Runestone Interactive Ebooks: A Research Platform for On-line Computer Science Learning

Barbara J. Ericson
barbarer@umich.edu
University of Michigan
School of Information
Ann Arbor, Michigan

Iman YeckehZaare
oneweb@umich.edu
University of Michigan
School of Information
Ann Arbor, MI

Mark J. Guzdial
mjguz@umich.edu
University of Michigan
Computer Science and Engineering
Ann Arbor, Michigan

ABSTRACT

The Runestone ebook platform is open source, extensible, and already serves over 25,000 learners a day. The site currently hosts 18 free ebooks for computing courses. Instructors can create a custom course from any of the existing ebooks on the site and can have their students register for that custom course. Instructors can create assignments from the existing material in each ebook, grade assignments, and visualize student progress. Instructors can even create new content for assignments. The Runestone ebooks contain instructional material and a variety of practice problem types with immediate feedback. One of the practice types, Parsons problems, is also adaptive, which means that the difficulty of the problem is based on the learner's performance. Learner interaction is recorded and can be analyzed. This paper presents the history of Runestone, describes the interactive features, summarizes the previous research studies, and provides detail on the recorded data. Interaction data can be shared with other learning environments through the Learning Tools Interoperability Standard (LTI).

CCS CONCEPTS

• **Social and professional topics → Informal education**; **Student assessment**; **K-12 education**; **Adult education**.

KEYWORDS

on-line learning, adaptive learning, practice tools, intelligent ebooks

ACM Reference Format:
Barbara J. Ericson, Iman YeckehZaare, and Mark J. Guzdial. 2019. Runestone Interactive Ebooks: A Research Platform for On-line Computer Science Learning. In *ICER '19: ACM International Computing Education Research, August 12–14, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Brad Miller created the Runestone ebook platform in 2011 during his sabbatical when he was supposed to be working on a new edition of two paper textbooks. He was having writer's block because he did not like the idea of using paper textbooks for computer science. He wanted to have electronic books (ebooks) with embedded code that the learner could run and edit. Brad also wanted ebooks to be free so that everyone has the chance to learn computer science, even if they can not afford a $200 textbook. Brad's goal was textit"democratizing textbooks for the 21st century".

Brad started looking for a way to run Python in the browser. He found Skulpt (skuplt.org), an in-browser implementation of Python. He built a Logo-style turtle graphics [17] module for Skulpt , but then realized that authors would not want to write JavaScript for every example. He started looking for a document system that would enable authors to easily write ebooks with executable and editable code. Brad found Sphinx, an extensible tool written in Python for creating beautiful documentation. Documents are written in reStructuredText (rst), which is a very minimal markup language that is similar to Markdown. Sphinx contains built-in *directives* that let you easily add images, figures, notes and more. To add an extension to Sphinx you create a new *directive*, which can have arguments, options, and content. Brad created a new directive in Sphinx to allow authors to easily author executable and editable code examples in Python. Brad used the web2py web framework since it supported database-driven web applications.

Over the years many people have joined the project and contributed many new directives (features). This paper will describe the features and what data is recorded for each and summarize the research studies.

2 RUNESTONE FEATURES

Runestone supports instructional features like text, images, videos, executable/editable code, audio tours, a code visualiser/stepper (Python Tutor), and expression evaluation. It also supports practice questions that provide immediate feedback: multiple-choice, fill-in-the-blank, drag-and-drop, clickable code, clickable table item, and Parsons problems. Runestone also includes a couple of question types that do not provide immediate feedback, but record the learner's answer: short-answer and poll questions. A timed exam feature can be used for timed assessments. One of new newer features of Runestone is a practice tool that instructors can use to encourage spaced, interleaved, and retrieval-based practice. The following subsections describe Runestone's features and the type of data that is recorded during usage of that feature.

2.1 Executable/Editable Code

The *activecode* directive is used to add executable and editable code as shown in Figure 1. If the code is Python, it is run as JavaScript on the client using Skulpt. If the code is in JavaScript or HTML it is

ICER '19, August 12–14, 2019, Toronto, ON, Canada

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

run directly in the browser. If the code is in Java or C++ the code is executed on a server.

