SAGE Field Study:

Parson’s Programming Puzzle Efficiency

Andrew Gorovoy (apg2165)

Jeffrey Fabian (jf2978)

COMS 3998, Section 28

Spring 2019

Table of Contents

INTRODUCTION

In 2006, writing as the head of the Computer Science Department at Carnegie Mellon University, Jeanette Wing articulated a call for educational action that releases her subject from the constraints of its scholarly seclusion. Stipulating that ubiquitous computing is to today as computational thinking (CT) is to tomorrow, she frames CT as a "universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use." (Wing, 2006). Although a consensus definition for the term has remained elusive, CT aims to describe how humans, not computers, think logically, algorithmically, procedurally, analytically, recursively, and creatively, with an engineering-attunement. This hybrid thinking helps us devise models and representations which enable problem solving, data transformation, and system automation. Meanwhile, computing's proliferation has empowered students to explore digital worlds interactively, enabling passion-driven discovery that engages the minds that schools too frequently fail to attract or satiate. Educators increasingly acknowledge that game-based learning (GBL) activates student motivation and improves outcomes, but teachers and game designers typically train in divergent dialects (Prensky, 2003).

In order to infuse CT for students in grades 6-8, a learning system must simultaneously address the needs of schools, teachers, and students. We put forth that these requirements are best realized by the SAGE learning system. Its foundational principles include: (1) Social, (2) Addictive, (3) Gameful, and (4) Engineering. Since modern problem-solving occurs in communities rather than in isolation, developing *social* agility while encountering CT concepts fosters a positive attitude to "failure" as learning unfolds, and encourages not only experimentation and collective construction of computational models, but also a culture of model critique that questions underlying assumptions and identifies pitfalls. Likewise, *addicting* students to learning is a desirable goal aligned with enabling students to enter a flow state when learning. That is, the clear sense of required action, intense concentration, continuous feedback and sense of intrinsic reward -- that often constitute a flow state -- would enable students to practice CT with the attentiveness required to develop deep understandings (Nakamura & Csikszentmihalyi, 2014). When CT concepts, practices, and perspectives meld within game contexts, the learning becomes *gameful*: goal-oriented, curiosity-driven, failure-fearless, optimistic, and fun. Instead of overlaying the game cycle, CT content blends throughout the environment as an intrinsic component of game-play. Finally, exploring multiple routes while solving problems engenders *engineering* habits of mind and mental processes that require important aspects of CT. This engineering problem solving becomes fun in GBL since a game is, as game designer Jesse Schell reminds us, "a problem-solving activity, approached with a playful attitude." (Schell, J., 2014).

As a result, SAGE transforms Scratch into a gameful intelligent tutoring system. Scratch-based Parsons Programming Puzzles and Code Completion instruction styles, alongside a dynamic scoring system, provides students with a continuous stream of feedback and support. For the present study, the SAGE learning system combines the aforementioned concepts into a cohesive environment that offers a strong foundation to the students interacting with the system. As per the purpose of this study, we seek to expose the learning system to both teachers and students in order to not only systematically gauge the efficiency of Parson’s Programming Puzzles, but also identify potential pain points to guide further development of the SAGE system.

1. *Literature Review*

1.1. Parson’s Programming Puzzles

Parsons programming puzzles (PPP) are a type of code practice problem in which the learner must place blocks of mixed up program code in the correct order (Parsons & Haden, 2008). PPP empower teachers to impart gameful interactive learning to students by designing targeted puzzles themselves. It is often the case that students solving PPP get real time feedback with every move they make and are guided towards the solution, making it both more convenient and helpful in the classroom (Ihantola &

Karavirta, 2011). Typically, PPP are built on the idea of a drag-and-drop concept where a student puts the block with the code piece in logical order by dragging and dropping at the correct order. In conjunction with the SAGE learning system built on Scratch, we bring PPP to a block-based programming environment primarily conducive to teaching students in grades 6-8.

With regards to the field study at hand, we seek to iterate on the promising results from prior studies that demonstrate PPP as a fun and effective instructional style that enables both students and teachers to learn and teach computational thinking more efficiently. Contrary to prior studies though, we seek to address the research gap in gauging the efficiency of PPP all within the block-based Scratch environment -- particularly targeting what makes PPP more efficient than a generic code completion (scaffolding) task.

