* 52 (2015), 200–210.

[33] Filiz Kalelioglu, Yasemin Gilbahar, and Volkan Kukul. 2016. A Framework for Computational Thinking Based on a Systematic Research Review. In *Baltic Journal of Modern Computing*. 583–596.

[34] Filiz Kalelioğlu and Yasemin Gülbahar. 2014. The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective. *Informatics in Education* 13, 1 (2014).

[35] Marjaana Kangas. 2010. Creative and playful learning: Learning through game co-creation and games in a playful learning environment. *Thinking skills and Creativity* 5, 1 (2010), 1–15.

[36] Leonard Kaufman and Peter J Rousseeuw. 2009. *Finding groups in data: an introduction to cluster analysis*. Vol. 344. John Wiley & Sons.

[37] Aaron D. Knochel and Ryan M. Patton. 2015. If Art Education Then Critical Digital Making: Computational Thinking and Creative Code. *Studies in Art Education* 57, 1 (2015), 21–38. https://doi.org/10.1080/00393541.2015.11666280 arXiv:https://doi.org/10.1080/00393541.2015.11666280

[38] Donald E Knuth. 1985. Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly* 92, 3 (1985), 170–181.

[39] Praveen Kuruvada, Daniel Asamoah, Nikunj Dalal, and Subhash Kak. 2010. The Use of Rapid Digital Game Creation to Learn Computational Thinking. *arXiv:1011.4093 [cs]* (Nov. 2010). http://arxiv.org/abs/1011.4093 arXiv: 1011.4093.

[40] James C. Lester, Hiller A. Spires, John L. Nietfeld, James Minogue, Bradford W. Mott, and Eleni V. Lobene. 2014. Designing game-based learning environments for elementary science education: A narrative-centered learning perspective. *Information Sciences* 264 (April 2014), 4–18. https://doi.org/10.1016/j.ins.2013.09.005

[41] Angeline S Lillard. 2013. Playful learning and Montessori education. *NAMTA Journal* 38, 2 (2013), 137–174.

[42] James Lockwood and Aidan Mooney. 2018. Computational Thinking in Secondary Education: Where Does It Fit? A Systematic Literary Review. *Online Submission* 2, 1 (2018), 41–60.

[43] Sze Yee Lye and Joyce Hwee Ling Koh. 2014. Review on Teaching and Learning of Computational Thinking Through Programming. *Comput. Hum. Behav.* 41, C (Dec. 2014), 51–61. https://doi.org/10.1016/j.chb.2014.09.012

[44] K. E. Maani and Vandana Maharaj. 2002. Links Between Systems Thinking and Complex Problem Solving-Further Evidence.

[45] Orni Meerbaum-Salant, Michal Armoni, and Mordechai (Moti) Ben-Ari. 2010. Learning Computer Science Concepts with Scratch. In *Proceedings of the Sixth International Workshop on Computing Education Research (ICER '10)*. ACM, New York, NY, USA, 69–76. https://doi.org/10.1145/1839594.1839607

[46] Punya Mishra and Matthew J Koehler. 2006. Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers college record* 108, 6 (2006), 1017.

[47] Jesús Moreno-León, Gregorio Robles, and Marcos Román-González. 2015. Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia* 46 (2015), 1–23.

[48] Jesús Moreno-León, Gregorio Robles, and Marcos Román-González. 2017. Towards Data-Driven Learning Paths to Develop Computational Thinking with Scratch. *IEEE Transactions on Emerging Topics in Computing* (2017).

[49] Jesús Moreno-León and Gregorio Robles. 2015. Dr. Scratch: A Web Tool to Automatically Evaluate Scratch Projects. In *Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE '15)*. ACM, New York, NY, USA, 132–133. https://doi.org/10.1145/2818314.2818338

[50] Jesús Moreno-León, Gregorio Robles, and Marcos Román-González. 2016. Code to Learn: Where Does It Belong in the K-12 Curriculum? *Journal of Information Technology Education: Research* 15 (2016), 283–303.

