

RESEARCH

Free and Open Access

Integrating Parsons Puzzles within Scratch Enables Efficient Computational Thinking Learning

Jeff Bender *, Bingpu Zhao, Alex Dziena and Gail Kaiser

*Correspondence:
jeffrey.bender@columbia.edu
Department of Computer
Science, Columbia University,
1214 Amsterdam Avenue, New
York, NY, 10027, United States
Full list of author information is
available at the end of the article

Abstract

A literature review revealed that students learning computational thinking via Scratch often require substantial teacher support. We surveyed grade 6-9 teachers to learn their perceptions of student engagement with computational thinking (CT) and how well their needs are met by existing CT learning systems. The results led us to extend the trend of balancing Scratch's agency with structure to better serve learners and reduce burden on teachers aiming to learn and teach CT. In this paper, we review architecture and implementation strategies developed to integrate Parsons Programming Puzzles (PPPs) with Scratch, and then analyze their effects on adults, who crucially influence the education of their children. The results from our pilot study suggest PPPs catalyze CT motivation, reduce extraneous cognitive load, and increase learning efficiency without jeopardizing performance on transfer tasks.

Keywords: Computational Thinking, Parsons Programming Puzzles, Scratch, Motivation, Cognitive Load

Introduction

In response to a crisis in CS teacher certification and a deficit of student exposure to CS in grades K-12 (Wilson et al., 2010; Leyzberg et al., 2017), governments are enacting policies requiring CT in schools (Whitehouse.gov, 2016; The Royal Society, 2016; Tamatea, 2019). Additional argument (Wing, 2006, 2008) and evidence (Grover et al., 2013) provide rationale for ensuring children achieve CT competency during the formative cognitive and social development cycles throughout grades K-12. Parsons programming puzzles (PPPs), which enable learners to practice CT by assembling into correct order sets of mixed-up programming constructs that comprise samples of well-written code focused on individual



© The Author(s). 202X **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

concepts, are one approach used to introduce CT efficiently (Parsons et al., 2006; Ericson et al., 2018).

These scaffolded program construction tasks facilitate learning the syntax and semantics underlying a CT concept. As the learner solves carefully designed single-solution puzzles, she arranges constructs from a curated assortment in a cycle of deliberate practice that exposes and addresses misconceptions (Kaczmarczyk et al., 2010; Emerson et al., 2020). Among the correct code fragments, she might find distractors which provoke cognitive conflict that reinforces learning (Karavirta et al., 2012). The partial suboptimal path distractor type, for example, might tempt a learner toward faulty progress without enabling her to solve the problem fully, thereby triggering recognition of a misconception and productive backtracking toward the correct solution (Harms et al., 2016). Researchers have hypothesized distractors might be beneficial in PPP for reasons similar to those leading to their inclusion in multiple choice tests (Parsons & Haden, 2006; King, 2004), such as the illumination of conceptual misunderstanding, flawed reasoning, or inconsistent reasoning.

Research indicates this structured approach to learning CT can lead to more efficient concept learning than alternatives such as learning by tutorial, or writing/fixing code (Harms et al., 2015; Ericson et al., 2017; Zhi et al., 2019). To measure efficiency, researchers often leverage cognitive load theory, which helps to distinguish between the complexity of the material, the instructional design, and the strategies used for knowledge construction. Since PPPs provide constrained problem spaces, they can induce lower cognitive load than that experienced when writing code with open-ended agency in a realm notoriously challenging for novices (Fabic et al., 2018).

In the current study¹, we seek further evidence of their efficiency by integrating PPPs into Scratch, a block-based environment initially designed for informal learning that invites exploration, collaboration, and knowledge construction through personally meaningful creation (Maloney et al., 2010). K-8 teachers use Scratch more than any other coding language internationally (Rich et al., 2019), resulting in an ecosystem with over 92 million registered users (MIT Media Lab, n.d.), and more research focus than any other environment in K-12 from 2012-2018 (McGill & Decker, 2020). However, historical findings indicate Scratchers infrequently demonstrate skill increases over time (Scaffidi et al., 2012), misconceive loops, variables, Booleans, nested conditionals, and procedures (Grover et al., 2017, 2018), and often adopt habits unaligned with accepted CS practice (Meerbaum-Salant et al., 2011). In a recent study of 74,830 Scratch projects, 45% contained at least one bug pattern (Frädriich et al., 2020). Instead of problem-solving algorithmically, Scratchers often engage in bricolage (Harel & Papert, 1991), which involves bottom-up tinkering that does not necessarily prove productive (Dong et al., 2019).

To balance this agency with structure as recommended in (Brennan, 2013), and to encourage the development of desired habits when learning CT concepts without stifling

learner creativity, researchers have designed external Scratch curricula (Brennan et al., 2014; Franklin et al., 2020), created introductory Scratch Microworlds with reduced functionality (Tsur & Rusk, 2018), and devised learning strategies based on the Use->Modify->Create pedagogy to scaffold instruction (Salac et al., 2020). We extend this trend by integrating with Scratch PPPs with explicit goals that offer gameful scoring targets and per-block feedback to disincentivize trial-and-error behavior and steer learners toward correct solutions. We reason that if learners initially can internalize CT concepts efficiently via PPPs, they can better deepen their understanding with heightened ownership and agency (Casanova et al., 2021) in less-restrictive interest-driven projects such as those described in (Kong et al., 2020) that embrace Scratch's roots in constructionism (Brennan et al., 2014). This strategy would enable the development of learning progressions through the cognitive, situated, and critical framings of CT documented in (Kafai et al., 2019), so that skill building leads to creative expression and participation in fostering equitable, ethical computing for all.

To test this reasoning, we ran a pilot study targeting adults, who comprise a general population that might not only benefit from learning CT, but who might most effectively mobilize the advancement of teaching and learning CT for all. The study separates participants into one of three training conditions: 1) PPPs; 2) PPPs with distractors (PPPDs); 3) programming with access to all blocks and without feedback (limited-constraint-feedback or LCF). Each successive condition offers the learner increasing agency by offering more block options from which to construct puzzle solutions. Condition 3, with block-move correctness feedback and scoring removed, most closely resembles the code writing experience native to Scratch. We investigated the following research questions:

R1) what are the effects on motivation and cognitive load when training occurs via: PPPs; PPPDs; LCF?;

R2) what are the effects on learning efficiency for training via PPP, PPPD, and LCF?

Although the 75-participant sample limits the number of statistically significant results, findings indicate:

F1) participants self-report higher motivation when training via PPPs and PPPDs, and less extraneous cognitive load when training via PPPs than via PPPDs or via LCF;

F2) participants training via PPPs and PPPDs experience increased learning efficiency compared with those training via LCF.

We first review the background and required software development. We then document the study purpose, formative and summative evaluations, and results before previewing future work.

Background

Since PPPs emerged in the CS literature as a new form of program completion problem in 2006, the community has investigated their strengths and weaknesses. Strengths include: scaffolded support of syntax and semantics learning; solvers with prior experience perform better and need less time (Harms et al., 2016); quicker grading and less grading variability than code writing problems (Ericson et al., 2017); easier detection of learning differences between students compared to code writing and code fixing problems (Morrison et al., 2016); a moderate correlation between PPP proficiency and code writing proficiency in an exam setting (Denny et al., 2008); less completion time required than for code writing exercises with equivalent performance on transfer tasks (Ericson et al., 2017; Zhi et al., 2019); higher enjoyment and less completion time required than for tutorial users with better performance on transfer tasks (Harms et al., 2016); and a lack of significant differences in performance across gender. Weaknesses include: constriction of puzzle-design surface to maintain single-solution structure (not strictly required, but commonly enforced to maintain strengths); the invitation of trial-and-error behavior in PPPs with excessive corrective feedback (Helminen et al., 2013); and a potential ceiling effect when feedback guides most learners to solve PPPs correctly, resulting in the need to evaluate learner process in addition to product when assessing (Helminen et al., 2012; Villamor et al., 2020).

The community also has explored differences in learning outcomes resulting from using different PPP elements. Evidence suggests that 2D puzzles, in which the student must not only correctly order programming constructs but also indent them correctly, are more difficult than 1D (Ihantola & Karavirta, 2011). Similarly, PPPs that conceal the number of lines of code needed for each solution section and those that include distractors are more difficult, require more time to complete, and produce higher cognitive load during training than those that specify section sizes and those without distractors (Garner, 2007; Harms et al., 2015). Learning differences continue to emerge when researchers vary these elements. For example, learners struggle more when distractors are randomly distributed among the correct code constructs than when they are paired with correct constructs (Denny et al., 2008).

To identify these strengths, weaknesses, and learning differences between PPP elements, researchers often leverage Cognitive Load Theory (CLT) (Sweller, 2010). According to CLT, the brain provides limited short-term memory and processing capability along with infinite long-term memory, and learning occurs via schema construction and elaboration

that leads to automation. Construction ensues by combining new, single elements into one larger element, and elaboration follows by adding new elements to an existing, larger element. Through intensive practice, individuals can automate their processing of these larger elements so that they execute without controlled processing.

