Distractors in Parsons Problems Decrease Learning Efficiency for Young Novice Programmers

Kyle J. Harms, Jason Chen, Caitlin Kelleher
Department of Computer Science & Engineering
Washington University in St. Louis
One Brookings Drive
St. Louis, MO 63130, United States
{kyle.harms, chenjy, ckelleher}@wustl.edu

ABSTRACT
Parsons problems are an increasingly popular method for helping inexperienced programmers improve their programming skills. In Parsons problems, learners are given a set of programming statements that they must assemble into the correct order. Parsons problems commonly use distractors, extra statements that are not part of the solution. Yet, little is known about the effect distractors have on a learner's ability to acquire new programming skills. We present a study comparing the effectiveness of learning programming from Parsons problems with and without distractors. The results suggest that distractors decrease learning efficiency. We found that distractor participants showed no difference in transfer task performance compared to those without distractors. However, the distractors increased learners cognitive load, decreased their success at completing Parsons problems by 26%, and increased learners' time on task by 14%.

General Terms
Human Factors

Keywords
Independent Learning, Parsons Problems, Distractors, Completion Problems, Cognitive Load

1. INTRODUCTION
Parsons problems are a popular way to help individuals learn computer programming. In a Parsons problem, a learner is given a program where the statements have been placed out of order. Learners solve the Parsons problem by assembling the statements into the correct order [47]. Frequently, Parsons problems also include extra statements that are not part of the solution, known as distractors [22, 47]. Parsons problems are typically used for drill and practice exercises that complement traditional classroom and online learning [10, 47], or for assessing learners' programming skills [8, 41]. More recently, they have also been used to help children learn programming independently [17].

Distractors are a common feature of many Parsons problem implementations [8, 10, 22, 47]. Yet, little is known about how distractors affect learning. Broadly, distractors in Parsons problems are often described as unnecessary code [10], extra fragments [18], or erroneous code [21]. Fortunately, some researchers provide a more precise explanation: distractors should be used to "illustrate a particular point" or to "highlight programming principles the student may ignore" [47]. Distractors based on these principles often attempt to illustrate common programming misconceptions and syntax errors [25, 47]. While this seems like a reasonable approach for using distractors, we are unaware of any empirical evidence on the effectiveness of distractors in Parsons problems.

In this paper, we present a study investigating middle school children's ability to learn programming concepts independently from Parsons problems with distractors. We conducted a formative evaluation exploring the potential use of distractors to encourage and foster a beneficial learning experience. We then report the results of a summative evaluation comparing the effectiveness of using *partial suboptimal path* distractors when learning programming concepts on transfer task performance. Our results show that distractors increased cognitive load for independent learners, reduced learners' ability to successfully complete Parsons problems, and significantly increased time on task. Distractor participants also showed no difference in transfer task performance compared to participants who trained without distractors.

2. RELATED WORK
At its core, a distractor is an *error*. Some suggest that the errors generated from distractors in Parsons problems may aid learning [25, 47]. Further, the reasons cited for using distractors in Parsons problems are similar to the reasons for using distractors in multiple choice tests [27, 33, 47]. In this section, we review research on the impacts of 1) generating errors on learning and 2) the use of distractors in testing.

2.1 Learning with Errors
Learners may encounter errors when learning from traditional educational materials or learning based games. We first discuss how errors affect learners in traditional education situations, followed by game based learning.

Humans naturally make errors as part of our learning process. Generating errors during learning can have both positive and negative consequences that educators must be careful to either leverage or mitigate. One approach to reducing errors made during learning is to teach common misconceptions to students when introducing new material [42]. This is especially important for low performing students, who benefited from learning about common misconceptions [42]. Errors can also affect high and low performing learners differently. Only high performing students benefited when learning via worked examples that contained errors [13]. Fortunately, providing students with corrective feedback on the errors mitigated the harmful effects for low performing students

ICER '16, September 08 - 12, 2016, Melbourne, VIC, Australia

ACM 978-1-4503-4449-4/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2960310.2960314

[61]. Another common approach to learning with errors is using *trial and error*. Researchers demonstrated that generating errors when learning via trial and error helped younger learners, but impaired older adult learners [7]. Specifically in the context of programming, errors did not benefit learners. Students who learned programming using a method that deliberately encouraged them to make errors had their rote learning efficiency reduced and they tended to make the same errors later [9]. The inconsistency among the literature suggests that educators must take into consideration learners' abilities and age, feedback and even the educational domain when considering the effects of errors during learning.

There is also a growing body of evidence on the effectiveness of game based learning for several disciplines [58]. Some of these games leverage correcting errors as a central game mechanic [19, 30], while others use distractors to lure players into generating errors [55]. Correcting errors as part of a debugging game has been shown to be an effective strategy for helping children learn programming [30]. While Parsons problems are not a game, they have puzzle-like qualities and have been shown, without distractors, to be effective for learning [17]. We are unaware of any empirical evidence of the benefit or harm that generating errors from distractors in Parsons problems may cause.

2.2 Distractors and Errors in Testing

Multiple choice questions typically have several incorrect answers designed to distract lower level learners away from the correct answer [27]. Test takers also acquire new information from the act of taking a test [51]. Depending on the circumstances, the errors generated from distractors during testing may benefit or harm learners [5]. We discuss how testing affects learning and how the errors generated by distractors impact learners.

While testing may be viewed as assessing learned material, it also plays an important role in learning [51, 52]. When testing, individuals must retrieve information from their memory. Information is not stored statically in memory, but rather it is reconstructed upon retrieval; retrieval can change the information itself [26]. The act of retrieving information from memory can be an effective way to promote learning and even retention of information [52]. This is known as the *testing effect* [51]. The key to the testing effect is the retrieval process itself. The act of trying to retrieve information during testing, even if unsuccessful, was found to enhance future learning [29, 49]. Further, if an individual fails to retrieve the information, guessing the answer did not hurt future performance [23]. The testing effect persists across testing formats, including multiple choice tests with distractors [34, 53].

If a test taker constructs an incorrect answer during retrieval, there is the possibility that the error is committed to memory which can impair future performance [1]. Test authors should be especially careful when authoring multiple choice tests with distractors because individuals may acquire false information from any errors [5, 36–38, 53]. Additionally, low performing students are more likely to acquire false information from multiple choice testing [5, 37], while higher performing students actually benefit from the errors [5]. Learning false information in testing appears to be due to faulty reasoning when selecting answers [38]. If faulty reasoning is at play, this can be mitigated by providing corrective feedback after testing [4, 6]. Feedback was found to improve learning when test takers generated errors during multiple choice testing [4, 6, 20, 48]. Given the potential for learning false information from multiple choice testing, test administrators should take measures, like feedback, to limit the potential for false learning.

Guidelines based on empirical evidence may also help test authors write better multiple choice tests with distractors [15, 16]. Even then, writing good distractors is often difficult for test authors [57]. Part of the difficulty in writing quality distractors is providing responses that separate low and high performers [50], but that also provide insights about students' misunderstandings [27, 33]. One approach to authoring multiple choice distractors is to utilize common misconceptions [15, 39]. In our context, misconceptions can be common programming mistakes, for example, failing to properly nest statements in a control flow construct [2].