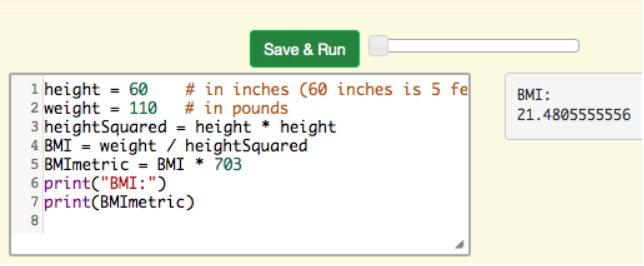

Figure 1: Executed Python code with output.

Runestone supports both Logo style turtle graphics [17] as shown in Figure 2 and image manipulation in both Python and Java as executable and editable code as shown in Figure 3. The image manipulation exercises are from the media computation approach in which learners write programs that modify media: images, sounds, videos, and text [11][10]. Media computation has improved student pass rates and engagement [12][22][20].

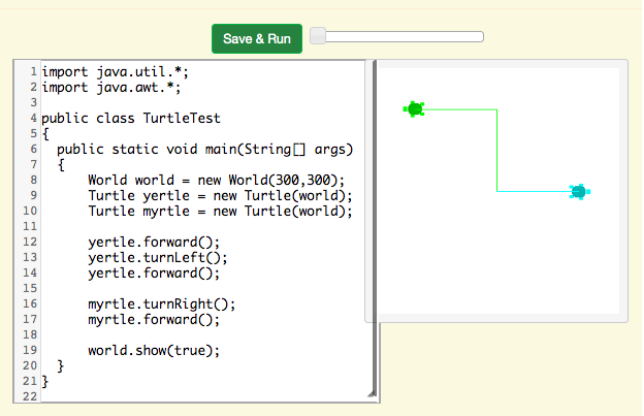

Figure 2: A turtle graphics example in Java.

The *activecode* directive can optionally include up to five audio tours associated with it. In an audio tour, one or more lines of code are highlighted as the audio plays as shown in Figure 4. This feature takes advantage of the fact that humans have dual channels which allow simultaneous processing of visual and audio information [14][15].

2.2 Code Visualizer/Stepper

A version of Philip Guo's Python Tutor can be added to Python ebooks using the *codelens* directive[9] as shown in Figure 5. It allows learners to step through the code line by line and displays the current value of all variables. Authors can also embed a question that will be asked before a specified line is executed. Questions can have students predict which line will execute next or the value of a variable after the next line executes.

Figure 3: A media computation example in Python.

Figure 4: An audio tour highlights one or more lines of codes as the audio plays that explains those lines.

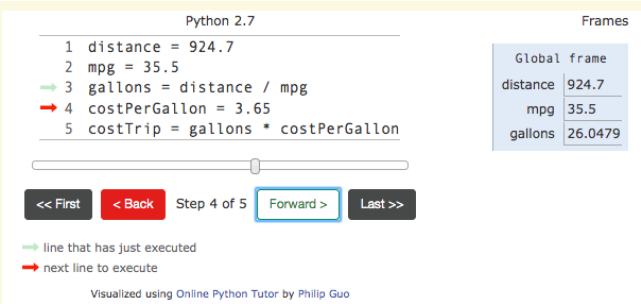

Figure 5: An example of the code visualizer/stepper (Python Tutor).

Runestone does not currently support embedded visualizers for other languages, such as Java, but authors can add a link to an external visualizer, such as the Java Tutor, which can display preloaded code.

2.3 Expression Evaluation

Expression evaluation is added using the *showeval* directive. It shows the next step in the evaluation of an expression after each

click of the *Next Step* button as shown in Figure 6. It displays the current expression on the next line and then animates replacing part of that expression with the result of the evaluation.

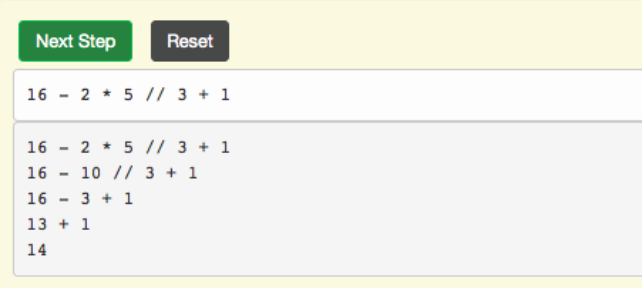

Figure 6: An example visualization of the evaluation of a complex expression.