CLT helps distinguish characteristics of and between PPP systems by offering a framework with tools to measure the three types of cognitive load experienced: intrinsic, extraneous, and germane. The total number of interacting elements perceived by the learner determines intrinsic load (IL); the sometimes-impeding organization and presentation of the content determines extraneous load (EL); and the instructional features necessary for schema construction determine the germane load (GL). PPP designers aim to reduce extraneous load to free learners' capacity to contend with germane load when attempting to maximize learning efficiency. For example, the pairing of distractors with correct constructs might increase germane load by focusing student attention on the intended, misconception-revealing differences between two solution options, while also reducing extraneous load by eliminating the need to search for and identify the two relevant options amidst a random distribution of constructs.

To measure relative learning efficiency quantitatively across conditions, researchers calculate instructional and performance efficiency (van Gog & Paas, 2008). These calculations account for learners who compensate for an increase in mental load by committing more mental effort, thereby maintaining constant performance while load varies. The data recorded often include empirical estimates of mental effort during instruction (EI) and transfer (ET) tasks and the performance (P) on transfer tasks. The EI and P calculation measures the instructional efficiency of the learning process, while the ET and P calculation measures the performance efficiency of the learning outcome. For example, in a study that included interactive puzzles in the transfer phase, results indicate PPPs with randomly distributed distractors decrease performance efficiency (Harms et al., 2016). In our study, we measure instructional efficiency with a focus on learning process economy.

Software Development

To investigate our research questions, we modified Scratch to facilitate the design, play, and assessment of PPPs. Aligned with the gamification strategy described in (Tahir, Mitrovic, & Sotardi, 2020), in which the game elements were added to SQL-Tutor, and similar to recent iSnap integrations offering progress panels and adaptive messages (Zhi et al., 2019; Marwan et al., 2020), we augmented Scratch to influence the behavior of learners. As shown in Figure 1, we first established a design mode which enables content developers to assign points to individual blocks and select blocks for inclusion in a new PPP palette.

Equipped with this functionality, teachers can assign higher point values to blocks relevant to the CT concept studied and can isolate in a single palette blocks pertinent to the puzzle.

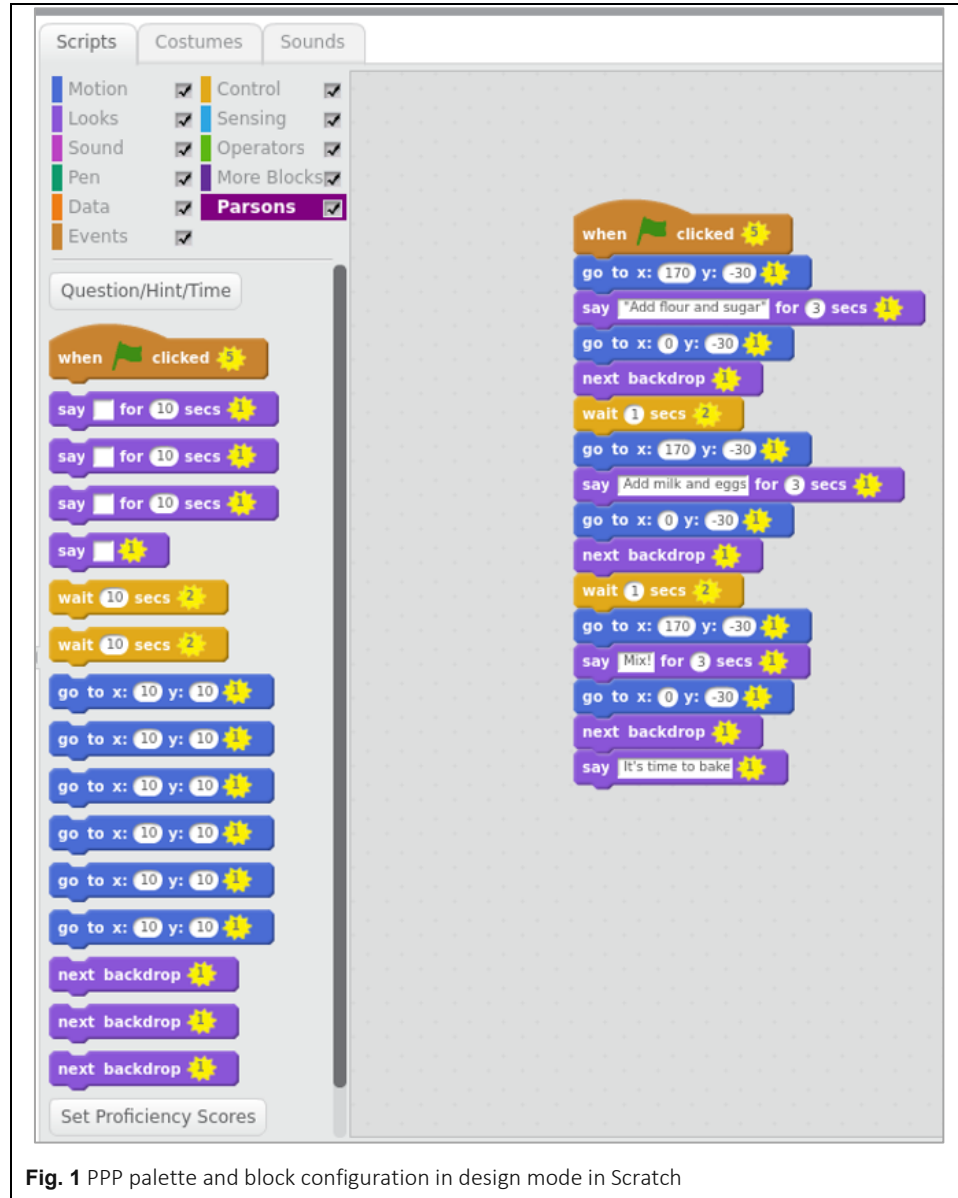


Fig. 1 PPP palette and block configuration in design mode in Scratch

As presented in Figure 2, we next established a play mode which enables students to load PPPs in a manner that displays the designed animated elements in the Scratch stage, but none of the blocks in the scripts pane authored as the solution. Technical detail is reported in (Sulaiman, et al., 2019), but relevant to this study is an assessment system that includes a gameful scoring algorithm intended to encourage deliberate practice and discourage trial-and-error behavior. Our early development attempts involved calculating the Manhattan distance of each block placed from its correct position and multiplying that by the points

assigned to each block and the length of the sequence, combined with subtractions for special cases such as multiple disconnected sequences of blocks in the learner solution. During testing, however, this strategy proved insufficient, as scores could confusingly decrease when a block placed incorrectly in a long sequence was moved to the correct place in a shorter sequence. The longest common subsequence feedback algorithm described in (Karavirta et al., 2012) ultimately inspired our final approach; ours differs in that we leverage block points, use them and subsequence length as multipliers, and sum the multiples from all subsequences matching the single correct solution while also deducting for incorrectness in absolute position. The closer the participant is to the solution, the higher the score.

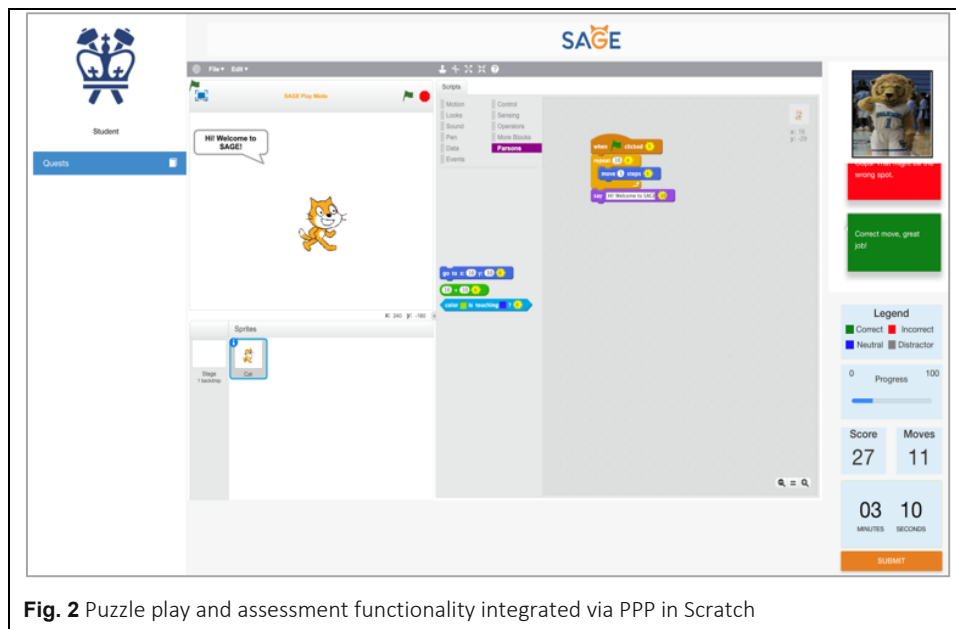


Fig. 2 Puzzle play and assessment functionality integrated via PPP in Scratch

To reduce complexity in the scoring algorithm but still discourage trial-and-error behavior, we simultaneously track a count we name *meaningful moves*, which increments when the learner drags a block from the palette to the scripts pane, connects existing sequences together, or disconnects a sequence into two. Other less significant block actions, such as the repositioning of a block or sequence within the scripts pane are discarded. Since we display this count (11) next to the score (27) and remaining time (3m 10s) as shown in Figure 2, we can encourage learners to achieve the highest score in the fewest moves and shortest time.