Overall, generating errors during learning may help learners under the right conditions. However, there is the potential that generating errors may also decrease the effectiveness of learning. Little empirical evidence exists to suggest how generating errors in Parsons problems will affect learners. Beneficial use of distractors in Parsons problems may only be effective under certain circumstances, similar to the learning with errors research. In this paper, we explore the potential for using distractors in Parsons problems to help novices learn programming independently.

3. LOOKING GLASS

For our study, we chose to use the blocks-based novice programming environment, Looking Glass [35]. In Looking Glass, users author programs that are 3D animations. Looking Glass also has support for Parsons problems as shown in Figure 1-A/B.

The Parsons problems in Looking Glass (known as programming completion problems) were influenced by prior work that demonstrated that high school students who completed partially written programs (i.e. completion problems) later constructed better quality programs than students who authored programs from scratch [59, 60]. To further aid independent learners, these code puzzles in Looking Glass include additional scaffolding within the interface to reduce the cognitive load that impairs learning (i.e. extraneous cognitive load) and increase the cognitive load that fosters learning (i.e. germane cognitive load) [17]. This includes a rich feedback mechanism designed to help learners correct errors without providing the answer [17]. In an evaluation, novices who learned programming constructs using programming completion problems demonstrated more evidence of learning compared to novices who learned using tutorials [17]. In the remainder of this paper, we refer to these Parsons problems simply as *puzzles*.

4. FORMATIVE EVALUATION

We conducted a formative evaluation to explore how to author distractors in puzzles to exploit the benefits of learning with errors while reducing the potential harm they may cause. Even though the most cited use of a distractor in Parsons problem is the programming syntax distractor [8, 10, 18, 22, 25, 47], we did not explore this distractor in our study. Blocks-based programming languages forgo syntax in an effort to reduce the barriers for learning programming. From our evaluation, we share three guidelines for distractors in code puzzles: 1) distractors that create *extra noise* are easy to ignore, 2) distractors should encourage learners to follow a *familiar, but suboptimal path*, and 3) allow only *one possible solution* to a puzzle.

For our formative study, we recruited 16 participants between the ages of 10 and 15 (10 female, 6 male; age: $M = 11.94$, $SD = 1.57$) from the Academy of Science of St. Louis mailing list. The Academy of Science is a not-for-profit organization dedicated to science outreach in the St. Louis metropolitan area. We compensated participants with a $10 gift certificate.

4.1 Extra Noise is Easy to Ignore

Prior work described distractors as extra code or unnecessary code in Parsons problems [10, 18]. We initially came up with three unnecessary code distractors: 1) create additional *unrelated* random

noise, 2) create additional *tangentially related* noise, and 3) insert *unrelated control flow constructs*.

For the *unrelated noise* distractors we added random method invocations to the puzzles, for example: *ufo.resize(2.0)*. Users very quickly realized that the random statements are easy to eliminate from their possible solution space, as one participant stated, "the extra actions (statements) contradict, so you can ignore them." We also tested *tangentially related* distractors by inserting additional method invocations that could plausibly be part of an animation. For example, in an animation about a dolphin rescuing a sinking boat, we inserted a method call to a background ship object, *ship.sail_towards(boat)*. However, participants gave similar explanations that they knew they could just ignore it because the ship never sailed towards the boat.

Given that participants were quick to dismiss the *extra noise*, we tried another distractor designed to encourage participants to make decisions about which control flow blocks they might need for the solution. We inserted several types of *unrelated control flow blocks* as distractors. We knew that this approach better engaged some participants when they asked, "Do I have to use all of them?" Further, the extra control blocks appeared to encourage thinking for some participants, as one participant stated: "It's sorta tricky, but it also makes you think harder, like what does the computer want me to do?" However, we note that adding unrelated control blocks were also easy to dismiss as unnecessary.

4.2 Use a Familiar, but Suboptimal Path

Instead of adding extra noise to the puzzles, we tried to engage participants by creating distractors that encouraged them to follow a *familiar, but suboptimal path*, when solving the puzzle. The idea is similar to the *misconception* distractors mentioned in Parsons problems [47]. The distractors allow a user to create a suboptimal solution for a puzzle that follows a familiar strategy that they have likely used before. For example, using identical duplicate statements instead of using a loop as shown in Figure 1-E. We used three strategies for suboptimal path distractors: 1) insert distractor statements into the solution for the puzzle's *initial state* (Figure 1-C), 2) *initially nest* constructs incorrectly (Figure 1-C), and 3) add *alternative statements* that can lead to a suboptimal solution (e.g. duplicate statements instead of a loop) (Figure 1-E).

Compared to the previous *noisy* distractors, participants engaged with the suboptimal path distractors frequently. For the *initial state* distractor one participant stated, "This one you have to think that you can move the pieces on the board already off." In *initial nesting* distractors, participants would recognize that they needed the combined effects of both constructs for the solution. Yet, participants had difficulty realizing that they needed to switch the nesting order, "I'm a little bit irritated. I've checked over and over the difference between my video and the correct one."

When participants followed the suboptimal path, it often led to them failing to recover and failing to discover the optimal solution. This was especially true for the *alternative statement* distractors. Participants had difficulty understanding why their solution's output looked correct even though the feedback indicated it was incorrect, as one participant stated, "It makes me feel a little frustrated. Well I looked at the video many times. I figured out what I was missing. I corrected my animation. But looking at it… and it tells me that it's wrong [and that] makes me feel frustrated." Even if many participants failed to recover, the suboptimal path distractors are a promising approach given that participants were frequently actively engaged with the distractors.