2.4 Multiple-Choice Questions

Runestone supports multiple-choice questions with one or more required answers with the *mchoice* directive. Immediate feedback is provided based on the answer or answers selected when the *Check Me* button is clicked.

2.5 Fill-in-the-Blank Questions

A *fillintheblank* directive is used to allow users to type an answer to a question. Feedback is displayed based on the regular expression that matches the answer when the *Check Me* button is clicked.

2.6 Drag and Drop Questions

The *dragndrop* directive allows learners to drag a definition to the matching concept as shown in Figure 7. Feedback is displayed below the question when the *Check Me* button is clicked.

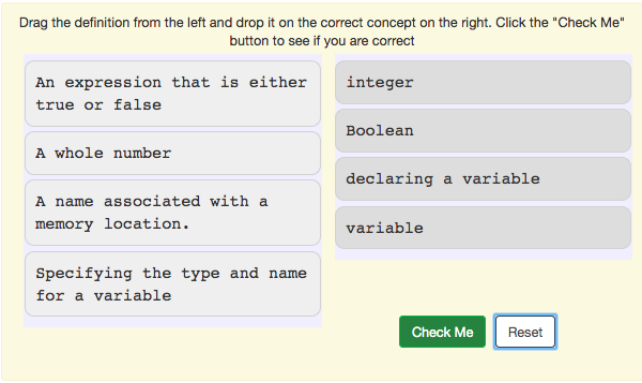

Figure 7: An example of a drag and drop question where the user drags the definition to the matching concept.

2.7 Clickable Code

Authors can add clickable code questions using the *clickablearea* directive. In clickable code the user clicks on one or more lines of code to answer a question. Each line that the user clicks is highlighted in yellow. The user clicks the *Check Me* button to submit an answer. The correct lines that were clicked remain highlighted in yellow and any incorrect lines that were clicked are boxed in red as shown in Figure 8. Textual feedback is displayed under the question as shown in Figure 8.

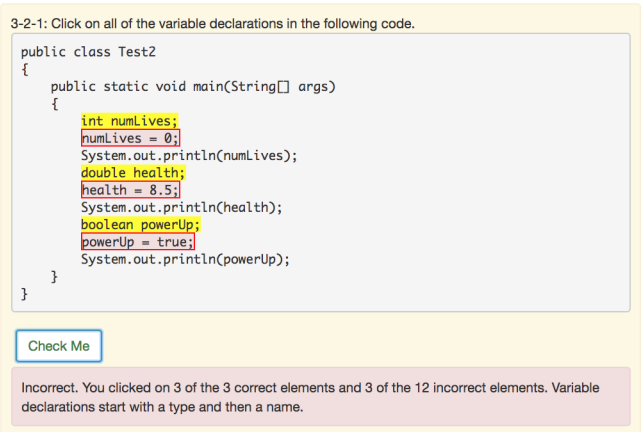

Figure 8: A clickable code question that shows the correct answers still highlighted in yellow and the incorrect answers boxed in red.

2.8 Clickable Table Item

Authors can add clickable table items using the *clickablearea* directive with a table. Learners click on one or more table item to answer a question as shown in Figure 9.

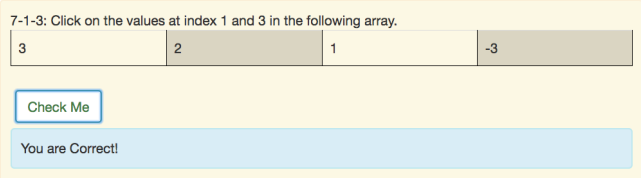

Figure 9: A clickable table item question that asks the user to click on the array values at index 1 and 3.

2.9 Short Answer

A *shortanswer* directive allows the learner to type in a textual response to a question a shown in Figure 10. The only feedback is that the answer has been saved.

2.10 Poll

The *poll* directive allows the author to create a poll question as shown in Figure 11. The only feedback is that the answer has been recorded.