1.2. Cognitive Load Theory

Cognitive Load Theory was developed by John Sweller in the late 1980s (Sweller, 1994). For learning to occur new information must be processed in working memory and then added to the knowledge representations (schemas) that exists in long-term memory (Sweller,2005). However, working memory has a limited capacity, and if that capacity is needed entirely to process new information, it cannot be used to modify or build schemas. Instructional material can be designed to reduce the cognitive load that is devoted to processing new information. It is important to note that the amount of cognitive load a learner experiences is based on three components: the complexity of the material or task, the way the instruction is designed, as well as the strategies used for constructing knowledge. The complexity of the material or task varies with the learner's prior knowledge. Parsons problems, as a type of code completion problem, should have a lower cognitive load than a problem that requires the learner to write the code from scratch, because the problem space is more constrained.

In addressing instructional efficiency for the experimental conditions we call upon cognitive load theory to conceptualize whether or not tasks do not over-complicate or introduce external difficulties, but rather facilitate the desirable difficulty that deep understanding requires.

1.3. Pretesting Effect

Pretests are regularly used as assessments in pre–posttest design studies with the expectation that they do not affect learning. There were some reasons, however, to expect that pretesting could enhance learning. Many studies have demonstrated benefits of pre-training activities such as advanced organizers, outlines , and statements to activate learners' prior knowledge schemas. Test questions have also been studied as pre training activities, beginning with early experiments on the effects of integrating adjunct questions into text passages. Adjunct questions interwoven into texts, both before and after the target information had been provided, showed improved retention of information asked about inthe question and, less reliably, information not asked about (Richland, Kornell & Kao, 2009).

With respect to the study at hand, we ensure that students are generally making progress and learning the material underlying our comparison of efficiency between each condition.

FORMATIVE EVALUATION

In order to formally develop our instructional design theory, we outline our rationale and overarching purpose of the current study. The aim of the present study is to gauge the efficiency of Parsons Programming Puzzles in the SAGE environment -- considering whether students learn more in the same amount of time or learn the same amount in less time using Parsons Programming Puzzles over other instructional styles. In addition, we seek to pilot the current SAGE programming environment and ensure that the learning system effectively infuses Computational Thinking concepts during instruction and identify pain points that will guide further development of the system.

1. Initial Exposure

This section could be devoted to a conducted formative evaluation -- that is, an informal pilot that exposes students to this study in a quick-and-dirty format. As with most formative evaluations, we would then outline any major thoughts or learnings from it here.

SUMMATIVE EVALUATION

The objective of this study is to gauge the efficiency and efficacy of using SAGE Parsons Puzzles to teach computational thinking concepts when compared to other methods of teaching and examination including building programs from scratch in Scratch or completing a partial solution to a programming exercise. The study will focus on analyzing the efficiency and efficacy of teaching computational thinking; these concepts including data management, control flow and sequencing, iteration, and input and output are independent of any specific programming language, and to keep the study independent of specific syntax and semantics, the field study will be conducted through the use of visual block programming language Scratch developed by the MIT Media Lab. Efficiency will be gauged by the minimization of cognitive load, maximization of correctness of activity solutions, and total time needed to complete given curriculum. This measurements will be administered through digital surveys and platform interaction tracking. The population segment involved in the study includes middle school students (6th - 8th grade) of varying previous programming/computer science experience.

1. Design

In order to compare and manipulate instructional styles, we introduce three distinct conditions: (1) SAGE Scratch, (2) SAGE Code Completion, and (3) SAGE Parsons Programming Puzzles. Conditions 1-3 are meant to progressively increase instructional efficiency with respect to both cognitive load and student performance dimensions.

1.1. Conditions

We expect the Scratch condition to show the lowest levels of instructional efficiency and the PPP to show the highest due to the reduction in learning barriers inherent to using a Palette as opposed to every block. Likewise, the Code Completion condition's utilization of a scaffolding would implicitly constrain the number of blocks students would need to consider. This result would be in accordance with prior findings that demonstrate Parsons' greater efficiency over code writing (Ericson, Margulieux, & Rick, 2017) and lowering intrinsic/extraneous cognitive load (Ericson, Foley, & Rick, 2018).

Despite the apparent similarities between the Code Completion and PPP conditions, we expect PPP to outperform Code Completion primarily because of the multitude of findings that demonstrate the effectiveness of Parsons' Puzzles in promoting student engagement (Parsons & Haden, 2008). As a result, we expect students to be particularly receptive to the material, have a deeper understanding of the material and thus increase levels of germane load. Since we will keep task instructions consistent throughout conditions and pre-emptively familiarize students with each problem format, we expect no significant difference in extraneous load.

1.2. Student Performance

Measurement will be calculated using performance on the puzzles and cognitive load score. The performance on the puzzle is calculated by summing the time of completion and correctness (score out of 100) across the entire sample of puzzles for that student.