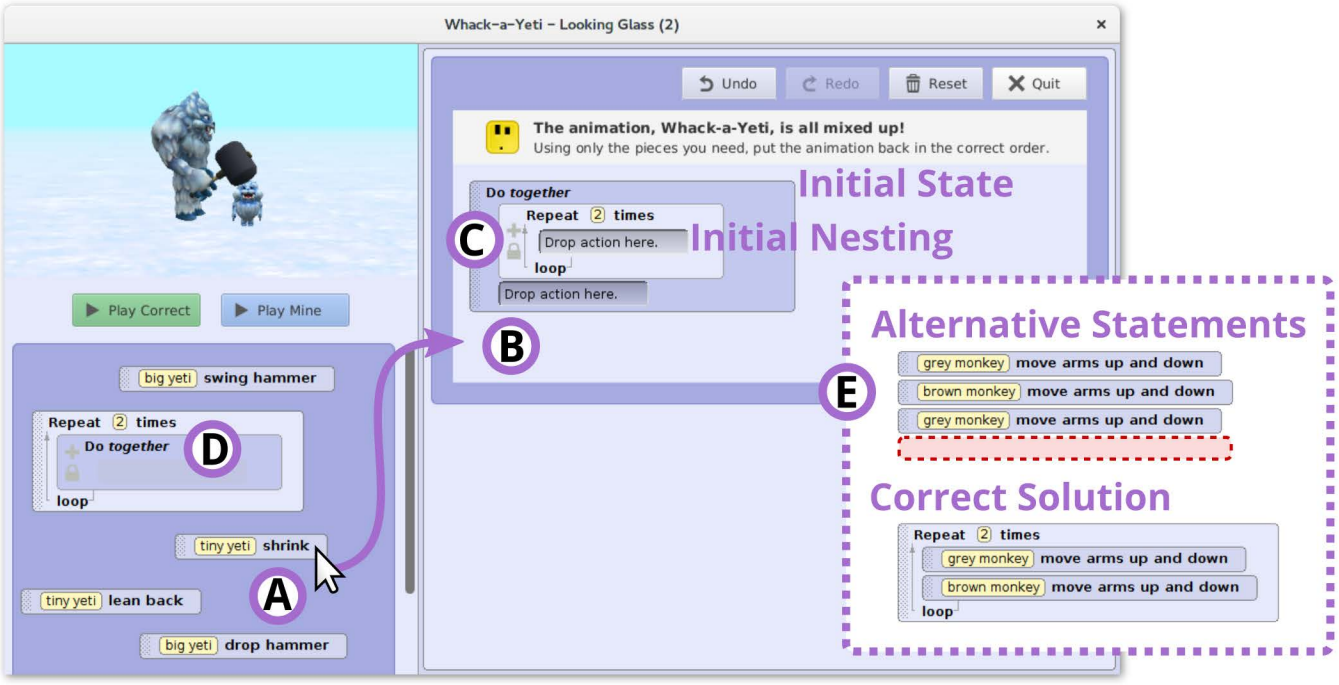

Figure 1. A puzzle (training task 5) in its initial state demonstrating *partial suboptimal path* distractors. Users solve a puzzle by assembling unused statements (A) into the correct order (B). In the *initial state* distractor (C), an incorrect *Do together* block is initially placed into the puzzle's solution. The *repeat* block is initially nested incorrectly into the *Do together* block for the *initial nesting* distractor; the correct solution is nested block (D). For the *alternative statements* distractor (E) (training task 2), users are given almost enough extra statements to create a suboptimal solution; the last statement is missing for the suboptimal solution.

4.3 Allow Only One Solution

The *familiar, but suboptimal path* distractors succeeded in leading participants down a suboptimal path, but they frequently failed to bring participants towards the optimal solution. The biggest problem with the suboptimal path distractors is that they create several suboptimal solutions in addition to the optimal solution. For example, the *alternative statement* distractors for a loop produce a complete suboptimal solution with the duplicate statements instead of the optimal loop solution. To gently nudge participants towards the optimal solution, we modified the distractors by removing the possibility of suboptimal solutions while still keeping the ability to make some progress down a partial suboptimal path.

One frequent source of suboptimal solutions is empty control flow blocks inserted by participants into the solution. Extra control flow statements can lead to no operation behavior (NOP) in Parsons problems [22]. For example, a *Repeat* block with no nested statements does nothing when executed. The NOP makes the program output match the correct output, even though the solution is incorrect because of the unused block. This can be particularly confusing for novices when their output looks correct but the puzzle feedback indicates that their solution is incorrect. The *initial nesting* distractors have a high tendency to produce NOP behavior. For example, a participant may unnest an incorrectly nested control block and then forget to remove the block from the solution. Yet, novices see nothing wrong with this behavior, as one participant stated, "I get that it's extra, but I don't agree…"

To help participants realize the existence of the optimal solution, we eliminated the existence of suboptimal solutions. For the *alternative statement* distractors we provided enough statements to lead a user down the suboptimal path, but not enough statements to actually finish the suboptimal solution. For example, in Figure 1-E the last duplicate statement is missing from the puzzle, so a user can start the suboptimal path, but must seek out the optimal solution to solve the puzzle. We also removed the NOP suboptimal solutions typically produced by the *initial nesting* distractors by locking nested control flow blocks together as shown in Figure 1-C/D. Because the blocks are locked together, the user can explore the suboptimal path (Figure 1-C), but because they cannot unnest the blocks, they cannot produce a puzzle with NOP behavior. Instead a user must swap out the incorrectly nested blocks with the correctly nested one (Figure 1-D). We also carefully authored and tested our puzzles to ensure that only one solution existed for each puzzle.

Not allowing participants to complete the suboptimal path had the desired effect of encouraging participants to seek out the optimal solution: "Are there any other pieces? Looks like I'm missing one." The *partial suboptimal path* distractors also appeared to lead to greater success. Several participants commented on how they enjoyed the distractors, as one participant stated, "I actually like that. I think it's pretty cool that you throw in some random ones because that's kinda how puzzles work." For the remainder of our study, we used the *partial suboptimal path* distractors.

5. SUMMATIVE EVALUATION

Using what we learned in the formative evaluation, we conducted a between subjects study to assess the impacts that distractors have on learning new programming knowledge. Our study contained two phases: training and transfer. In the training phase participants learned new programming skills using code puzzles that we then evaluated in the transfer phase. We developed a series of six puzzles using the *partial suboptimal path* distractors. Our puzzles introduced participants to three programming nested constructs: *Repeat* nested in a *Repeat*, *Do Together* nested in a *Repeat*, and *Repeat* nested in a *Do Together*. From this evaluation, we intended to answer the following questions: 1) What effect do distractors have on task completion time, task success, and mental effort? and 2) Do distractor participants show more evidence of learning than those who learned programming concepts without them?

5.1 Participants

We recruited 102 participants between the ages of 10 and 15 from the Academy of Science of St. Louis mailing list. We screened participants for participation in any of our previous studies. We removed nine participants from our data set for prior participation. We also removed one participant from our data set for not completing most of the study materials. In total we analyzed the data for 92 participants (32 female, 60 male; age: $M = 12.01$, $SD = 1.67$). We compensated participants with a $10 gift certificate.

5.2 Materials

We developed, tested, and refined our study materials through a pilot study. For the pilot study we recruited 14 participants (9 female, 5 male; age: $M = 12.93$, $SD = 2.23$).

5.2.1 Familiarization Tasks

For each phase of the study, we developed familiarization tasks to help introduce participants to the format of the tasks and the mechanics of the programming environment.

5.2.2 Training Tasks

We developed six puzzles for the training tasks. The six tasks were designed to be completed sequentially. We developed two sets of training tasks: one set without distractors, the other with distractors. See Table 1 for the programming concepts and distractor types for each training task. The training familiarization task is identical in format to the actual training tasks. However, the training familiarization task did not contain any distractors.

5.2.3 Transfer Tasks

In the transfer phase, we wanted to acquire evidence that the participants learned the programming concepts from the training phase. Specifically, we wanted to know if participants could identify the correct programming concepts and the proper nested structure when given a novel problem. We developed three transfer tasks to evaluate participants' mastery of nesting a *Repeat* block within a *Repeat*, nesting a *Do Together* within a *Repeat*, and nesting a *Repeat* within a *Do Together*.