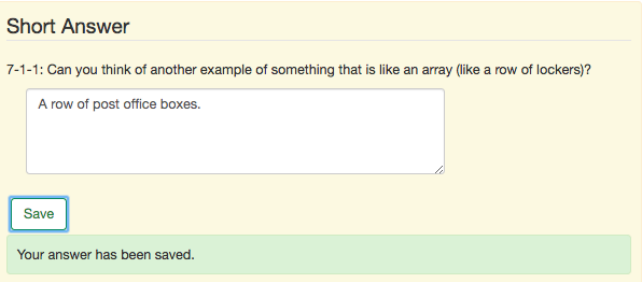

Figure 10: A short answer question that allows the user to type in text to answer a question. The text is saved to the server.

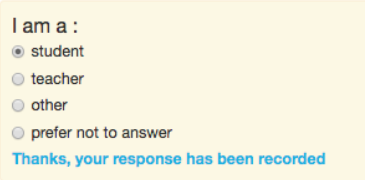

Figure 11: An example poll question. The answer is saved on the server.

2.11 Tabbed Panel

The *tabbed* directive creates a tabbed panel that can contain several tabs. Each tab is created with the *tab* directive. Each tab can contain content. In the example shown in Figure 12 the learner is asked to write the code to solve a problem, but there is also a tab with the answer and another tab for a discussion area. The learner clicks on a tab to display the content in that tab.

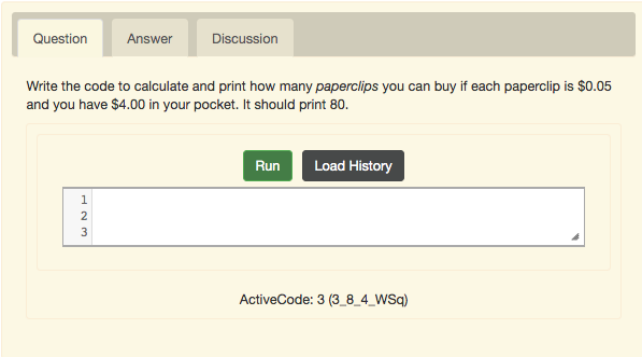

Figure 12: An example tabbed panel with three tabs. One tab contains the question, one the answer, and one the discussion.

2.12 Parsons Problems

Parsons problems are mixed up code problems where the learner places the code in the correct order to solve a problem [19]. They can be added using the *parsons* directive.

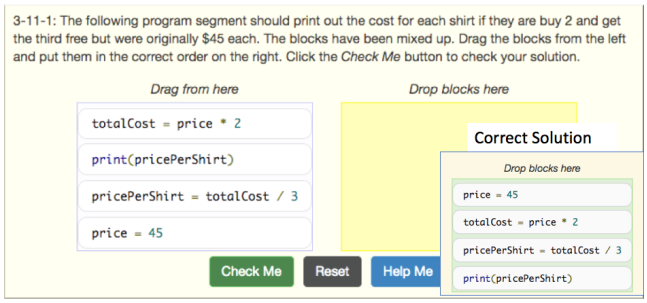

Figure 13: An example Parsons problem in Python. The mixed-up blocks are shown on the left and the correct solution is shown on the right.

Parsons problems can optionally display labels to support collaborative work as shown in Figure 14. These labels are also useful when Parsons problems are used in paper-based exams [3][2]. On a paper-based exam the learner can simply write the block labels in the correct order. This reduces the cognitive load for test takers since they do not have to generate a solution. Paper-based Parsons problems are faster to grade than the equivalent code writing problem and the grading is more consistent [2]. Scores on Parsons problems have correlated with scores on code writing problems [3][2].

Parsons problems can also contain *distractors*, which are extra code blocks that are not needed in a correct solution. Distractors can either be randomly mixed in with the correct code or shown paired with the corresponding correct code as shown in Figure 14. Some Parsons problems also require the learner to indent the code correctly. These are called two-dimensional Parsons problems. Some languages, like Python, require indentation to specify the code structure such as which statements are in the body of a loop.

Ericson invented two types of adaptation for Parsons problems [5]. In intra-problem adaptation if the learner is struggling to solve the current problem it can dynamically be made easier by disabling distractor blocks, providing indentation, or combining blocks. In inter-problem adaptation if the learner solved the last problem in just one attempt the next problem can be made harder by randomly mixing all distractors in with the correct code blocks. If the learner took many attempts to solve the last problem the next problem can be made easier by pairing the correct and distactor code or removing distractors.