We expect the Scratch condition to show the lowest levels of instructional efficiency and the PPP to show the highest due to the reduction in learning barriers inherent to using a Palette as opposed to every block. Likewise, the Code Completion condition's utilization of a scaffolding would implicitly constrain the number of blocks students would need to consider. This result would be in accordance with prior findings that demonstrate Parsons' greater efficiency over code writing (Ericson, Margulieux, & Rick, 2017).

1.3. Cognitive Load

Student cognitive load (CL) can be partitioned into three types namely, intrinsic, extraneous, germane load. Ideally and according to Cognitive Load Theory, minimizing innate and extraneous load and maximizing germane load would promote the most learning. For the sake of simplicity in our analysis, "Good" cognitive load will be derived as the germane load score, and "Bad " cognitive load score will be derived as the extraneous/innate load score.

1.4. Instructional Efficiency

In order to represent differences in instructional efficiency, we call to prior research in Cognitive Load Theory. Partitioned into three types -- intrinsic, extraneous and germane load -- we seek to minimize intrinsic and extraneous load and maximize germane load in order to promote the most learning (Sweller,2010). In accordance with CLT, the worked example effect also states that students learn best given a detailed description and/or demonstration of a problem alongside interleaved practice problems (Sweller,2005).

As a result, worked examples and practice problems offer a framework by which we conduct our learning material and enable us to measure student performance. Student performance, represented by the time spent and correctness of a student's submission, together with student cognitive load constitute our measurement for instructional efficiency that we seek to optimize.

1.5. Phases

After the pre-examination, students will be introduced to the mechanics of the tasks and the programming environment and correct ways to navigate the environment and tasks. This will be done through a series of powerpoint slides and screencasts demonstrating the SAGE environment and the tasks respective to the experimental conditions.

1.6. Materials

Throughout the study, students were exposed to a number of field study materials including surveys, programming puzzles, slideshow presentations, and examinations. These materials were distributed in the form of Google surveys or through the SAGE platform. Links to these items were sent through an email list, and accessed through student email accounts.

For the familiarization phase, students were exposed to a presentation that served as a tutorial of the SAGE programming environment. This presentation introduced location and expected behavior of buttons and features of the platform. In addition, it introduced students to the three conditions used in the study for the puzzles types. The condition introduction explains game mechanics and expected protocol for each puzzle type. For example, the idea of the palette is explained for Parsons Puzzle and the basic code scaffolding is explained for Code Completion puzzles. A link to the presentation can be found in the materials appendix. During this phase, students were also required to take a survey that reviews demographics and prior programming experience.

For the activity phase, students were required to complete pre/post examinations, puzzles respective to their randomly assigned condition, and cognitive load surveys. The pre/post examinations are designed to measure the students baseline knowledge of computational thinking. The tests contain three types of questions that mimic the same mechanics in the activity puzzles. These include tracing code, completing code to execute a desired behavior, and selecting a palette of blocks that would allow for the construction of a program with desired behavior. When students were completing the puzzles, they did so using the SAGE platform. The SAGE platform provided instructions for each puzzle. The puzzles focus on basic computational topics primarily including control flow and iteration.

It is important to emphasize that the objective of the study is to gauge the efficiency of Parsons Puzzle when compared to other puzzle types in teaching computational thinking. Therefore, the actual topics being taught are not of critical importance. We selected mentioned topics due to their foundational importance in code design and development. The puzzle is a combination of instructions, example behavior, game objectives, and game theme. Puzzles were designed initially on the basis of a computational thinking topic. This was then followed by choosing some theme such as Minecraft or Ice Age that would serve as a medium for the game and selected computational topic. For example, for the first puzzle, students focused on control flow in a puzzle that had a Minecraft theme. Each of the puzzles across conditions shared the same theme and concept, but differed in game mechanics. The instructions also included a video/GIF of the desired end behavior to provide some guidance for the students if the instructions are not clear enough or the student is experiencing some confusion. Students were then expected to attempt to solve their given puzzle in the allotted time. After students completed each of the puzzles, they completed a cognitive load survey. This survey was provided as a link that appears once students press the submit button. Students, after completing all three test phase puzzles respective to their condition, then completed a post-test. The pretest and post-test were isomorphic in nature. Only variable names, variable values, and string content was changed;

difficulty level and concepts tested were consistent across the examinations. Links to these materials can be found in the Materials appendix.