Each transfer task is a complete program with existing and correctly ordered statements as shown in Figure 2. The existing statements are only method invocations; there are no control flow constructs. To complete each task, participants need only identify and insert the correct control flow constructs from the programming environment as shown in Figure 2-A. Once a construct is inserted into the program, participants drag the appropriate existing statements, maintaining their order, into the newly inserted control block. Each transfer task also included a video of the correct animation, written instructions for completing the task, and inline

Table 1. Training Task Summary

Task	Programing Concept	Partial Suboptimal Path Distractors
1	Do Together	Alternative Statements
2	Repeat	Alternative Statements × 2
3	Repeat { Repeat }	Initial State; Alternative Stat.
4	Repeat { Repeat }	Initial State; Alternative Stat.
5	Repeat { Do Together }	Initial State; Initial Nesting
6	Do Together { Repeat }	Initial Nesting; Alternative Statements

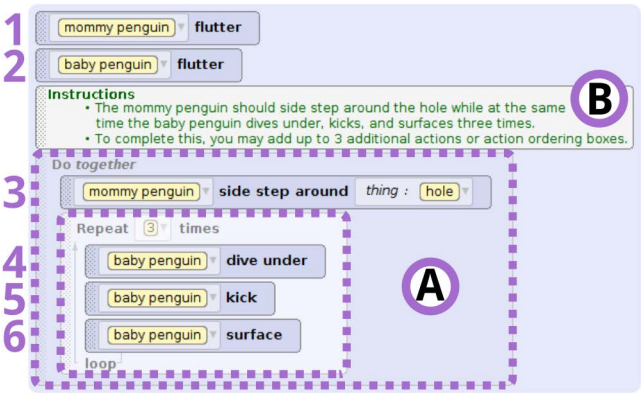

Figure 2. Example Transfer Task.
Statements 1-6 are already inserted and correctly ordered. Participants need to insert additional control blocks (A) to make the animation match the instructions (B).

comments describing the correct program output (Figure 2-B). The instructions noted that the existing statements are in the correct order and that participants are limited to only inserting three additional blocks into the program. We designed the transfer tasks to focus participants' time on demonstrating their programming knowledge, not authoring programs from scratch.

The transfer phase had two separate familiarization tasks. The first familiarization task introduced participants to the mechanics of authoring a sequential program in the programming environment. The second task is identical in format to the actual transfer tasks.

5.2.4 Surveys

We included three surveys in our evaluation: a self-developed programming experience survey, cognitive load task survey, and the Computer Science Cognitive Load Component Survey (CS CLCS) [40]. The programming experience survey gathered information about participants' age, gender, schooling, prior study participation, and prior programming education and experience.

The cognitive load task survey included two validated and reliable scales [45] for measuring cognitive load: mental effort [44] and difficulty [24]. Both scales are Likert item surveys with nine items from 1 (very, very low mental effort / extremely easy) to 9 (very, very high mental effort / extremely difficult). Historically when measuring cognitive load, researchers used either mental effort [44] or difficulty [24] scales to measure cognitive load. However, research now suggests that while mental effort and difficulty are correlated, they are not the same thing [12]. Because prior work on code puzzles used the difficulty scale [17], we included both scales for comparison. Please note that when we discuss cognitive load, we only refer to the mental effort scale.

We used a validated survey to measure the different types of cognitive load. The CS CLCS is an adaptation of an existing validated survey, the Cognitive Load Component Survey [31, 32], for the domain of computer science. The CS CLCS is a ten item Likert survey with three separate cognitive load scales: intrinsic, extraneous, and germane [40]. Each question is rated on an eleven-point scale from 0 (not at all the case) to 10 (completely the case).

5.3 Methods

We conducted our evaluation over several different multi-user sessions. Each participant attended a single two hour session. We seated participants to minimize viewing other participants' screens and to minimize interaction between participants.

We conducted a between subjects study with two conditions: control (no distractors) and distractors. We randomly assigned participants to either the control or the distractors condition (control: 47, distractors: 45). The study contained two parts: the training phase and the transfer phase as shown in Figure 3. In the training phase, participants completed puzzles with or without distractors as assigned in their condition. In the transfer phase, participants completed three transfer tasks. After completion of the transfer phase, we allowed participants to create their own animation or work on additional puzzles.

Before the study, we asked participants to complete the programming experience survey. Once completed, a member of the research team followed up on the survey responses by interviewing each participant about their responses. This follow up interview allowed us to correct any misreported data about participants' prior programming experience.

5.3.1 Training Phase

In the training phase, we asked participants to complete a familiarization task, six training tasks, and the CS CLCS. The familiarization task used the same format as the training tasks.

At the start of the training phase, we gave participants an instruction sheet with directions on how to complete a training task. We gave participants twelve minutes to complete each training task; there was no time limit for the familiarization task. We permitted participants to ask for help during the familiarization task. However, after completion of the familiarization task, we required participants to complete the remaining six training tasks without assistance. Each participant completed the training tasks in the same order. After completing each task, we asked participants to complete the cognitive load task survey where they rated their mental effort and difficulty for that task.

Upon completion of all training tasks, we gave participants a reminder sheet with a picture and the title for each of the training task animations (i.e. not the familiarization task). We then asked participants to complete the same cognitive load task survey ranking their mental effort and difficulty across all six tasks. We then asked them to complete the CS CLCS. We encouraged participants to reference the reminder sheet when completing these surveys to help them recall their experience.

5.3.2 Transfer Phase

After completing the training phase, participants began the transfer phase of the study. Similar to the training phase, participants first completed two familiarization tasks, three transfer tasks, and then the CS CLCS.

With each familiarization task, we provided an instruction sheet. The first familiarization task's instruction sheet demonstrated the basics of the Looking Glass programming environment. In the second familiarization task, the instruction sheet provided directions on how to complete the transfer tasks in the study. We allowed participants to keep both instruction sheets for the remainder of the transfer phase as a reference. During the familiarization tasks, participants could ask for help. However, we provided no assistance during the actual transfer tasks. We assigned the three transfer tasks using a balanced "Williams" Latin squares

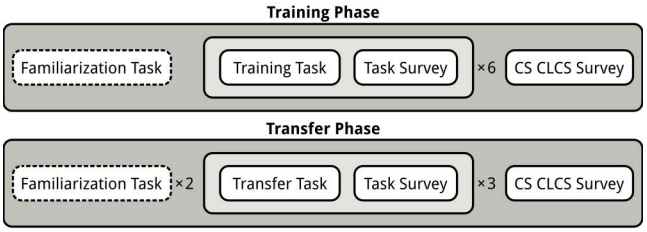

Figure 3. Summative Evaluation Procedure

design to control for learning effects [62]. Participants were given six minutes to complete each transfer task. Following the second familiarization task and each transfer task, we asked participants to complete the cognitive load task survey. Finally, upon completion of all transfer tasks, we gave participants a reminder sheet with each of the three transfer tasks' titles and pictures of each animation. We then asked them to complete a cognitive load task survey ranking their overall mental effort and difficulty for all transfer tasks and the CS CLCS using the reminder sheet.

6. ANALYSIS

We analyzed the data gathered in the training and transfer phases in the study. We also discuss how we control for external factors, like prior programming experience, in our analysis.

6.1 Training & Transfer Tasks

For the training and transfer tasks, we analyzed task time and performance, as well as the cognitive load scales. We collected cognitive load using several scales: mental effort across all tasks, difficulty across all tasks, overall intrinsic, extraneous, and germane cognitive loads. For each scale, we computed Cronbach's alpha for reliability ($\alpha > .70$).