Additionally, we built auto-initialization and auto-execution functionality to reflect progress visually after each block placement during puzzle play. These mechanisms enable the display of gameful animations while an avatar presents per-block correctness feedback, while concurrently disabling Scratch features that might otherwise distract from CT

learning, such as sprite editing controls. According to the feedback classification in (Raubenheimer, 2021), this immediate correct-incorrect-distractor feedback is constructivist since it is problem- and instance-oriented, which has been correlated with significantly lower student failure rates than alternative types such as those solution- and instance-oriented. The auto-execution functionality also calculates completion progress so that the learner receives appropriate responses when she correctly solves the puzzle or the allotted time expires.

To help teachers and content developers organize learning, we wrapped these new features in Scratch within custom-built learning management tooling that facilitates the ingestion of class rosters, the structuring of learning paths in which games comprise quests which comprise missions, and the saving and loading of game/quest/mission progress. We intend this playful framing to further gamify the learning experience and nudge learners toward increased motivation as described in (Bovermann & Bastiaens, 2020). For the motivation to sustain beyond this learning experience, however, further progress in standardizing interoperability protocols between learning systems is necessary throughout the CS education community, similar to the one proposed in (Brusilovsky et al., 2018). The reference architecture we present in (Sulaiman, et al., 2019) is intended as one small step forward in that direction.

Study Purpose

This extended functionality positioned us to fill gaps in existing research. One study purpose was to explore the adult-use of CT learning system functionality primarily designed for children. Recent research has: 1) found significant correlation of motivation and previous programming experience with self-efficacy and inclination toward a CS career in elementary students (Aivaloglou & Hermans, 2019); 2) indicated drag-and-drop programming can increase three CS motivational factors in middle school (Bush et al., 2020); 3) suggested computing experiences prior to university can affect the world-image of computing habits, perceptions, and attitudes which enable or inhibit pathways into CS (Schulte et al., 2007); 4) identified a parental role framework to enable adults to choose productive strategies to promote and foster children's CT (Ohland et al., 2019); and 5) illuminated benefits of community commitment and a CS/CT focused ecosystem inclusive of the home and community (Cao et al., 2020; DeLyser, 2018). Since demographic factors can drive communal values, and perceptions of how computing fulfills those values can affect sense of belonging and student retention (Lewis et al., 2019), we measure adult motivation and cognitive load while probing for attitudinal change that might influence the CT inclination for participants' children.

A second purpose was to further identify PPP elements that optimize learning efficiency, since the behavior of programming environments can affect novices' learning (Karvelas &

Becker, 2020). While many researchers have hypothesized (Denny et al., 2008) and less often produced evidence (Ericson et al., 2018) that PPPs can result in more efficient learning than alternatives such as writing or fixing code, recently some have attempted to measure the contributions of various PPP elements (Kumar, 2017, 2019a, 2019b; Sirkia, 2016), including the effect of displaying the number of lines of code in puzzle solutions (no effect on pre-post improvement, more time spent), of pairing distractors with related variants (effective in the longer of two studied puzzles), of using single-character or mnemonic variable names (no significant differences), and of presenting program visualizations alongside PPPs (visualizations were used most by novices who sought feedback the most via multiple submissions). We measure PPP learning efficiency with and without distractors, while offering a comparison to programming with LCF. Derived from the literature, our hypotheses were:

H1) PPP and PPPD training increase motivation and reduce extraneous cognitive load compared to training via programming with LCF;

H2) PPP training yields highest learning efficiency.

Formative Evaluation

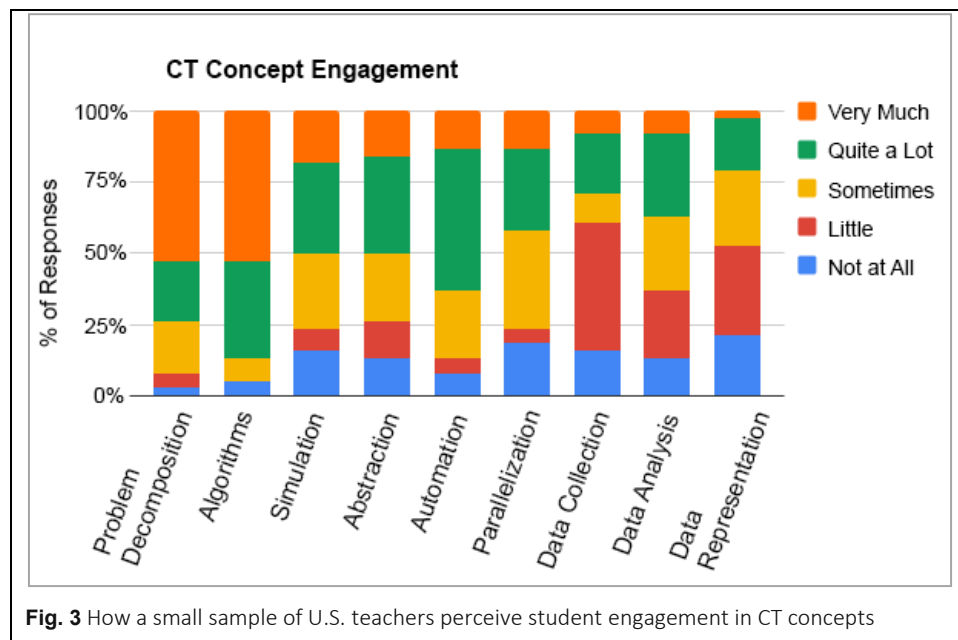
As an early step in a roadmap of studies intended to explore the efficacy of adding intelligent and gameful systems to novice programming environments, and with an aim to reinforce construct validity, we engaged in a formative evaluation with grade 6-9 educators. Through design thinking activities (Razzouk & Shute, 2012) including iterative surveys, interviews, and prototyping, we advanced our learning design technique, similar to the approach described in (Kashmira & Mason, 2020), which illuminates design thinking as a strategy useful for exploration, managing uncertainty, learning from failures, and empathizing with the needs of the learner when connecting learning objectives to learning design. Our goals included: 1) identifying the CT concepts receiving focus; 2) eliciting the pedagogical needs of practicing teachers; 3) and refining puzzle and feedback systems. We focus discussion here on goal 1.

Participants

The participants included 21 teachers from learning organizations such as Girls Who Code (Girls Who Code, n.d.) and codeHER (CodeHER, n.d.), and 17 from U.S. schools. 11% had taught with Scratch for at least 2-4 years, 63% for 6-18 months, and 26% had instructed with Scratch for less than 6 months. 92% taught CS with Scratch, but 16% also or alternatively taught math, and 16% taught science, language arts, or applied arts with Scratch. 34% of the teachers used Scratch for at least 51% of their curriculum, 29% used it for 26-50%, and 29% used Scratch for at least 11-15%.

CT Concept Engagement

(Ihantola et al., 2016) highlights the concerning status quo in which most studies in the field focus on a single institution and a single course, without validation by subsequent replicating research, leading to limited understanding of the reasons results occur. To contribute replication results, and to identify the CT concepts receiving focus, we distributed a survey that included a question from a survey previously distributed to K-9 teachers in five European countries (Mannila et al., 2014). This question asks teachers to respond with their perceptions of student engagement in nine facets of CT. Since we targeted a narrower set of teachers in the U.S., it is perhaps unsurprising that the results do not match the earlier international study, in which teachers reported their students most frequently use CT concepts related to data (e.g. analysis). However, we present this finding to reinforce the replication concerns raised, and to underscore the challenges the community faces when attempting to disseminate CT globally.