In the final phase of the study, students completed a post-test identical to the post-test completed in the prior phase. After, students were then administered an open-ended feedback survey. This survey was designed with the intent to collect general criticisms and potential improvements about the SAGE platform. No reportable data was collected during this phase. Links to the mentioned materials can be found in the Materials Appendix.

All students were asked to take pre-examinations in order to identify a baseline level of computational thinking competency. The tested concepts include variables, control flow, and iteration. The examinations were administered in the form of Google surveys that students took on laptops or desktop computers. The question types mimic the type of questions and question format students were tasked with in later parts of the study. These include code tracing; selecting a single block to complete a code block in order to achieve some desired output; and selecting a palette of blocks that would allow for the completion of a simple program with described behaviors and outputs.

Students after completing the study then completed a post-examination. The post-examination was administered and distributed following identical protocol as the pre-examination. The pre and post examinations were isomorphic in nature. Variable names and values will be changed, but the underlying concept and difficulty level will be identical.

2. Methods

2.1. Participants

The population segment for the study will focus on middle school students (6th-8th grade) from the Manhattan, New York area. These students have varying levels of prior programming experience. To account for this, a demographics and prior programming experience survey was administered prior to the start of the programming section of the field study. Results from this survey were then used to gauge quality of data and justify the potential removal of participant's data from final analysis if necessary.

NOTE OUR **RESULTS** AND **ANALYSIS** SECTIONS SIMPLY DESCRIBE POTENTIAL OPTIONS GIVEN THE FIELD STUDY HAS YET TO BE CONDUCTED

3. Results

3.1. Demographics

This section would be dedicated to giving the reader a sense of the number of participants, their age, sex, gender, levels of prior programming experience, etc. that actually participated in the study.

3.2. Pre-Test & Post-Test

This section would be dedicated to the raw results of the pre-test, immediate post-test and (potentially) the delayed post-test.

3.3. Instructional Efficiency

Likewise, this section would be dedicated to the raw results of the student performance and cognitive load as well either a composite score for IE or a graphical representation of that data.

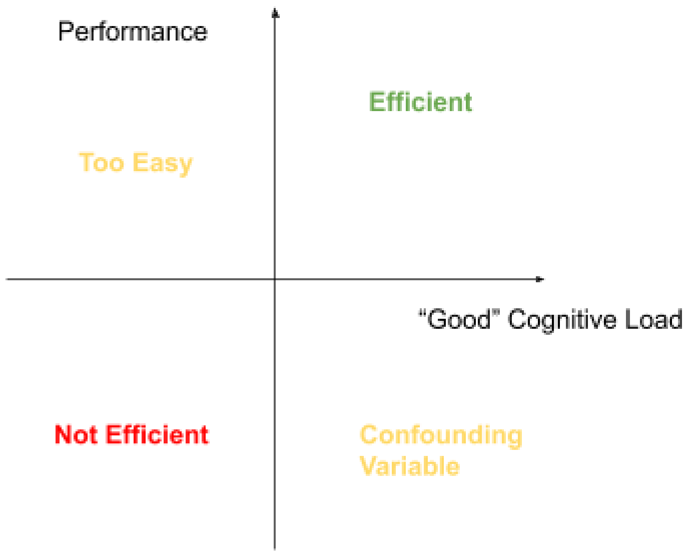
4. Analysis

4.1. Baseline Learning Levels

Measuring differences across the pre and post tests would be relatively straightforward

4.2. Efficiency

The general breakdown of our proposed instructional efficiency analysis includes each student datum composed of their condition, performance (time and correctness) and levels of good/bad cognitive load. From there, we would compare efficiency between conditions (between-subjects) according to the following two options: **(A)** Combine CL and performance into a composite score for each student and compare scores across conditions (e.g. ANOVA) **(B)** use a graphical categorization graph that plots student data according to the two dimensions and measure student clusters according to where they fall on the graph



DISCUSSION

1. Threats to Validity

Three types of validity were considered during the design of this experiment: ecological, internal, external.

This field study was led and facilitated by the teacher of the classroom. This was a point of discussion during preliminary designs of this study. When designing a study, an isolated environment with the ability to only focus on the manipulated variable is prized; however, when observing the efficiency and efficacy of educational tools, it is impossible to remove the “noise” from the classroom environment. We believe our study is ecologically valid and takes into account teachers days to day interruptions, issues, and experiences. With the teacher facilitating the study in a normal classroom environment, we believe this is the most realistic simulation of the application of SAGE in the classroom. Results derived from this study should be reflective of application of SAGE in other classrooms as the experimental environment was highly realistic and comparable to real world uses of SAGE outside of an experimental environment.