Recall that we asked participants to rate their cognitive load for each task as well as rate their overall cognitive load after completing all tasks. Prior research has found that ratings completed at the end of a phase, as opposed to after each task, are often slightly higher than the mean of the ratings from all tasks [11]. The cognitive load task survey required minimal time (pilot testing revealed that this is typically less than 30 seconds) for participants to complete after each task. However, due to timing constraints and concerns over participant fatigue, we decided to only ask participants to complete the CS CLCS once, at the end of each phase. Because the CS CLCS was completed at the end of the phase the results may be slightly higher compared to the per task cognitive load ratings. To account for this potential difference in the CS CLCS results, we use both the immediate cognitive load task survey ratings and the overall ratings in our analysis.

6.1.1 Training Tasks

For the training tasks, we analyzed time on task, task completion, the cognitive load surveys, and distractor usage. Because task completion may have been affected by other factors, we analyzed whether participants ran out of time or gave up on the task. We also analyzed distractor usage within the distractors condition to evaluate the effectiveness of the distractors. We analyzed the percent of participants who used the distractors in each task and the percent of time participants used distractors in each task.

6.1.2 Transfer Tasks

Similar to the training tasks, we analyzed the time each participant took to complete each transfer task and their performance for each transfer task. For the transfer tasks, we measured performance by whether the participant's solution matches the correct solution exactly; we did not score with partial credit.

6.1.3 Cognitive Load

The mental effort scales ($\alpha_{training} = .90$; $\alpha_{transfer} = .89$) and difficulty scales ($\alpha_{training} = .89$; $\alpha_{transfer} = .79$) are both reliable for both phases. The intrinsic ($\alpha_{training} = .89$; $\alpha_{transfer} = .92$), extraneous ($\alpha_{training} = .78$; $\alpha_{transfer} = .71$), and germane ($\alpha_{training} = .87$; $\alpha_{transfer} = .92$) cognitive load scales from the CS CLCS were also reliable. There is a strong correlation between the mean mental effort for all tasks and the overall mental effort $r_{training} = .82$, $p_{training} < .001$; $r_{transfer} = .80$, $p_{transfer} < .001$. Likewise, there is a strong correlation between the mean difficulty for all tasks and the overall difficulty $r_{training} = .85$, $p_{training} < .001$; $r_{transfer} = .80$, $p_{transfer} < .001$. Because these correlations are so strong, we only report the results of mental effort using ratings from each task (i.e. we ignore the overall ratings). Due to this strong correlation, we have not adjusted the results of the CS CLCS. We also see a strong correlation between mean mental effort and difficulty $r_{training} = .92$, $p_{training} < .001$; $r_{transfer} = .88$, $p_{transfer} < .001$. This suggests that mental effort and difficulty are very closely related. Because of this strong correlation, we only report the mental effort scale when discussing cognitive load.

6.2 Controlling External Effects

Our programming experience survey with follow-on interview revealed that 71% of participants had limited prior programming experience, notably many participants had used Scratch [54] or participated in an Hour of Code activity [19]. Because prior programming experience may influence the outcome of our results, we controlled for prior programming experience, as well as age, formal programming education, and gender, using covariates in our analysis. For all statistical results, where appropriate, we used ANCOVA or MANCOVA with Pillai's trace to compare the dependent variables from all six training tasks or all three transfer tasks against the control and distractor conditions.

Our analysis revealed that the age and prior experience covariates are significant for most results. In general, older participants and participants with prior experience performed better, while needing less time. The formal education and gender covariates are not significant for any results. This suggests that at the middle school level formal education may not differ much from informal learning. Researchers have also advocated for gender inclusiveness in software, including programming environments, in part because problem solving strategies tend to cluster by gender [3]. Reassuringly, we found no evidence of any gender differences.

When reporting our results, we also include effect size alongside *p* values. *P* values are heavily influenced by sample size; the larger the sample size, the more likely a statistical test will return significance ($p < .05$) [56]. Effect size removes the influence of sample size by measuring the magnitude a variable differs between conditions [56]. When reporting effect size for ANCOVA results we report omega squared (ω^2) and eta squared (η^2) for MANCOVA. We report the magnitude (small, medium, and large) based on the following values: .01, .06, and .14 [28].

7. RESULTS

We share the results of our summative evaluation in terms of how they address our research questions. However, before we report on our research questions we first verify that the distractors in our study performed as expected.

7.1 Distractor Usage

We expected participants to interact with the distractors or use the distractors when solving the training tasks. We used two measures to identify whether participants made use of the distractors: whether participants interacted with the distractors and the percent of time spent using those distractors.

Table 2. Training Task Distractors Usage

Task	% of Participants	Time Spent
1	69%	$M = 21\%$, $SD = 23\%$
2	69%	$M = 22\%$, $SD = 25\%$
3	100%	$M = 68\%$, $SD = 20\%$
4	64%	$M = 19\%$, $SD = 22\%$
5	98%	$M = 74\%$, $SD = 26\%$
6	53%	$M = 13\%$, $SD = 20\%$

All distractor participants used distractors in at least two of the training tasks (*Mdn* = 5). In fact, the majority of distractor participants (84%) used distractors in four or more of the six training tasks. When authoring multiple choice tests, researchers recommend removing infrequently chosen distractors; distractors should have a response frequency greater than 5% [14, 16]. Across each training task, a majority of participants used distractors in each task (*M* = 76%, *SD* = 19%). See Table 2 for the percentage of participants who used distractors for each task. We also note that participants spent on average 36% of their time interacting with the distractors (*SD* = 27%). Table 2 also shows the average percent of time that participants used the distractors for each training task.

In general, we believe that this evidence suggests that participants used the distractors an appropriate amount. All participants in the distractors condition used distractors and spent an average of 36% of their time during the training phase using the distractors. While the specific numbers vary across tasks and participants, we believe that spending roughly 30% of task time generating errors, is a reasonable amount of time. We are confident that the distractors worked as intended and so we spend the remainder of the results section investigating our research questions.

7.2 How do distractors affect task completion time, task success, and cognitive load?

In this section, we report how the distractors affected the training task experience for participants. Overall, we found that distractors caused participants to spend 14% more time on the training tasks, while completing 26% less tasks and increasing participants' cognitive load. See Table 3 for a summary of the training results.

7.2.1 Training Task Time

Distractor participants spent more time completing the training tasks (*M* = 6.15, *SD* = 1.00 minutes) compared to the control condition (*M* = 5.42, *SD* = 1.11 minutes). There was a significant and large effect of distractors on training task completion time $V = .17$, $F(6, 81) = 2.86$, $p < .05$, $\eta^2 = .17$.

7.2.2 Training Task Performance

Distractor participants (*M* = 65%, *SD* = 23%) correctly completed fewer training tasks than the control participants (*M* = 88%, *SD* = 11%). This effect was significant and large, $V = .30$, $F(6, 81) = 5.65$, $p < .001$, $\eta^2 = .30$. Roughly, this translates to control participants correctly completing about five tasks (*Mdn* = 6) while the distractor participants only completed close to four tasks (*Mdn* = 4).