[51] Jesús Moreno-León, Gregorio Robles, and Marcos Román-González. 2016. Comparing computational thinking development assessment scores with software complexity metrics. In *2016 IEEE Global Engineering Education Conference (EDUCON)*. 1040–1045. https://doi.org/10.1109/EDUCON.2016.7474681

[52] Jesús Moreno-León, Marcos Román-González, Casper Harteveld, and Gregorio Robles. 2017. On the Automatic Assessment of Computational Thinking Skills: A Comparison with Human Experts. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '17)*. ACM, New York, NY, USA, 2788–2795. https://doi.org/10.1145/3027063.3053216

[53] Chrystalla Mouza, Yi-Cheng Pan, Lori L. Pollock, James Atlas, and Terry Harvey. 2014. Partner4CS: Bringing Computational Thinking to Middle School through Game Design. FabLearn.

[54] Stamatios Papadakis, Michail Kalogiannakis, and Nicholas Zaranis. 2016. Developing Fundamental Programming Concepts and Computational Thinking with ScratchJr in Preschool Education: A Case Study. *Int. J. Mob. Learn. Organ.* 10, 3 (Jan. 2016), 187–202. https://doi.org/10.1504/IJMLO.2016.077867

[55] Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA.

[56] R.T. Pithers and Rebecca Soden. 2000. Critical thinking in education: a review. *Educational Research* 42, 3 (2000), 237–249. https://doi.org/10.1080/001318800440579 arXiv:https://doi.org/10.1080/001318800440579

[57] Marc Prensky. 2003. Digital Game-based Learning. *Comput. Entertain.* 1, 1 (Oct. 2003), 21–21. https://doi.org/10.1145/950566.950596

[58] Gillian Puttick, Jackie Barnes, Giovanni Maria Troiano, Casper Harteveld, Eli Tucker-Raymond, Mike Cassidy, and Gillian Smith. 2018. Exploring how student designers model climate system complexity in computer games. In *Proceedings of the Summit on Connected Learning, CLS'18*. ETC Press, 196–204.

[59] Gillian Puttick and Eli Tucker-Raymond. 2018. Building Systems from Scratch: an Exploratory Study of Students Learning About Climate Change. *Journal of Science Education and Technology* 27, 4 (Aug. 2018), 306–321. https://doi.org/10.1007/s10956-017-9725-x

[60] Mitchel Resnick. 1990. *LEGO/Logo–learning Through and about Design*. Epistemology and Learning Group, MIT Media Laboratory. Google-Books-ID: AEmytgAACAAJ.

[61] Mitchel Resnick. 1996. Distributed Constructionism. In *Proceedings of the 1996 International Conference on Learning Sciences (ICLS '96)*. International Society of the Learning Sciences, Evanston, Illinois, 280–284. http://dl.acm.org/citation.cfm?id=1161135.1161173

[62] Mitchel Resnick. 2003. Playful learning and creative societies. *Education Update* 8, 6 (2003), 1–2.

[63] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and others. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[64] Gregorio Robles, Jesús Moreno-León, Efthimia Aivaloglou, and Felienne Hermans. 2017. Software clones in Scratch projects: On the presence of copy-and-paste in Computational Thinking learning. In *Software Clones (IWSC), 2017 IEEE 11th International Workshop on*. IEEE, 1–7.

[65] Gregorio Robles, Jesús Moreno-León, Efthimia Aivaloglou, and Felienne Hermans. 2017. Software clones in scratch projects: on the presence of copy-and-paste in computational thinking learning. In *2017 IEEE 11th International Workshop on Software Clones (IWSC)*. 1–7. https://doi.org/10.1109/IWSC.2017.7880506

[66] Marcos Romn-Gonzlez, Juan-Carlos Prez-Gonzlez, and Carmen Jimnez-Fernndez. 2017. Which Cognitive Abilities Underlie Computational Thinking? Criterion Validity of the Computational Thinking Test. *Comput. Hum. Behav.* 72, C (July 2017), 678–691. https://doi.org/10.1016/j.chb.2016.08.047

[67] Marcos Román-González. 2015. Computational Thinking Test: Design, Guidelines, and Content Validation. https://doi.org/10.13140/RG.2.1.4203.4329

[68] Marcos Román-González, Juan-Carlos Pérez-González, Jesús Moreno-León, and Gregorio Robles. 2018. Can computational talent be detected? Predictive validity of the Computational Thinking Test. *International Journal of Child-Computer Interaction* (07 2018). https://doi.org/10.1016/j.ijcci.2018.06.004