Internal validity refers specifically to whether an experimental treatment/condition makes a difference to the outcome or not, and whether there is sufficient evidence to substantiate the claim. The following are possible threats to internal validity and experimental design solutions to counteract these threats:

- ❏ **History**: Timespan of an experiment can introduce confounding variables. The field study is done over the span of 3 consecutive days and then a follow up session a week later. The consecutive sessions allow control for any lurking variables to be introduced into the experiment. The follow up a week later tests for retention of information, however is at greater risk of being affected by items unconnected to the study and falsey influencing results.

- ❏ **Testing**: Students may memorize test questions leading to false results when asked to redo tests. Therefore, the pretest and post-tests are isomorphic in nature only differing in surface level aspects such as variable names that will make memoization of questions difficult.
- ❏ **Statistical Regression and Floor Effect**: If students have no experience at all in computer science and block programming language, the familiarization phase and activity phase may lead to no improvements at all. This is known as the floor effect, and will invalidate our study. This was accounted for with a prior programming experience survey and computational concepts that are foundational and straight forward in nature.

- ❑ **Selection of subjects**: Since this in some institutions will be run as a volunteer study, the participant pool may be biased in experience and approach to computer science and computational thinking. Once again, this will be handled through the use of prior programming experience survey that will provide evidence for any adjustments or invalidations of data.

External validity refers to the generalizability of the treatment/condition outcomes across various settings. The following are possible threats to external validity and experimental design solutions to counteract these threats:

- ❑ **Reactive or interaction effect of testing**: A pretest might increase or decrease a subject's responsiveness to the experimental variable. The effect of pretest to subsequent tests has been correlated previously (Wilson & Putnam, 1982, Lana, 1959). This potential threat was accounted for by having multiple conditions , and comparing the change in score between tests across experimental conditions. The actual value to the score on a test is not critical in this field study.

APPENDIX

1. Materials

Pretests:

- Iteration pretest
- Control flow pretest

Post-tests:

- Iteration post-test
- Control flow post-test

Email Templates:

- Familiarization
- Pretest
- Post-test
- Follow Up

Familiarization Presentation:

- Slides

Surveys:

- Demographics and prior experience
- Cognitive load survey
- Open ended feedback survey
- SAGE teacher survey

Instruction Manual

- Document

REFERENCES

Ericson, B. J., Foley, J. D., & Rick, J. (2018, August). Evaluating the Efficiency and Effectiveness of Adaptive Parsons Problems. In Proceedings of the 2018 ACM Conference on International Computing Education Research (pp. 60-68). ACM.

Ericson, B. J., Margulieux, L. E., & Rick, J. (2017, November). Solving parsons problems versus fixing and writing code. In Proceedings of the 17th Koli Calling International Conference on Computing Education Research (pp. 20-29). ACM.

Ihantola, P., & Karavirta, V. (2011). Two-dimensional parson’s puzzles: The concept, tools, and first observations. Journal of Information Technology Education, 10, 119-132.

Lana, R. E. (1959). A further investigation of the pretest-treatment interaction effect. Journal of Applied Psychology, 43(6), 421.

Nakamura, J., & Csikszentmihalyi, M. (2014). The concept of flow. In Flow and the foundations of positive psychology (pp. 239-263). Springer, Dordrecht.

Parsons, D., & Haden, P. (2006, January). Parson's programming puzzles: a fun and effective learning tool for first programming courses. In Proceedings of the 8th Australasian Conference on Computing Education-Volume 52 (pp. 157-163). Australian Computer Society, Inc..

Prensky, M. (2003). Digital game-based learning. Computers in Entertainment (CIE), 1(1), 21-21.

Putnam, L. L., & Wilson, C. E. (1982). Communicative strategies in organizational conflicts: Reliability and validity of a measurement scale. Annals of the International Communication Association, 6(1), 629-652.

Richland, L. E., Kornell, N., & Kao, L. S. (2009). The pretesting effect: Do unsuccessful retrieval attempts enhance learning?. Journal of Experimental Psychology: Applied, 15(3), 243.

Schell, J. (2014). The Art of Game Design: A book of lenses. AK Peters/CRC Press.

Sweller, J. (2005). Implications of cognitive load theory for multimedia learning. The Cambridge handbook of multimedia learning, 19-30.

Sweller, J. (2010). Element interactivity and intrinsic, extraneous, and germane cognitive load. Educational psychology review, 22(2), 123-138.

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. Learning and instruction, 4(4), 295-312.

Wing, J. M. (2006). Computational thinking. Communications of the ACM, 49(3), 33-35.