Recall that participants were given twelve minutes to complete each training task along with the option to quit working on a task at any point. Quitting a task early is analogous to giving up on the task. Distractors participants were significantly more likely to give up on a task compared to control participants, $V = .20$, $F(6, 81) = 3.33$, $p < .01$, $\eta^2 = .20$. In fact, distractor participants quit an average of 1.8 tasks (*SD* = 2.16, *Mdn* = 1) compared to an average of 0.45 tasks (*SD* = 1.00, *Mdn* = 0) for control participants.

Table 3. Mean Training Task Results

Task	Time (min.)*		% Complete***		% Quit Early**	
	Con.	Dis.	Con.	Dis.	Con.	Dis.
1	5.17	6.43	98%	78%	2%	13%
2	4.47	5.59	96%	80%	2%	18%
3	7.30	7.90	72%	29%	11%	40%
4	5.87	6.25	83%	58%	11%	33%
5	4.22	5.79	94%	60%	2%	36%
6	5.47	4.93	77%	53%	17%	40%

*** $p < .001$, ** $p < .01$, * $p < .05$

Table 4. Mean Transfer Task Results

Transfer Task	Time (min.)		% Complete	
	Con.	Dis.	Con.	Dis.
Do Together { Repeat }	3.49	3.07	55%	49%
Repeat { Do Together }	3.58	3.08	66%	42%
Repeat { Repeat }	4.56	3.95	43%	38%

There is not a significant effect of distractors on participants running out of time while working on a task, $V = .06$, $F(6, 81) = .92$, $p = .49$, $\eta^2 = .06$. This suggests that participants in both conditions had sufficient time to complete to the training tasks. Overall, we see that distractors appear to have a negative impact on participants' ability to complete the training tasks.

7.2.3 Training Task Cognitive Load

Across all training tasks, distractor participants reported higher cognitive load than control participants. On average control participants reported their mental effort as 5.18 ≈ *neither low nor high* (*SD* = 1.67), whereas distractor participants reported their mental effort as 5.73 ≈ *rather high* (*SD* = 1.57). The effect of distractors on cognitive load is significant and large, $V = .21$, $F(6, 81) = 3.55$, $p < .01$, $\eta^2 = .21$. These results suggest that distractors increase cognitive load for learners.

The CS CLCS reveals that the distractors increased extraneous cognitive load, but not intrinsic or germane load. Intrinsic and germane cognitive load are beneficial to learning. Extraneous cognitive load is imposed by the presentation of an instructional task and it is considered harmful to learning. Distractors did not have a significant effect on intrinsic and germane cognitive load; intrinsic: $F(1, 86) = .01$, $p = .94$, $\omega^2 = .01$; germane: $F(1, 86) = .35$, $p = .56$, $\omega^2 = .01$. The effect of distractors on extraneous cognitive load is significant, but small, $F(1, 86) = 6.03$, $p < .05$, $\omega^2 = .05$.

7.3 Do distractors participants show more evidence of learning?

Lastly, we wanted to know how distractors affected participants' ability to learn programming knowledge. Overall, we found no differences in control and distractor participants' transfer task time, transfer task performance, and cognitive load. See Table 4 for a summary of the transfer task results.

7.3.1 Transfer Task Time

Control participants (*M* = 3.88, *SD* = .60 minutes) spent roughly the same amount of time working on the transfer tasks as distractor participants (*M* = 3.37, *SD* = .51 minutes). There is not a significant effect of distractors on transfer task completion time, $V = .06$, $F(3, 84) = 1.73$, $p = .17$, $\eta^2 = .06$.

7.3.2 Transfer Task Performance

Control participants (*M* = 55%, *SD* = 12%) completed roughly the same number of transfer tasks correctly as distractor participants (*M* = 43%, *SD* = 6%). There is not a significant effect of distractors on correctly completing transfer tasks, $V = .06$, $F(3, 84) = 1.78$, $p = .16$, $\eta^2 = .06$.

7.3.3 Transfer Task Cognitive Load

Across all transfer tasks control and distractor participants reported roughly the same cognitive load. On average control participants (M = 4.36, SD = 1.77) and distractor participants (M = 4.44, SD = 1.73) reported *rather low* mental effort. The effect of distractors on cognitive load is not significant, $V = .001$, $F(3, 84) = .03$, $p = .99$, $\eta^2 = .001$. Further, there is no significant difference between conditions for intrinsic, $F(1, 86) = .52$, $p = .47$, $\omega^2 = .005$, extraneous, $F(1, 86) = .63$, $p = .43$, $\omega^2 = .004$, and germane, $F(1, 86) = .02$, $p = .89$, $\omega^2 = .01$, cognitive load.

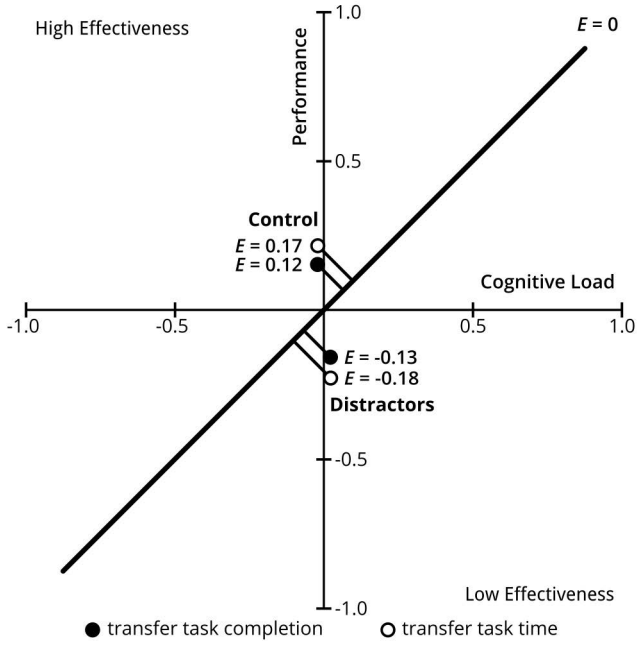

Figure 4. Instructional Efficiency (E)

7.4 Instructional Efficiency

To get an idea of how participants' cognitive load related to their performance in the transfer tasks, we computed instructional efficiency. Instructional efficiency is a measure that combines the transfer phase's cognitive load and performance in order to compare the effectiveness of instructional conditions [43, 46]. We decided to use two performance measures for computing the instructional efficiency: transfer task completion and transfer task time. Both can be considered valid measures since time on task is often related to performance [12, 46]. Figure 4 shows the instructional efficiency for our evaluation. Above the line $E = 0$ is the high effectiveness area of the graph (top-left corner); below the line $E = 0$ is the low effectiveness area (bottom-right corner). From Figure 4, we see that for both measures of performance, the control condition is more effective than the distractors condition.

8. DISCUSSION & FUTURE WORK

The results of this study suggest that distractors in code puzzles provide little benefit for novices learning programming. In this section, we highlight several questions requiring additional investigation.

8.1 Transfer Performance

Distractor participants experienced reduced learning efficiency during the training phase, while experiencing no difference in performance in the transfer phase. Based on our related work, we expected distractors to increase or decrease transfer task performance: the Parsons problem literature often hypothesizes that distractors are beneficial, [25, 47], suggesting increased transfer task performance; alternatively, based on the learning with errors and testing literature we would have expected distractors to decrease transfer task performance. However, the lack of difference in transfer performance is inconsistent with either expectation. This is an unexpected result that requires further study.