Our findings in Figure 3 indicate teachers perceive their students engage in data CT concepts less than others such as abstraction and algorithms. Aside from the differences in population samples, and the associated threat to internal validity due to implicit differences in curricula ((Barendsen et al., 2015) notes a low ratio of data knowledge in K-9 U.S. CSTA materials, 2%, compared with the English national curriculum, 14%, English Computing at School, 16%, and Italian guidelines, 25%), an extra explanation for this contrast could be related to the respondent recruitment process, as we specifically targeted Scratch teachers, whereas the earlier study did not. Since the small sample introduces a threat to external validity, future studies could try to replicate these results while controlling for

technology and teacher pedagogical content knowledge (PCK) utilizing a Content Representation approach like the one described in (Grgurina et al., 2014), in which researchers elicited via interview then charted teachers PCK across eight categories. Regardless, the lack of student engagement with data warrants investigation, as it is an alarming result for an increasingly data-driven society.

Summative Evaluation

Study Design

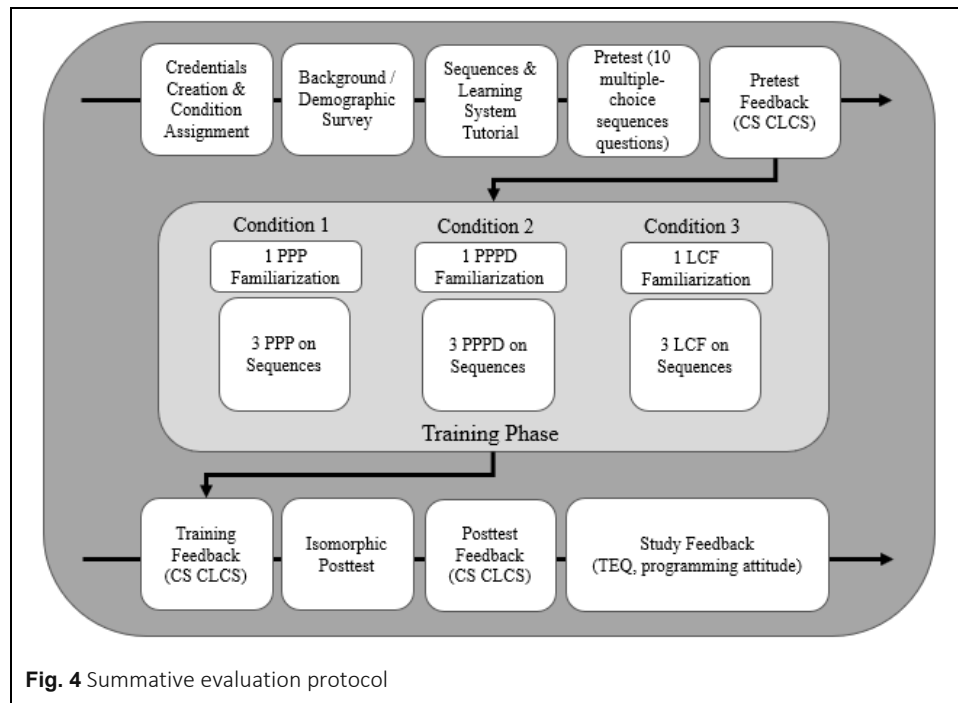


Fig. 4 Summative evaluation protocol

The formative evaluation helped us roadmap implementations, craft learning materials, and plan an initial summative evaluation. To produce evidence supporting answers to **R1-2**, we organized a 10-step between-subjects study via Amazon Mechanical Turk (Amazon Mechanical Turk, 2021) with the CT concept *sequences* operating as the learning objective. As depicted in Figure 4 and detailed in Table 1, the steps involved: 1) creation of credentials in the learning system and assignment to 1 of 3 conditions characterized in Table 2; 2) a background survey; 3) review of a 6-minute video tutorial on the UI and CT concept *sequences* delivered via Panopto (Panopto, n.d.); 4) a pretest; 5) pretest feedback; 6) familiarization and training varied by condition; 7) training feedback; 8) an isomorphic posttest; 9) posttest feedback; 10) study feedback. These steps required responses to the validated CS cognitive load component survey (CS CLCS) (Morrison, et al., 2014), to a programming attitude Likert scale survey derived from categorized text-based responses

by adult learners in (Charters, et al., 2014), and to the intrinsic motivation Task Evaluation Questionnaire (TEQ) (SDT), which is a validated 22-item Likert scale measurement designed to reflect participant experience on four subscales: interest/enjoyment, perceived competence, perceived choice, and pressure/tension. In step 6, participants followed written instructions to guide their solving of four puzzles; instructions for the first puzzle included a graphical representation of the correct solution and an explanation of the behavior of each block used for the purpose of familiarization. Each puzzle auto-submitted upon correct completion or after 500 seconds if the participant had not previously submitted an incorrect solution. We advised participants to complete steps 4, 6, and 8 without interruption and required completion of all steps within two hours. Protocol materials are publicly available in (Integrating Parsons Puzzles with Scratch, 2021).

Table 1 Study protocol & measurements

#	Activity	Content	Tools	Data Collected
1	Registration	Credentials creation & condition assignment	Custom Scratch extension	Username & password
2	Background info	Demographics	Qualtrics	Age, gender, education, country, programming experience & attitude, CT perceptions
3	Tutorial	6-minute video on the learning system & sequences	Panopto	N/A
4	Pretest (isomorphic)	10 multiple-choice questions	Qualtrics	Pretest responses & score
5	CS CLCS	10 CL questions with 0-10 scaled responses	Qualtrics	Pretest CL & IL/EL/GL components
6	Puzzles	4 puzzles on sequences	Custom Scratch extension	Per-puzzle time spent, time-stamped block moves and score, correctness, distractor usage, generated feedback
7	CS CLCS	10 CL questions with 0-10 scaled responses	Qualtrics	Puzzle CL & IL/EL/GL components
8	Posttest (isomorphic)	10 multiple-choice questions	Qualtrics	Posttest responses & score

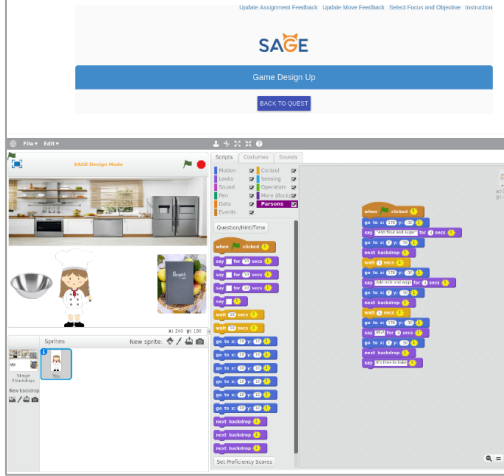
9	CS CLCS	10 CL questions with 0-10 scaled responses	Qualtrics	Posttest CL & IL/EL/GL components
10	Concluding measurements	Motivation, programming attitude, CT perceptions	Qualtrics	TEQ score, programming attitude, CT perceptions

We randomly assigned participants to one of three conditions operating as the independent variable: 1) PPP training (PPP); 2) PPP with distractors training (PPPD); 3) training by solving puzzles with access to all blocks and without move correctness or score feedback (LCF). The dependent variables included time spent and performance on the pretests and posttests, time spent and block moves made in puzzles, and the cognitive load, programming attitude, and TEQ results.

Table 2 Training and participant characteristics across three study conditions

Cond.	Presentation	# Distractors	Feedback	# Participants	Avg. Prog. Exp. 0-10	Gender	Country
PPP	1-palette	0	Correctness	31	3.3	65%M 35%F	PPP
PPPD	1-palette	2-4	Correctness & distractor notification	22	4.7	50%M 50%F	PPPD
LCF	All palettes	All Scratch blocks	None	22	3.7	68%M 32%F	LCF

Study Design



You tried a slice of your pie and realized that you forgot to add sugar! In this puzzle, you will consult a recipe book to bake another pie for your friends.

1. First, you go to the recipe book at location x:-170, y:-30.
2. Second, you say one step of the recipe instruction (seen below) for 3 seconds (just to make sure you get it right this time!).
3. Third, you go to the mixing bowl to complete the task at location x:0, y:-30.
4. Fourth, you switch to next backdrop to reflect changes in the mixing bowl.
5. Then, make sure to wait for a second to see the changes. However, note that this only happens for the first two times.
6. There are three steps in the recipe book, so you go through the sequence above three times. The recipe is as follows:
 1. Step1: Add flour and sugar.
 2. Step2: Add eggs and milk
 3. Step3: Mix
7. After you mix, you say "It's time to bake".