2.13 Timed exams

Authors can add timed exams to ebooks using the *timed* directive. These are intended for summative assessment and can contain any of the practice question types such as multiple-choice questions, fill-in-the-blank questions, and Parsons problems. A timed exam can have a time limit or can just display the amount of time the user has spent on the exam. The learner must click the Start button to start a timed exam and then one question is displayed at a time as shown in figure Figure 15. The user can use the question number buttons to jump to a particular question.

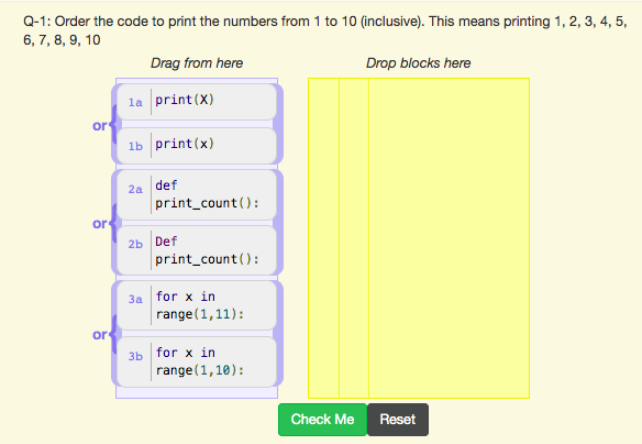

Figure 14: An example Parsons problem in Python. The mixed-up blocks are shown on the left and the correct solution is shown on the right.

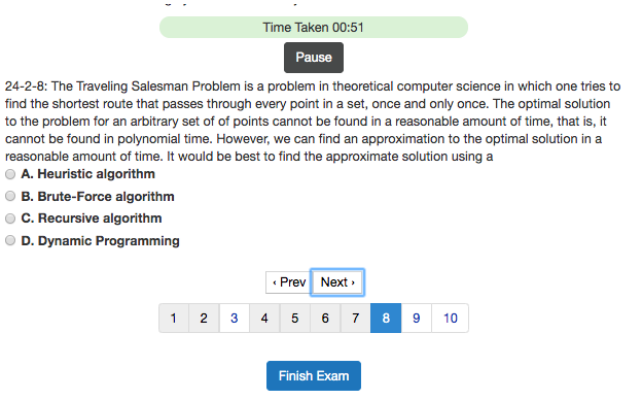

Figure 15: An example timed exam. Notice that only one question is displayed at a time.

3 SUMMARY OF RESEARCH STUDIES

The next sections summarize the research studies that have been conducted with Runestone ebooks.

3.1 First Ebook Trial

Brad and his colleague David Ranum at Luther College created an interactive version of the *How to Think Like a Computer Scientist* ebook in 2011. Their ebook contained text, images, videos, executable and editable Python code, and a code visualizer (Philip Guo's Python Tutor). Brad and David trialed the interactive version of the this ebook in the fall of 2011 with 66 undergraduate students and gathered feedback in a survey. The survey provided evidence that the students were positive about the ebook. Sixty-one (90%) of the students said that they would want to use a similar ebook in another course [16].

3.2 Studies by CSLearning4U

The CSLearning4U group applied findings from educational psychology to interactive ebooks. In particular the group used worked examples [21] plus practice [23] and multiple-modalities (audio tours) [14]. One goal was to make learning more efficient to meet the needs of busy in-service secondary teachers. The group first modified the *How to Think Like a Computer Scientist* ebook and later created a teacher and student version of an ebook for the Advanced Placement (AP) Computer Science Principles (CSP) course. This course is offered in secondary schools and is intended to be equivalent to a college level course for non-majors. It covers programming fundamentals including variables, loops, conditionals, and functions.

The group conducted a usability survey with 18 teachers comparing several interactive features (code execution, code visualisation, Parsons problems, and multiple choice) on three ebook platforms (Runestone, Zyante, and CS Circles) [4]. The majority of the participants preferred the Runestone platform interface.

A log file analysis of usage of the *How to Think Like a Computer Scientist* ebook found that more users attempted the Parsons problems than nearby multiple-choice questions, that the number of users declined from the start to the end of each chapter, and that users were more likely to attempt practice problems than watch videos, listen to audio tours, or edit code [6]. While most users were able to solve the Parsons problems in just one or two attempts, there were users who took between 20 and 100 attempts to solve each problem, and some users gave up and never solved them.