[69] Ricarose Vallarta Roque. 2007. *OpenBlocks: an extendable framework for graphical block programming systems*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[70] Elizabeth Rowe, Jodi Asbell-Clarke, Kathryn Cunningham, and Santiago Gasca. 2017. Assessing Implicit Computational Thinking in Zoombinis Gameplay: Pizza Pass, Fleens & Bubblewonder Abyss. In *Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play (CHI PLAY '17 Extended Abstracts)*. ACM, New York, NY, USA, 195–200. https://doi.org/10.1145/3130859.3131294

[71] Elizabeth Rowe, Jodi Asbell-Clarke, Santiago Gasca, and Kathryn Cunningham. 2017. Assessing Implicit Computational Thinking in Zoombinis Gameplay. In *Proceedings of the 12th International Conference on the Foundations of Digital Games (FDG '17)*. ACM, New York, NY, USA, Article 45, 4 pages. https://doi.org/10.1145/3102071.3106352

[72] Julio Santisteban and Jennifer Santisteban-Muñoz. 2018. Psychometric Computational Thinking Test. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018)*. ACM, New York, NY, USA, 393–393. https://doi.org/10.1145/3197091.3205823

[73] Florian Scharf, Thomas Winkler, and Michael Herczeg. 2008. Tangicons: Algorithmic Reasoning in a Collaborative Game for Children in Kindergarten and First Class. In *Proceedings of the 7th International Conference on Interaction Design and Children (IDC '08)*. ACM, New York, NY, USA, 242–249. https://doi.org/10.1145/1463689.1463762

[74] Linda Seiter and Brendan Foreman. 2013. Modeling the Learning Progressions of Computational Thinking of Primary Grade Students. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (ICER '13)*. ACM, New York, NY, USA, 59–66. https://doi.org/10.1145/2493394.2493403

[75] Amber Settle, Debra S. Goldberg, and Valerie Barr. 2013. Beyond Computer Science: Computational Thinking Across Disciplines. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13)*. ACM, New York, NY, USA, 311–312. https://doi.org/10.1145/2462476.2462511

[76] Valerie J Shute, Chen Sun, and Jodi Asbell-Clarke. 2017. Demystifying computational thinking. *Educational Research Review* 22 (2017), 142–158.

[77] Sigmund Tobias, J. Dexter Fletcher, and Alexander P. Wind. 2014. Game-Based Learning. In *Handbook of Research on Educational Communications and Technology*. Springer, New York, NY, 485–503. https://doi.org/10.1007/978-1-4614-3185-5_38

[78] Michael Gr. Voskoglou and Sheryl Buckley. 2012. Problem Solving and Computational Thinking in a Learning Environment. *CoRR* abs/1212.0750 (2012). arXiv:1212.0750 http://arxiv.org/abs/1212.0750

[79] David Weintrop, Elham Beheshti, Michael Horn, Kai Orton, Kemi Jona, Laura Trouille, and Uri Wilensky. 2016. Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology* 25, 1 (Feb. 2016), 127–147. https://doi.org/10.1007/s10956-015-9581-5

[80] David Weintrop, Nathan Holbert, Michael S Horn, and Uri Wilensky. 2016. Computational thinking in constructionist video games. *International Journal of Game-Based Learning (IJGBL)* 6, 1 (2016), 1–17.

[81] Linda Werner, Jill Denner, Shannon Campe, and Damon Chizuru Kawamoto. 2012. The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*. ACM, New York, NY, USA, 215–220. https://doi.org/10.1145/2157136.2157200

[82] Amanda Wilson, Thomas Hainey, and Thomas Connolly. 2012. Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In *6th European conference on games-based learning (ECGBL)*. 4–5.

[83] Jeannette M. Wing. 2006. Computational Thinking. *Commun. ACM* 49, 3 (March 2006), 33–35. https://doi.org/10.1145/1118178.1118215

[84] Jeannette M. Wing. 2008. Computational thinking and thinking about computing. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 366, 1881 (Oct. 2008), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

[85] Min Lun Wu and Kari Richards. 2011. Facilitating Computational Thinking Through Game Design. In *Proceedings of the 6th International Conference on E-learning and Games, Edutainment Technologies (Edutainment'11)*. Springer-Verlag, Berlin, Heidelberg, 220–227. http://dl.acm.org/citation.cfm?id=2040452.2040499