Number of blocks required to solve this puzzle: 16

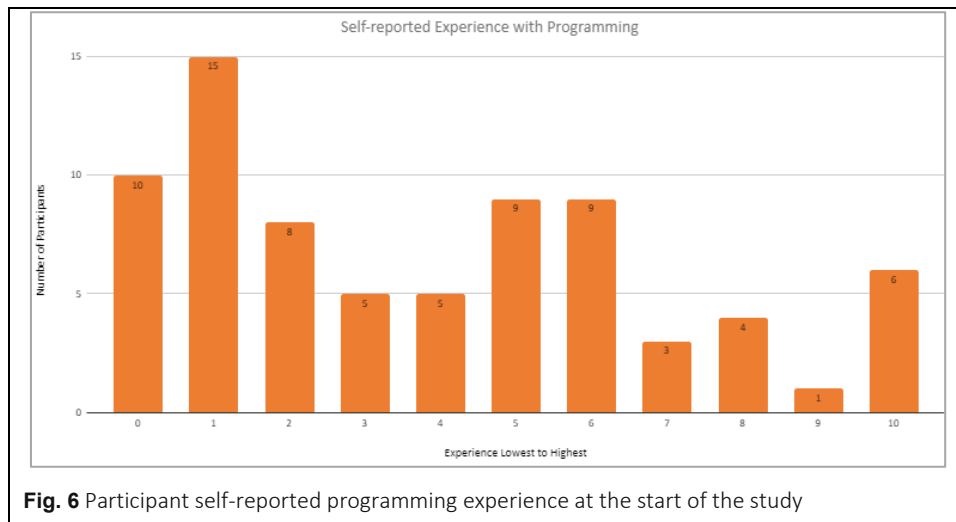
Fig. 5 Example sequences PPP solution and instructions

Following guidance in (Harms et al., 2015), we aimed to design motivating scenarios with memorable segments while providing a challenge without being tricky and leaving the participants with a positive impression. To familiarize them, we included in the instructions for the first puzzle the solution and block-use descriptions. We also included more detailed instructions than typically found in PPPs, effectively resulting in a hybridization of the tutorial and PPP approaches described in (Harms et al., 2016). We thought this approach might best minimize ambiguity and highly scaffold early learning of new CT concepts in the absence of an instructor. An example puzzle solution and the associated instructions are shown in Figure 5.

We tested and refined our materials in collaboration with a high school teacher, 16 of her freshman physics students with little prior exposure to CT, and eight undergraduates with diverse majors. Tests included trials of the surveys and puzzles, and think-alouds in which the participant would interact with puzzles while verbalizing her thoughts. Although we did not further formally assess validity and reliability, these results led to refinements such as puzzle theme modification, normalization of pre/posttest difficulty, and simplification of language used in survey questions.

Participants

In alignment with Wing's mobilizing declaration that CT is a "fundamental skill for everyone, not just for computer scientists" (Wing, 2006), and with the interventionist spirit of design-based research (Barab, 2014), we sought a learner population inclusive of those who might not otherwise encounter an opportunity to engage purposefully with CT but regardless might influence its trajectory in the lives of children. The learning objective of the CT concept *sequences* is suitable for this largely novice set of participants, as teachers often present this concept first in a CT curriculum (e.g. Brennan et al., 2014). By presenting our study as a Human Intelligence Task on Amazon Mechanical Turk, we recruited from a general population of over 100K individuals (Difallaha, et al., 2018) 75 adults with varying educational experience (24% graduated high school, 60% earned an undergraduate degree, 16% earned a graduate degree) and the variety of self-reported programming experience presented in Figure 6. 46 men and 29 women comprise the sample population sourced from eight countries including the U.S. (60%), India (20%), and Brazil (11%). As presented in Table 2, the backgrounds and self-reported programming experience of participants across conditions are largely homogenous, with slightly higher average programming experience on a 0-10 scale reported for the PPPD condition (4.7) than LCF (3.7) and PPP (3.3), higher female participation in the PPPD condition (50%) than in PPP (35%) and LCF (32%), and lower U.S. representation in the LCF condition (50%), than in PPP (65%) and PPPD (67%). Additional participant demographic detail and all summative evaluation data are available in (Integrating Parsons Puzzles with Scratch., 2021).



Analysis & Results

Data Collection & Processing

We created seven surveys in Qualtrics (Qualtrics, n.d.) to capture data not directly collected by our CT learning system. For the pretest and posttest, we recorded time elapsed, and to grade the multiple-choice responses, we wrote a Python script. To help measure performance and efficiency, we added instrumentation to: 1) record time from puzzle start until submission; 2) trace each block moved; and 3) calculate score via the algorithm dependent on block position and points described in section “Software Development”. In the following subsections we analyze the collected and processed data. Since they did not exhibit Shapiro-Wilk normality ($p < 0.05$), we used non-parametric statistics, including Kruskal-Wallis H, Mann-Whitney U, and Spearman r tests between-subjects, and Wilcoxon tests within-subjects, to address skewness and kurtosis. Due to sample-size limitations, we report both significant findings and those non-significant that appear to have the highest potential to reach significance in post-pilot studies with hundreds of participants.

Cognitive Load

We did not find significant differences in overall cognitive load during training between conditions ($H(2) = .506$, $p = .776$), nor in the subtypes. Upon closer review, we found no notable differences in intrinsic and germane load, but moderate non-significant differences in extraneous load (PPP: $M = 3.12$; PPPD: $M = 3.55$; LCF: $M = 3.90$). This result signals weak support for **H1**, as PPP participants self-reported lower extraneous load than PPPD participants, while LCF participants reported the highest. Since the LCF condition

presented far more block choices (548 across 4 puzzles) than the PPPD condition (55), which in turn presented more choices than the PPP condition (41, rank correlation $r(1)=1.0$, $p<.01$), this result indicates reducing impediments to block identification frees capacity for intrinsic and germane load. The higher extraneous cognitive load for training via PPPDs than with PPPs aligns with the findings in (Harms et al., 2016), which reported no significant difference for intrinsic and germane load, but one for extraneous load with the highest mean in the distractor condition, in a study comparing PPPs with and without distractors. Pedagogically, distractors present an opportunity for the instructor and/or learning system to intentionally challenge the learner to address potential misconceptions. We recommend further study to track cognitive load as agency increases, since misconceptions not addressed during structured learning could amplify in open-ended environments, resulting in higher cognitive load if measured in sum across a longitudinal span. Incremental addition of blocks options, and the associated incremental EL, could help learners prepare for future learning as they advance.

Performance

Though we did not find significant training performance differences across conditions ($H(2)=.853$, $p=.653$), participants in the PPP and PPPD conditions interacted with the blocks significantly more ($H(2)=21.141$, $p<0.001$, $\epsilon^2=0.29$). Using a Bonferroni-adjusted alpha of .017 (.05/3), we found significant differences between conditions PPP ($M=52.2$) and LCF ($M=32.1$), $p=0.001$, and PPPD ($M=57.9$) and LCF, $p<0.001$. The fewer block moves made by participants in the LCF condition indicates some may have perceived the task as sufficiently overwhelming to decrease the probability of exploratory programming behavior.

Although participants in each condition solved more posttest than pretest questions correctly (PPP: $M=0.65$, PPPD: $M=0.82$, LCF: $M=0.32$), with those in the PPPD condition yielding the highest increase, there is no significant difference in performance gain across conditions ($H(2)=1.335$, $p=0.513$). This lack of transfer performance disparity between PPP and PPPD conditions ostensibly replicates findings in (Harms et al., 2016), which found no significant difference in performance on transfer tasks for those training via PPPs and PPPDs. It is also similar to findings on PPP inter-problem and intra-problem adaptation in (Ericson et al., 2018), in which no significant differences in learning gains occurred from pretest to immediate posttest across three conditions involving PPPDs and one involving code writing, which is similar to the LCF condition in our study.

Efficiency

To measure efficiency, we analyzed training and transfer task time across conditions. During training, participants in the LCF condition, despite making fewer block moves,

required significantly more time than those in the PPP and PPPD conditions ($H(2)=6.203$, $p=0.045$, $\epsilon^2=0.08$). After the Bonferroni adjustment, significance moderated slightly: PPP ($M=9.3m$) vs. LCF ($M=11.3m$): $p=0.090$; PPPD ($M=9.6m$) vs. LCF: $p=0.063$). Since transfer task performance did not vary significantly across conditions, this result suggests training via PPPs and PPPDs enables more efficient CT learning, per the EI and P instructional efficiency calculation introduced earlier. We did not, however, find a significant difference in the transfer task time ($H(2)=0.883$, $p=0.643$).

To emphasize the opportunity for efficient CT learning, we calculated instructional efficiency, using pre/posttest improvement to measure transfer performance and both time and cognitive load as measurements of mental effort during training, as recommend in (Paas & Merrienboer, 1993). Figure 7 presents areas of high and low effectiveness separated by the effort line $E=0$. The chart depicts higher instructional efficiency for training with PPPs and PPPDs than with LCF. However, this result does not support **H2**, as the PPPD condition yielded the highest instructional efficiency. This result contrasts with findings in (Harms et al., 2016), which found evidence of decreased learning efficiency from PPPDs when compared to PPPs, but it aligns with hypotheses regarding distractor learning benefits in (Parsons et al., 2006; Karavirta et al., 2012) that propose distractors can facilitate the highlighting of both subtle and complex principles as well as edge cases and common misconceptions.