Ten teachers worked through the first eight chapters of the AP CSP ebook at their own pace [4]. Five of the ten teachers completed the first eight chapters for a 50% completion rate. This is higher than typical MOOC completion rates, however the teachers received a $50 gift card for completing the chapters. The teachers who had the highest scores on the post-tests also used the interactive features the most [4].

In a large-scale study with 130 teachers using version two of the teacher ebook we found that teachers valued the worked example plus practice approach, Parsons problems, and having lots of practice problems [8]. Many teachers did not edit code or listen to audio tours. However, some teachers reported that they found audio tours useful for demonstrating how to describe code. Teachers reported that using the ebook increased their confidence in their ability to teach the content.

In a log file analysis of the teacher and student versions of the AP CSP ebook we found evidence that teachers use the ebook differently than students [18]. Students ran the same program multiple times without changing it and made many more failed attempts to solve problems. Teachers may be more aware of successful and unsuccessful learning strategies.

3.3 Parsons Problems Research

Ericson used Runestone to study the efficiency and effectiveness of solving Parsons problems versus fixing code and versus writing the equivalent code [7]. Undergraduate students solved Parsons problems significantly more quickly than fixing or writing the equivalent code and with comparable learning gains from pretest to post-test.

Ericson also tested the efficiency and effectiveness of solving adaptive Parsons problems, versus non-adaptive Parsons problems, and versus writing the equivalent code [5]. Again, undergraduate students solved both types of Parsons problems significantly more quickly than writing the equivalent code. The adaptive Parsons problem group had significantly higher learning gains than the control group.

3.4 Practice Tool Research

YeckehZaare created and tested an adaptive practice tool on the Runestone platform [24] based on the theory of desirable difficulties [1]. Desirable difficulties are those that improve long-term retention, but reduce short-term learning gains. One example is spacing practice over many days rather than studying the night before an exam. Another is interleaving different topics rather than studying just one topic at a time. A third is forcing yourself to retrieve an answer to a question rather than just rereading the question and answer. The practice tool provides spaced, interleaved, and retrieval practice by reusing the practice questions in an ebook and presenting a question on a topic just before an algorithm (a modified version of SuperMemo 2) predicts that the learner is likely to forget that topic. One question is shown at a time as shown in Figure 16. Students had to answer at least ten questions correctly during a practice session to earn a point and earned the maximum number of points by practicing at least 45 days over a semester.

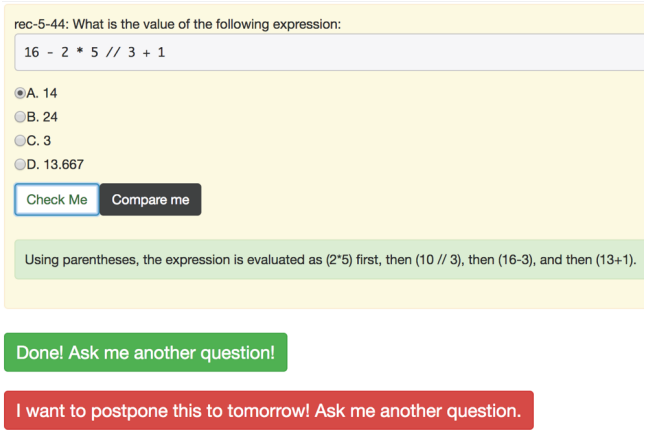

Figure 16: The practice tool interface.

Students tend to prefer techniques that improve short-term learning over those that improve long-term retention and recall [13]. However, in a study with 193 undergraduate students we found that every hour of using the practice tool was correlated with an increase in the average final exam grades of 1.04%[24]. Students reported finding the tool useful and motivating. In fact, 62 (32%) of the students used the practice tool more than 45 days, even though they didn't earn any additional points for doing so, which indicates that they found the tool valuable.