8.2 Programming Experience

Most participants had little prior exposure to the programming concepts tested in the transfer phase and on average performed poorly on all transfer tasks (control: 55%, distractors: 43%). The overall low performance suggests that participants were struggling to simply understand the basic concepts. Learners with an existing basic understanding of a programming concept may find that distractors help them transition their understanding of that concept towards mastery.

8.3 Educational Context

In this study, we specifically targeted independent learning as a context for using distractors. Yet, Parsons problems are frequently used as practice alongside traditional classroom instruction [10, 47]. While these contexts may differ, many new programmers often struggle to learn programming, even with access to classroom resources. Given the low transfer performance, it is likely that distractors may reduce learning gains for these struggling students. Future work is needed to understand how distractors affect students' learning in different educational contexts.

8.4 Motivation

The higher failure rate for completing the puzzles in the distractor condition is worrisome (mean completion: control 88%, distractors 65%). Lower success may leave users feeling deflated and less motivated to continue learning programming. Future research is necessary to understand how distractors may affect learners' motivation to continuing learning programming from code puzzles.

9. THREATS TO VALIDITY

This study looked at one type of code puzzle distractor, the *partial suboptimal path* distractor, in a very specific context: middle school children learning programming independently. There are other types of distractors and contexts that this study did not explore. For example, Parsons problems frequently contain *syntax error* distractors [25, 47]. Considering that syntax is often a significant hurdle for new programmers, future research is necessary to understand how other types of distractors affect learners.

We also note that our recruiting mailing list often includes parents who consistently seek additional learning opportunities for their children, possibly enabling selection bias towards higher performing students. Thus our results may not fully represent the wide range of abilities of middle school children.

10. CONCLUSION

The results of our study suggest that distractors in code puzzles provide no clear benefit while reducing learning efficiency. This result can better help computer science educators to understand the potential implications that using distractors in code puzzles may have on their students. Educators should use caution if they choose to use distractors in code puzzles because the result may not match their expectations. We also encourage researchers to explore and evaluate circumstances where distractors in code puzzles have a clear benefit. Understanding how distractors impact learners is just one step towards improving the effectiveness of code puzzles to better meet the needs of future programmers.

11. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1054587.

12. REFERENCES

[1] Bridger, E.K. and Mecklinger, A. 2014. Errorful and errorless learning: The impact of cue–target constraint in learning from errors. *Memory & Cognition*. 42, 6 (Mar. 2014), 898–911.

[2] Buffum, P.S. et al. 2015. A Practical Guide to Developing and Validating Computer Science Knowledge Assessments with Application to Middle School. *Proceedings of the 46th*

ACM Technical Symposium on Computer Science Education (New York, NY, USA, 2015), 622–627.

[3] Burnett, M. et al. 2010. Gender Differences and Programming Environments: Across Programming Populations. *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (New York, NY, USA, 2010), 28:1–28:10.

[4] Butler, A.C. et al. 2007. The effect of type and timing of feedback on learning from multiple-choice tests. *Journal of Experimental Psychology: Applied*. 13, 4 (2007), 273–281.

[5] Butler, A.C. et al. 2006. When additional multiple-choice lures aid versus hinder later memory. *Applied Cognitive Psychology*. 20, 7 (Nov. 2006), 941–956.

[6] Butler, A.C. and Roediger, H.L. 2008. Feedback enhances the positive effects and reduces the negative effects of multiple-choice testing. *Memory & Cognition*. 36, 3 (Apr. 2008), 604–616.

[7] Cyr, A.-A. and Anderson, N.D. 2015. Mistakes as stepping stones: Effects of errors on episodic memory among younger and older adults. *Journal of Experimental Psychology: Learning, Memory, and Cognition*. 41, 3 (2015), 841–850.

[8] Denny, P. et al. 2008. Evaluating a New Exam Question: Parsons Problems. *Proceedings of the Fourth International Workshop on Computing Education Research* (New York, NY, USA, 2008), 113–124.

[9] Elley, W.B. 1966. The Role of Errors in Learning with Feedback. *British Journal of Educational Psychology*. 36, 3 (Nov. 1966), 296–300.

[10] Ericson, B.J. et al. 2015. Analysis of Interactive Features Designed to Enhance Learning in an Ebook. *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (New York, NY, USA, 2015), 169–178.

[11] van Gog, T. et al. 2012. Timing and Frequency of Mental Effort Measurement: Evidence in Favour of Repeated Measures. *Applied Cognitive Psychology*. 26, 6 (Nov. 2012), 833–839.

[12] Gog, T. van and Paas, F. 2008. Instructional Efficiency: Revisiting the Original Construct in Educational Research. *Educational Psychologist*. 43, 1 (Jan. 2008), 16–26.

[13] Große, C.S. and Renkl, A. 2007. Finding and fixing errors in worked examples: Can this foster learning outcomes? *Learning and Instruction*. 17, 6 (Dec. 2007), 612–634.

[14] Haladyna, T.M. and Downing, S.M. 1988. Functional Distractors: Implications for Test-Item Writing and Test Design. (Apr. 1988).

[15] Haladyna, T.M. and Downing, S.M. 1993. How Many Options is Enough for a Multiple-Choice Test Item? *Educational and Psychological Measurement*. 53, 4 (Dec. 1993), 999–1010.

[16] Haladyna, T.M. and Downing, S.M. 1989. Validity of a Taxonomy of Multiple-Choice Item-Writing Rules. *Applied Measurement in Education*. 2, 1 (Jan. 1989), 51–78.

[17] Harms, K.J. et al. 2015. Enabling independent learning of programming concepts through programming completion puzzles. *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (Oct. 2015), 271–279.

[18] Helminen, J. et al. 2013. How Do Students Solve Parsons Programming Problems? – Execution-Based vs. Line-Based Feedback. *Learning and Teaching in Computing and Engineering (LaTiCE), 2013* (Mar. 2013), 55–61.

[19] Hour of Code: *http://csedweek.org/*.

[20] Huelser, B.J. and Metcalfe, J. 2011. Making related errors facilitates learning, but learners do not know it. *Memory & Cognition*. 40, 4 (Dec. 2011), 514–527.

[21] Ihantola, P. et al. 2013. How to Study Programming on Mobile Touch Devices: Interactive Python Code Exercises. *Proceedings of the 13th Koli Calling International Conference on Computing Education Research* (New York, NY, USA, 2013), 51–58.

[22] Ihantola, P. and Karavirta, V. 2011. Two-dimensional parson's puzzles: The concept, tools, and first observations. *Journal of Information Technology Education*. 10, (2011).

[23] K, H. et al. 2011. Does incorrect guessing impair fact learning? *Journal of Educational Psychology*. 103, 1 (2011), 48–59.

[24] Kalyuga, S. et al. 1999. Managing split-attention and redundancy in multimedia instruction. *Applied cognitive psychology*. 13, 4 (1999), 351–371.

[25] Karavirta, V. et al. 2012. A Mobile Learning Application for Parsons Problems with Automatic Feedback. *Proceedings of the 12th Koli Calling International Conference on Computing Education Research* (New York, NY, USA, 2012), 11–18.