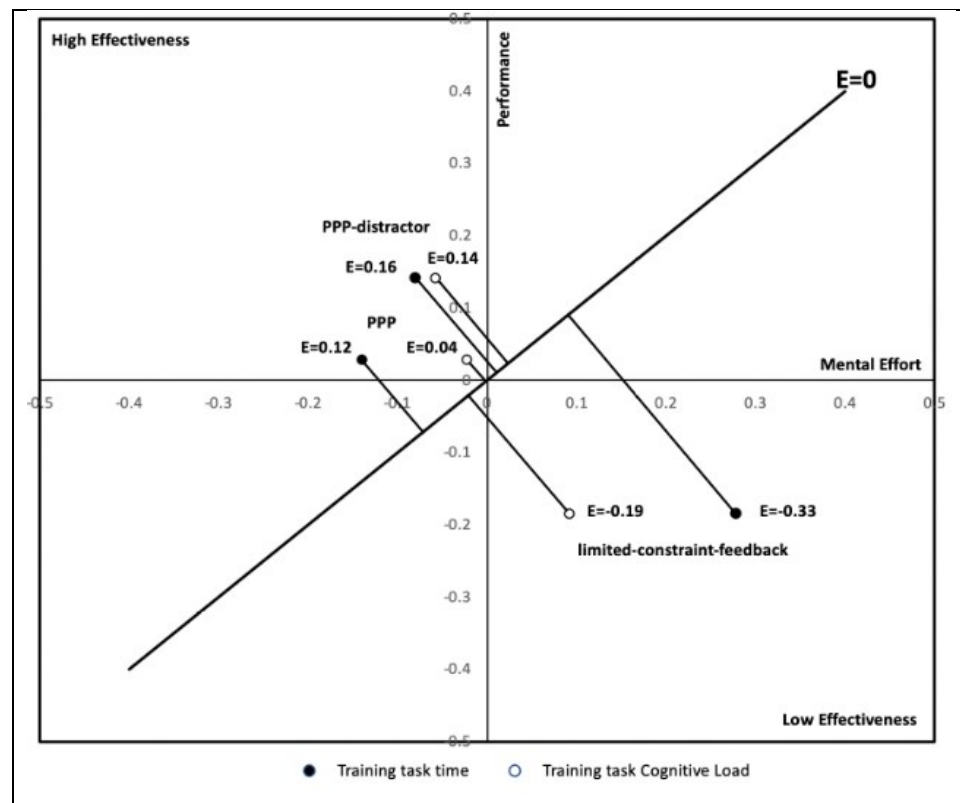


Fig. 7 Instructional efficiency (E) for each of the three conditions

Motivation

To analyze motivation quantitatively, we scored the TEQ and calculated the within-subject change in programming attitude that occurred between the start and end of the study. Although there was no significant difference in TEQ results across conditions, for the perceived competence subscale ($H(2)=.156$, $p=.925$), participants who trained with PPPs ($M=4.89$) and PPPDs ($M=4.91$) scored marginally higher than those who trained with LCF ($M=4.53$). Those who trained via LCF ($M=4.06$) reported marginally higher pressure/tension ($H(2)=2.626$, $p=.269$) than those training via PPPs (3.42) and PPPDs (3.45), which suggests that a smaller block option set and correctness feedback after every move can decrease the experience of pressure when solving a timed parsons puzzle.

We also found significant positive attitude changes from the start to end of the study. PPP participants' attitude shifted most by: *perceiving programming as more fun* ($z=2.392$, $p=0.017$, $r=0.07$), *more enjoyable* ($z=2.428$, $p=0.015$, $r=0.28$), *easier to start* ($z=3.038$, $p=0.002$, $r=0.55$), *less difficult to understand* ($z=-3.343$, $p=0.01$, $r=-0.6$), and *less of a foreign concept* ($z=-3.074$, $p=0.002$, $r=0.55$). PPPD participants' attitudes also shifted positively by: *perceiving programming as easier to start* ($z=2.514$, $p=0.012$, $r=0.54$). LCF participants shifted least by: *perceive programming as more enjoyable* ($z=2.514$, $p=0.012$, $r=0.46$). When including only those with little prior programming experience, PPP participants reported *programming more as something they want to learn* ($z=1.997$, $p=0.046$, $r=0.48$) and *less boring* ($z=-1.961$, $p=0.050$, $r=0.48$) in addition to the attitude shifts described above. Although these results indicate attitude improvement and signal support for **H1**, the lack of longitudinal data poses a threat to internal validity, as we cannot claim change at study conclusion persists. The small sample also prevented the finding of significant differences when comparing between conditions. For example, participants training via PPPs reported a notable but non-significant decrease in their perception that *programming is a foreign concept* compared to PPPD and LCF participants ($H(2)=4.369$, $p=.113$, PPP vs. PPD: $p=.309$, PPP vs. LCF: $p=.186$). Related mean changes and within-subject results are included in Table 3.

Table 3 Within-subject attitude change. Positive shifts (p), negative shifts (n). * $p<0.05$, ** $p<0.01$

Programming is...	PPP	PPP-distractor	limited-constraint-feedback
something I've wanted to learn (p)	$M=0.19$, $SD=1.40$	$M=0.27$, $SD=1.31$	$M=0.00$, $SD=1.19$
fun (p)	$M=0.74$, $SD=1.67^*$	$M=0.40$, $SD=1.74$	$M=0.36$, $SD=1.43$

enjoyable (p)	M=0.90, SD=1.83*	M=-0.05, SD=1.68	M=0.68, SD=1.76*
important to know (p)	M=0.25, SD=1.48	M=-0.05, SD=1.17	M=0.09, SD=1.19
easy to start (p)	M=1.35, SD=2.29*	M=0.68, SD=1.13*	M=0.45, SD=1.71
something that takes practice (p)	M=0.065, SD=1.09	M=0.05, SD=1.29	M=-0.32, SD=1.17
too difficult to understand (n)	M=-1.48, SD=2.03**	M=-0.77, SD=1.77	M=-0.64, SD=1.89
boring (n)	M=-0.41, SD=1.6	M=-0.32, SD=1.17	M=-0.54, SD=1.90
a foreign concept (n)	M=-1.13, SD=1.83*	M=-0.27, SD=1.55	M=0.00, SD=2.07
too time consuming (n)	M=-0.35, SD=2.09	M=-0.09, SD=1.27	M=-0.09, SD=2.44

To supplement the quantitative results, we sought qualitative feedback by requesting that participants describe their attitude or view toward programming after the learning experience. For both those who self-reported low and high prior programming experience, we recorded more hesitant responses from those who trained via limited constraint and feedback than those who trained via PPPs and PPPDs. One LCF participant who selected “have tried programming activities, but have not taken a class” in the demographic survey, reflected on sustained struggle: “I still feel like programming is insanely complex. When I was in college I dropped out of computer science as soon as we started python. I just couldn't understand what we were doing, and maybe I could understand it if I really tried. It just seems to be better geared towards certain people.” A second LCF participant with the same prior programming experience selection revealed marginal incremental motivational change: “I have already begun to study programming but have not stayed consistent with my studies. This has encouraged me to give more attention to the subject.” A third LCF participant who selected “have tried programming activities, but not taken a class” alluded to seeking external supports: “I hope to translate what I have gained today to my studies in coding. It is a bit tedious, and there is a lot to know, but I think that many basic codes can be written with the help of a search engine or some material.”

In contrast, PPP and PPPD participants reflected more direct positive attitudinal change. One PPP participant whose prior programming experience selection was “never attempted to program before” noted that she “definitely enjoyed the puzzles and feel[s] more knowledgeable in terms of programming. It made me much more interested in learning to program.” A second PPP participant who recorded the same prior programming experience discovered possibility in her capability: “I feel like it's not as complicated as I thought it was. i could learn a lot through practicing more of it.” A third PPP participant who selected

“have tried programming activities, but have not taken a class” demonstrated confidence in his ability as well as opportunity for novices: “[T]his activity was somewhat easy but programming is really much harder than this. [B]ut this is a good way for a kid to start learning.” Aligned with this viewpoint was one PPPD participant who selected “never attempted to program before” and revealed potential for future pursuit of CT: “I would love to learn more about programming and encourage my son to start learning programming early.” A second PPPD participant who selected “have tried programming activities, but not taken a class” focused on the puzzle approach to learning in his response: “I think it is a skill that can be learned through practice. It was nice to look at programming as a series of puzzles rather than a complex language.” These results support **H1** and those in (Charters, Lee, Ko, & Loksa, 2014), which found significant attitude improvement regardless of gender and education level after a brief online programming experience.

Motivation

We conclude the analysis by summarizing findings for each varied PPP element in Table 4.