4 RECORDED DATA

Runestone records ebook interaction data and this data can be retrieved from a log file. Instructors can download a log file from the instructor's page. The log includes the date and time of the interaction (timestamp), the user id (sid), the directive (event), and additional information (act) based on the event type as shown in Figure 17. The log also includes a unique id (div_id) for each problem in an ebook, the course id (course_id), and the short name for the ebook (base_course).

	id	timestamp	sid	event	act	div_id	course_id	base_course
476	473	17879	8/21/17 21:38	18	mChoice	answer:1:correct	q2_2_2	12	apcsareview
477	474	17881	8/21/17 21:38	18	mChoice	answer:0:correct	q2_2_3	12	apcsareview
478	475	17882	8/21/17 21:38	18	page	view	JavaBasics/f	12	apcsareview
479	476	17884	8/21/17 21:38	18	activecode	edit	lcfc1	12	apcsareview
480	477	17885	8/21/17 21:39	18	parsonsMove	start|6_0-5_0-4_0-0-2_3_0-0_1_0|-|c0	thirdClass	12	apcsareview
481	478	17886	8/21/17 21:39	18	parsonsMove	move|6_0-5_0-4_0-0_1_0|2_3_0|c0	thirdClass	12	apcsareview
482	479	17888	8/21/17 21:39	18	parsonsMove	move|6_0-5_0-4_0|0_1_0-2_3_0|c0	thirdClass	12	apcsareview
483	480	17889	8/21/17 21:39	17	page	view	JavaBasics/i	12	apcsareview
484	481	17890	8/21/17 21:39	18	parsonsMove	move|6_0-5_0|0_1_0-2_3_0-4_0|c0	thirdClass	12	apcsareview
485	482	17891	8/21/17 21:39	18	parsonsMove	move|6_0|0_1_0-2_3_0-4_0-5_0|c0	thirdClass	12	apcsareview
486	483	17892	8/21/17 21:39	18	parsonsMove	move|-|0_1_0-2_3_0-4_0-5_0-6_0|c0	thirdClass	12	apcsareview
487	484	17894	8/21/17 21:39	18	parsons	correct|-|0_1_0-2_3_0-4_0-5_0-6_0|c1-s	thirdClass	12	apcsareview
488	485	17895	8/21/17 21:39	18	parsonsMove	start|10_0-6_7_0-4_5_0-0_1_0-2_3_0-8_0-9_0-11_0|-|c0	fourthClass	12	apcsareview
489	486	17896	8/21/17 21:40	18	parsonsMove	move|10_0-6_7_0-4_5_0-2_3_0-8_0-9_0-11_0|0_1_0|c0	fourthClass	12	apcsareview
490	487	17897	8/21/17 21:40	18	parsonsMove	move|10_0-6_7_0-2_3_0-8_0-9_0-11_0|0_1_0-4_5_0|c0	fourthClass	12	apcsareview
491	488	17899	8/21/17 21:40	18	parsonsMove	move|10_0-6_7_0-2_3_0-9_0-11_0|0_1_0-4_5_0-8_0|c0	fourthClass	12	apcsareview
492	489	17900	8/21/17 21:40	18	parsonsMove	move|6_7_0-2_3_0-9_0-11_0|0_1_0-4_5_0-8_0-10_0|c0	fourthClass	12	apcsareview
493	490	17901	8/21/17 21:40	18	parsonsMove	move|6_7_0-2_3_0-9_0|0_1_0-4_5_0-8_0-10_0-11_0|c0	fourthClass	12	apcsareview
494	491	17902	8/21/17 21:40	18	parsons	correct|6_7_0-2_3_0-9_0|0_1_0-4_5_0-8_0-10_0-11_0|c1-s	fourthClass	12	apcsareview
495	492	17903	8/21/17 21:41	17	video	play	ants	12	apcsareview

Figure 17: Part of a log file from Runestone that has been anonymized

The log contains a record of each page load, code execution, code edit, video play, audio tour play, and practice question answer. All code that is executed or saved is available as well, but it does not appear in the standard log file. You have to request the code from Brad Miller who runs the Runestone server. The log also records the answers to each of the practice questions and whether that answer was correct as shown in Figure 17 for the multiple-choice questions (mchoice) and Parsons problems (parsonsprob).