[26] Karpicke, J.D. 2012. Retrieval-Based Learning Active Retrieval Promotes Meaningful Learning. *Current Directions in Psychological Science*. 21, 3 (Jun. 2012), 157–163.

[27] King, K.V. et al. 2004. The distractor rationale taxonomy: Enhancing multiple-choice items in reading and mathematics. *Assessment Report. Pearson*. (2004).

[28] Kirk, R.E. 1996. Practical Significance: A Concept Whose Time Has Come. *Educational and Psychological Measurement*. 56, 5 (Oct. 1996), 746–759.

[29] Kornell, N. et al. 2009. Unsuccessful retrieval attempts enhance subsequent learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition*. 35, 4 (2009), 989–998.

[30] Lee, M.J. and Ko, A.J. 2015. Comparing the Effectiveness of Online Learning Approaches on CS1 Learning Outcomes. *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (New York, NY, USA, 2015), 237–246.

[31] Leppink, J. et al. 2013. Development of an instrument for measuring different types of cognitive load. *Behavior Research Methods*. 45, 4 (Apr. 2013), 1058–1072.

[32] Leppink, J. et al. 2014. Effects of pairs of problems and examples on task performance and different types of cognitive load. *Learning and Instruction*. 30, (Apr. 2014), 32–42.

[33] Lin, J. et al. 2010. Distractor rationale taxonomy: Diagnostic assessment of reading with ordered multiple-choice items. *American Educational Research Association, Denver, CO*. (2010).

[34] Little, J.L. and Bjork, E.L. 2012. The persisting benefits of using multiple-choice tests as learning events. *Proceedings of the 34th Annual Conference of the Cognitive Science Society* (2012), 683–688.

[35] Looking Glass: *http://lookingglass.wustl.edu/*.

[36] Marsh, E. and Cantor, A. 2014. Learning from the Test: Dos and Don'ts for Using Multiple-Choice Tests. *Integrating Cognitive Science with Innovative Teaching in STEM Disciplines*. (Sep. 2014).

[37] Marsh, E.J. et al. 2009. Memorial consequences of answering SAT II questions. *Journal of Experimental Psychology: Applied*. 15, 1 (2009), 1–11.

[38] Marsh, E.J. et al. 2007. The memorial consequences of multiple-choice testing. *Psychonomic Bulletin & Review*. 14, 2 (Apr. 2007), 194–199.

[39] Mkrtchyan, A. 2011. Distractor Quality Analyze In Multiple Choice Questions Based On Information Retrieval Model. *EDULEARN11 Proceedings*. (2011), 1624–1631.

[40] Morrison, B.B. et al. 2014. Measuring Cognitive Load in Introductory CS: Adaptation of an Instrument. *Proceedings of the Tenth Annual Conference on International Computing Education Research* (New York, NY, USA, 2014), 131–138.

[41] Morrison, B.B. et al. 2016. Subgoals Help Students Solve Parsons Problems. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (New York, NY, USA, 2016), 42–47.

[42] Muller, D. a. et al. 2008. Saying the wrong thing: improving learning with multimedia by including misconceptions. *Journal of Computer Assisted Learning*. 24, 2 (Apr. 2008), 144–155.

[43] Paas, F. and Van Merrienboer, J. 1994. Variability of worked examples and transfer of geometrical problem-solving skills: A cognitive-load approach. *Journal of Educational Psychology*. 86, 1 (1994), 122–133.

[44] Paas, F.G. 1992. Training strategies for attaining transfer of problem-solving skill in statistics: A cognitive-load approach. *Journal of Educational Psychology*. 84, 4 (1992), 429–434.

[45] Paas, F.G.W.C. et al. 1994. Measurement of cognitive load in instructional research. *Perceptual and Motor Skills*. 79, 1 (Aug. 1994), 419–430.

[46] Paas, F.G.W.C. and Merriënboer, J.J.G.V. 1993. The Efficiency of Instructional Conditions: An Approach to Combine Mental Effort and Performance Measures. *Human Factors: The Journal of the Human Factors and Ergonomics Society*. 35, 4 (Dec. 1993), 737–743.

[47] Parsons, D. and Haden, P. 2006. Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52* (Darlinghurst, Australia, Australia, 2006), 157–163.

[48] Potts, R. and Shanks, D.R. 2014. The benefit of generating errors during learning. *Journal of Experimental Psychology: General*. 143, 2 (2014), 644–667.

[49] Richland, L.E. et al. 2009. The pretesting effect: Do unsuccessful retrieval attempts enhance learning? *Journal of Experimental Psychology: Applied*. 15, 3 (2009), 243–257.

[50] Rodriguez, M.C. et al. 2014. Distractor Functioning in Modified Items for Test Accessibility. *SAGE Open*. 4, 4 (Oct. 2014), 2158244014553586.

[51] Roediger, H.L. and Karpicke, J.D. 2006. Test-Enhanced Learning Taking Memory Tests Improves Long-Term Retention. *Psychological Science*. 17, 3 (Mar. 2006), 249–255.

[52] Roediger III, H.L. and Butler, A.C. 2011. The critical role of retrieval practice in long-term retention. *Trends in Cognitive Sciences*. 15, 1 (Jan. 2011), 20–27.

[53] Roediger III, H.L. and Marsh, E.J. 2005. The Positive and Negative Consequences of Multiple-Choice Testing. *Journal of Experimental Psychology: Learning, Memory, and Cognition*. 31, 5 (2005), 1155–1159.

[54] Scratch: *https://scratch.mit.edu/*.

[55] Smith, A.M. et al. 2012. A Case Study of Expressively Constrainable Level Design Automation Tools for a Puzzle Game. *Proceedings of the International Conference on the Foundations of Digital Games* (New York, NY, USA, 2012), 156–163.

[56] Sullivan, G.M. and Feinn, R. 2012. Using Effect Size—or Why the P Value Is Not Enough. *Journal of Graduate Medical Education*. 4, 3 (Sep. 2012), 279–282.

[57] Tarrant, M. et al. 2009. An assessment of functioning and non-functioning distractors in multiple-choice questions: a descriptive analysis. *BMC Medical Education*. 9, (2009), 40.

[58] Tobias, S. et al. 2014. Game-Based Learning. *Handbook of Research on Educational Communications and Technology*. J.M. Spector et al., eds. Springer New York. 485–503.

[59] Van Merriënboer, J.J.G. 1990. Strategies for Programming Instruction in High School: Program Completion vs. Program Generation. *Journal of Educational Computing Research*. 6, 3 (Jan. 1990), 265–285.

[60] Van Merrienboer, J.J.G. and De Croock, M.B.M. 1992. Strategies for Computer-Based Programming Instruction: Program Completion Vs. Program Generation. *Journal of Educational Computing Research*. 8, 3 (Jan. 1992), 365–394.

[61] Wang, M. et al. 2015. Using Feedback to Improve Learning: Differentiating between Correct and Erroneous Examples. *2015 International Symposium on Educational Technology (ISET)* (Jul. 2015), 99–103.

[62] Williams, E. 1949. Experimental Designs Balanced for the Estimation of Residual Effects of Treatments. *Australian Journal of Chemistry*. 2, 2 (Jan. 1949), 149–168.