Table 4 General summary of findings across conditions

Cond.	Extraneous Cognitive Load	Instructional Efficiency	Motivation/ Attitude
PPP	1-palette	0	Correctness
PPPD	1-palette	2-4	Correctness & distractor notification
LCF	All palettes	All Scratch blocks	None

Conclusion & Future Work

Our survey of grade 6-9 teachers exposed teacher perceptions of limited student engagement with data concepts central to CT. These results led us to extend the trend of balancing Scratch’s agency with structure to better serve learners and reduce burden on teachers. A small pilot study of an adult population using a learning system that integrates PPPs with Scratch yielded results indicating the structure provided by PPPs catalyzes motivation for CT, reduces extraneous cognitive load, and increases learning efficiency without sacrificing performance on transfer tasks.

While these results reveal opportunities to advance the teaching and learning of CT via augmentations to block-based programming environments, we remain cautious due to external validity limitations: the single CT concept, *sequences*, and small summative evaluation population (75 adults), threaten generalizability. In future work, we intend to study additional CT concepts, such as *conditionals* and *looping*, functionality variation, such as offering increasing agency through the introduction of teacher-defined, objective-

driven feedback, the fading of correctness feedback, and the configurable integration of multiple Scratch palettes for each puzzle, and participants, including online studies with over 500 adults as well as smaller, middle school classroom studies. These conditions should facilitate the study of CT learning beyond that of the beginners under focus in this study by offering tooling to apply incremental cognitive load, deepen CT concept uptake, and transition learners toward interest-driven projects that sustain motivation. With continued investigation, we aim to identify factors supportive of reliably efficient, effective, and equitable CT learning that build bridges between cognitive, situated, and critical CT.

Abbreviations

CT: computational thinking; PPP: Parsons programming puzzles; PPPD: Parsons programming puzzles with distractors; LCF: Parsons programming puzzles with limited constraint and feedback (similar to traditional block-based programming); CLT: cognitive load theory; IL: intrinsic load; EL: extraneous load; GL: germane load; EI: mental effort during instruction; ET: mental effort during transfer tasks; P: performance on transfer tasks; CS CLS: computer science cognitive load component survey; PCK: pedagogical content knowledge; TEQ: intrinsic motivation task evaluation questionnaire.

Endnotes

¹This paper is an extended version of the paper “Integrating Parsons Puzzles with Scratch” presented at the 29th International Conference on Computers in Education (ICCE 2021). It includes additional detail in CS education context, software development, study design, participants, materials, and results to encourage and facilitate replication.

Acknowledgements

We especially thank teachers from participating schools such as South Bronx Early College Academy and Life Sciences Secondary School, and informal learning environments such as codeHER and Girls Who Code, as well as the several dozen contributing university student researchers. Criteria for authorship included substantial contribution to the research and analysis.

Author's contributions

JB participated in the development of the learning system, the design of the formative and summative protocols, and wrote this paper. BZ created materials for the summative evaluation and led the data analysis. AD participated in the development of the learning system, assisted in the formative evaluation, and implemented the supplementary website. GK guided and supervised all aspects of this research.

Funding

The Programming Systems Laboratory is supported in part by DARPA N6600121C4018, NSF CCF-1815494 and NSF CNS-1563555.

Availability of data and materials

Protocol materials and study data are available in referenced website Integrating Parsons Puzzles with Scratch, <https://ippssupplement.github.io/>. The developed software, SAGE, is available at <https://github.com/cu-sage>. It is platform independent, involves Node.js, AngularJS, ActionScript, and MongoDB, with licensing via GPL.

Declarations

Competing interests

The authors declare that they have no competing interests.

Author details

The authors are affiliated with the Department of Computer Science, Columbia University, New York, NY.

Received: (Date) Accepted: (Date)

Published online: (Date)

References

- Aivaloglou, E., & Hermans, F. (2019). Early programming education and career orientation: the effects of gender, self-efficacy, motivation and stereotype. *ACM SIGCSE*, (pp. 679-685).
- Amazon Mechanical Turk. (2021). <https://www.mturk.com/>. Accessed May 2020.
- Barab, S. (2014). Design-based research: a methodological toolkit for engineering change. In K. Sawyer (Ed.), *The Cambridge Handbook of the Learning Sciences*. Cambridge University Press.
- Barendsen, E., Mannila, L., Demo, B., Grgurina, N., Izu, C., Mirolo, C., . . . Stupurienė, G. (2015). Concepts in K-9 computer science education. *ACM ITICSE-WGR*, (pp. 85-116).
- Bovermann, K., & Bastiaens, T. J. (2020). Towards a motivational design? Connecting gamification user types and online learning activities. *Research and Practice in Technology Enhanced Learning*, 15(1), (pp. 1-18).
- Brennan, K. (2013). *Best of both worlds: issues of structure and agency in computational creation*. MIT.
- Brennan, K., Balch, C., & Chung, M. (2014). *Creative computing*. Harvard Graduate School of Education.
- Brusilovsky, P., Mlmi, L., Hosseini, R., Guerra, J., Sirkä, T., & Pollari-Malmi, K. (2018). An integrated practice system for learning programming in Python: design and evaluation. *Research and Practice in Technology Enhanced Learning*, 13(1), (pp. 1-40).
- Bush, J. B., Gilmore, M. R., & Miller, S. B. (2020). Drag and drop programming experiences and equity: analysis of a large scale middle school student motivation survey. *ACM SIGCSE*, (pp. 664-670).
- Cao, L., Rorrer, A., Pugalee, D., Maher, M. L., Dorodchi, M., Frye, D., . . . Wiebe, E. (2020). Work in progress report: a STEM ecosystem approach to CS/CT. *ACM SIGCSE*, (pp. 999-1004).
- Casanova, D., Alsop, G., & Huet, I. (2021). Giving away some of their powers! Towards learner agency in digital assessment and feedback. *Research and Practice in Technology Enhanced Learning*, 16(1), (pp. 1-19).
- Charters, P., Lee, M., Ko, A., & Loksa, D. (2014). Challenging stereotypes and changing attitudes: the effect of a brief programming encounter on adults' attitudes toward programming. *ACM SIGCSE*, (pp. 653-658).
- CodeHER. <https://www.codehergirls.org/>. Accessed December 2021.
- DeLyser, L. (2018). A community model of CSforALL. *ACM ITICSE*, (pp. 99-104).
- Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: parsons problems. *ICCE*, (pp. 113-124).
- Difallaha, D., Filatova, E., & Ipeirotis, P. (2018). Demographics and dynamics of mechanical Turk workers. *ACM Web Search and Data Mining*, (pp. 135-143).
- Dong, Y., Marwan, S., Catete, V., Price, T., & Barnes, T. (2019). Defining tinkering behavior in open-ended block-based programming assignments. *SIGCSE*, (pp. 1204-1210).
- Emerson, A., Smith, A., Rodriguez, F., W. E., & Bradford, W. (2020). Cluster-based analysis of novice coding misconceptions in block-based programming. *ACM SIGCSE*, (pp. 825-832).
- Ericson, B., ...Rick, J. (2018). Evaluating the efficiency and effectiveness of adaptive parsons problems. *ICER*, (pp. 60-68).
- Ericson, B., ... Rick, J. (2017). Solving parsons problems versus fixing and writing code. *Koli Calling ICER*, (pp. 20-29).
- Fabir, G. V. F., Mitrovic, A., & Neshatian, K. (2018). Investigating the effects of learning activities in a mobile Python tutor for targeting multiple coding skills. *Research and Practice in Technology Enhanced Learning*, 13(1), (pp. 1-24).
- Frädich, C., Obermüller, ... Fraser, G. (2020). Common bugs in Scratch programs. *ACM ITICSE*, (pp. 89-95).
- Franklin, D., Weintrop, D., Palmer, J., Coenraad, M., Cobian, M., Beck, K., . . . Crenshaw, Z. (2020). Scratch Encore: the design and pilot of a culturally-relevant. *ACM SIGCSE*, (pp. 794-800).
- Garner, S. (2007). An exploration of how a technology-facilitated part-complete solution method supports the learning of computer programming. *Informing Science & Information Technology*.
- Girls Who Code. <https://girlswhocode.com/>. Accessed December 2021.
- Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., & Stoker, I. (2014). Computational thinking skills in dutch secondary education: exploring pedagogical content knowledge. *ACM Koli Calling*, (pp. 173-174).
- Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. *ACM SIGCSE*, (pp. 267-272).
- Grover, S., & Pea, R. (2013). Computational thinking in K-12: a review of the state of the field. *Edu. Res.*, 42(1), (pp. 38-42).
- Grover, S., Basu, S., & Schank, P. (2018). What we can learn about student learning from open-ended programming projects in middle school computer science. *ACM ITICSE*, (pp. 999-1004).
- Harel, I., & Papert, S. (1991). *Constructionism*. Ablex Publishing.
- Harms, K., Balzuweit, E., Chen, J., & Kelleher, C. (2016). Learning programming from tutorials and code puzzles: children's perceptions of value. *IEEE Visual Languages and Human-Centric Computing*, (pp. 59-67).
- Harms, K., Chen, J., & Kelleher, C. (2016). Distractors in parsons problems decrease learning efficiency for young novice programmers. *International Computing Education Research*, (pp. 241-250).
- Harms, K., Rowlett, N., & Kelleher, C. (2015). Enabling independent learning of programming concepts through programming completion puzzles. *IEEE Visual Languages and HCC*, (pp. 271-279).

- Helminen, J., Ihantola, P., Karavirta, V., & Alaoutinen, S. (2013). How do students solve parsons programming problems? – execution-based vs. line-based feedback. *Learning and Teaching in Computing and Engineering*, (pp. 55-61).
- Helminen, J., Ihantola, P., Karavirta, & Malmi, L. (2012). How do students solve parsons programming problems? An analysis of interaction traces. *ICCE*, (pp. 119-126).
- Ihantola, P., & Karavirta, V. (2011). Two-dimensional parson's puzzles *IT Education*, 10, (pp. 119-132).
- Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S., . . . Rubio, M. (2016). Educational data mining and learning analytics in programming: literature review and case studies. *ITICSE WGR*, (pp. 41-63)
- Integrating Parsons Puzzles with Scratch. <https://ippssupplement.github.io/>. Accessed December 2021.
- Kaczmarczyk, L.C., Petrick, E. R., East, J. P., & Herman, G. L.. (2010). Identifying student misconceptions of programming. *SIGCSE*, (pp. 107-111).
- Kafai, Y., Proctor, C., & Lui, D. (2020). From theory bias to theory dialogue: embracing cognitive, situated, and critical framings of computational thinking in K-12 CS education. *ACM Inroads*, (pp. 44-53).
- Karavirta, V., Helminen, J., & Ihantola, P. (2012). A mobile learning application for parsons problems with automatic feedback. *International Conference on Computing Education Research*, (pp. 11-18).
- Karvelas, I., Li, A., & Becker, B. A. (2020). The effects of compilation mechanisms and error message presentation on novice programmer behavior. *ACM SIGCSE*, (pp. 759-765).
- Kashmira, D., & Mason, J. (2020). Empowering learning designers through design thinking. *ICCE*, (pp. 497-502).
- King, K. V., Gardner, D. A., Zucker, S., & Jorgensen, M.A. *The distractor rationale taxonomy: enhancing multiple-choice items in reading and mathematics*. Pearson.
- Kong, S., Lai, M., & Siu, C. (2020). Development of CT concepts in Scratch programming. *ICCE*, (pp. 652-657).
- Kumar. (2017). The effect of providing motivational support in parsons puzzle tutors. *AI in Ed.*, (pp. 528-531).
- Kumar, A. N. (2019a). Helping students solve Parsons puzzles better. *ACM ITICSE*, (pp. 65-70).
- Kumar, A. N. (2019b). Mnemonic variable names in Parsons puzzles. *ACM CompEd*, (pp. 120-126).
- Lewis, C., Bruno, P., Raygoza, J., & Wang, J. (2019). Alignment of goals and perceptions of computing predicts students' sense of belonging in computing. *ACM ITICSE*, (pp. 11-19).
- Leyzberg, D., & Moretti, C. (2017). Teaching CS to CS teachers. *ACM SIGCSE*, (pp. 369-374).
- Maloney, J., .. Eastmond, E. (2010). The Scratch programming language and environment. *Computing Education*, (pp. 1-15).
- Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational thinking in K-9 education. *Innovation & Technology in Computer Science Education*, (pp. 1-29).
- Marwan, S., Gao, G., Fisk, S., Price, T., & Barnes, T. (2020). Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science. *ACM ICER*, (pp. 194-203).
- McGill, M. M., & Decker, A. (2020). Tools, languages, and environments used in primary and secondary computing education. *ACM ITICSE*, (pp. 103-109).
- Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in Scratch. *ACM ITICSE*, (pp. 168-172).
- MIT Media Lab. (n.d.). *Scratch Statistics*. <https://scratch.mit.edu/statistics/>. Accessed December 2021.
- Morrison, B., Dorn, B., & Guzdial, M. (2014). Measuring cognitive load in introductory CS. *ICER*, (pp. 131-138).
- Morrison, B., .. Guzdial, M. (2016). Subgoals help students solve Parsons problems. *ACM SIGCSE*, (pp. 42-47).
- Ohland, C., Ehsan, H., & Cardella, M. E. (2019). Parental influence on children's computational thinking in an informal setting (fundamental research). *ASEE*.
- Panopto. <https://www.panopto.com/>. Accessed December 2021.
- Parsons, D., & Haden, P. (2006). Parson's programming puzzles: a fun and effective learning tool for first programming courses. *Australian Conference on Computing Education*, (pp. 157-163).
- Paas, F., & Merrienboer, J. (1993). The efficiency of instructional conditions. *Human Factors*, 35(4), 737-743.
- Qualtrics. (n.d.). *Qualtrics*. <https://qualtrics.com>. Accessed December 2021.
- Raubenheimer, G., Jeffries, B., & Yacef, K. (2021). Toward empirical analysis of pedagogical feedback in computer programming learning environments. *Australian Conference on Computing Education*, (pp. 189-195).
- Razzouk, R. & Shute, V. (2012). What is design thinking and why is it important? *Review of Educational Research*, 82(3), (pp. 330-348).
- Rich, P. J., Browning, S., Perkins, M., Shoop, T. Y., & Belikov, O. M. (2019). Coding in K-8: international trends in teaching elementary/primary computing. *Tech Trends*, 63(3), 311-329.
- Salac, J., Thomas, C., Butler, C., Sanchez, A., & Franklin, D. (2020). TIPP&SEE: a learning strategy to guide students through use-modify Scratch activities. *SIGCSE*, (pp. 79-85).
- Scaffidi, C., & Chambers, C. (2012). Skill progression demonstrated by users in the Scratch animation environment. *International journal of Human-Computer Interaction*, 28, 383-398.
- Schulte, C., & Knobelsdorf, M. (2007). Attitudes towards computer science-computing experiences as a starting point and barrier to computer science. *ACM ICER*, (pp. 27-38).
- SDT. (n.d.). Self-determination Theory. <https://selfdeterminationtheory.org/intrinsic-motivation-inventory/>. Accessed April 2019.
- Sirkia, T. (2016). Combining parson's problems with program visualization in CS1 context. *Koli Calling*, (pp. 155-159).

- Sulaiman, J., Dziena, A., Bender, J., & Kaiser, G. (2019). SAGE-RA: a reference architecture to advance the teaching and learning of computational thinking. *Embedding AI in Education Policy and Practice for Southeast Asia*, (pp. 421-431).
- Sweller, J. (2010). *Cognitive Load Theory: Recent Theoretical Advances*. Cambridge University Press.
- Tahir, F., Mitrovic, A., & Sotardi, V. (2020). Investigating the effects of gamifying SQL-Tutor. *ICCE*, (pp. 416-425).
- Tamatea, L. (2019). Compulsory coding in education: liberal-humanism, Baudrillard and the 'problem' of abstraction. *Research and Practice in Technology Enhanced Learning*, 14(1), (pp. 1-29).
- The Royal Society. (2016). *After the Reboot: Computing Ed. in UK Schools*.
<https://royalsociety.org/~media/policy/projects/computing-education/computing-education-report.pdf>.
 Accessed June 2017.
- Tsur, M., & Rusk., N. (2018). Scratch microworlds: designing project-based introductions to coding. *SIGCSE*, (pp. 894-899).
- van Gog, T., & Paas, F. (2008). Instructional efficiency: revisiting the original construct in educational research. *Educational Psychologist*, 43(1), (pp. 16-26).
- Villamor, M. M. (2020). A review on process-oriented approaches for analyzing novice solutions to programming problems. *Research and Practice in Technology Enhanced Learning*, 15(1), (pp. 1-23).
- Whitehouse.gov. (2016). *CS for All*. <https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all>.
 Accessed February 2019.
- Wilson, C., Sudol, L., Stephenson, C., & Stehlik, M. (2010). *Running on Empty: The Failure to Teach K-12 CS in the Digital Age*. CSTA. <http://www.acm.org/runningonempty/fullreport2.pdf>. Accessed August 2017.
- Wing, J. (2006). Computational Thinking. *Communications of the ACM*, 49(3), 33-35.
- Wing, J. (2008). Computational thinking and thinking about computing. *Philosophical Transactions*, 355(1881), (pp. 3717-3725).
- Zhi, R., Chi, M., Barnes, T., & Price, T. W. (2019). Evaluating the effectiveness of parsons problems for block-based programming. *ACM ICER*, (pp. 51-59).

Publisher's Note

The Asia-Pacific Society for Computers in Education (APSCE) remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.