The log contains extensive data on Parsons problems. It records every move of a block as well as the contents the source area on the left and the solution area on the right as shown in 17. For example, the first Parsons problem entry with a unique problem id of "thirdClass" has "start|6_05_0-4_0-2_3_0-0_1_0|-|c0" in the act column. The "start" means that the user has started the problem. The first "|" starts the information about the source area on the left and the second "|" starts the information about the solution area on the right. The numbers before the last "_" are the line numbers from the Parsons problem source and the number after the last "_" is the level of indentation. Zero means that the block is not indented. Parsons problems start with all of the code blocks in the source area on the left and the user drags the blocks to the solution area on the right. The "-" means that the solution area on the right does not contain any blocks. The state when the problem is correct is "correct|-|0_1_0-2_3_0-4_0-5_0-6_0|c1-s" which shows that all of the code blocks have been put in the correct order in the solution area on the right side without any indentation. Code blocks can contain more than one line of code as shown in the second example with a unique problem id of "fourthclass". This problem has several lines in one block as shown by the start state "start|10_0-6_7_0-4_5_0-0_1_0-2_3_0-8_0-9_0-11_0|-|c0". It has line 10 in one block and then lines 6 and 7 together in the next block.

ACKNOWLEDGMENTS

We thank Brad Miller for his work developing and enhancing the Runestone platform, the people who have contributed features to Runestone, and the students and teachers who have used ebooks on the Runestone platform.

REFERENCES

[1] Elizabeth L Bjork, Robert A Bjork, et al. 2011. Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning. *Psychology and the real world: Essays illustrating fundamental contributions to society* 2, 59-68 (2011).

[2] Nick Cheng and Brian Harrington. 2017. The Code Mangler: Evaluating Coding Ability Without Writing any Code. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 123–128.

[3] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. 2008. Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth international workshop on computing education research*. ACM, 113–124.

[4] Barbara Ericson, Mark Guzdial, Briana Morrison, Miranda Parker, Matthew Moldavan, and Lekha Surasani. 2015. An eBook for teachers learning CS principles. *ACM Inroads* 6, 4 (2015), 84–86.

[5] Barbara J. Ericson, James D. Foley, and Jochen Rick. 2018. Evaluating the Efficiency and Effectiveness of Adaptive Parsons Problems. In *Proceedings of the 2018 ACM Conference on International Computing Education Research (ICER '18)*. ACM, New York, NY, USA, 60–68. https://doi.org/10.1145/3230977.3231000

[6] Barbara J Ericson, Mark J Guzdial, and Briana B Morrison. 2015. Analysis of interactive features designed to enhance learning in an ebook. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*. ACM, 169–178.

[7] Barbara J Ericson, Lauren E Margulieux, and Jochen Rick. 2017. Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th Koli Calling Conference on Computing Education Research*. ACM, 20–29.

[8] Barbara J Ericson, Kantwon Rogers, Miranda Parker, Briana Morrison, and Mark Guzdial. 2016. Identifying design principles for CS teacher Ebooks through design-based research. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ACM, 191–200.

[9] Philip J Guo. 2013. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 579–584.

[10] Mark Guzdial and Barbara Ericson. 2007. *Introduction to computing & programming in Java: a multimedia approach*. Pearson Prentice Hall.

[11] Mark Guzdial and Barbara Ericson. 2016. *Introduction to computing and programming in python*. Pearson.

[12] Mark Guzdial and Allison Elliott Tew. 2006. Imagineering inauthentic legitimate peripheral participation: an instructional design approach for motivating computing education. In *Proceedings of the second international workshop on Computing education research*. ACM, 51–58.

[13] Nate Kornell and Robert A Bjork. 2009. A stability bias in human memory: Overestimating remembering and underestimating learning. *Journal of experimental psychology: General* 138, 4 (2009), 449.

[14] Richard E Mayer. 2008. Applying the science of learning: Evidence-based principles for the design of multimedia instruction. *American psychologist* 63, 8 (2008), 760.

[15] Richard E Mayer and Roxana Moreno. 1998. A split-attention effect in multimedia learning: Evidence for dual processing systems in working memory. *Journal of educational psychology* 90, 2 (1998), 312.

[16] Bradley N Miller and David L Ranum. 2012. Beyond PDF and ePub: toward an interactive textbook. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*. ACM, 150–155.

[17] Seymour Papert. 1980. *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc.

[18] Miranda C Parker, Kantwon Rogers, Barbara J Ericson, and Mark Guzdial. 2017. Students and Teachers Use An Online AP CS Principles EBook Differently: Teacher Behavior Consistent with Expert Learners. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. ACM, 101–109.

[19] Dale Parsons and Patricia Haden. 2006. 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*. Australian Computer Society, Inc., 157–163.

[20] Leo Porter and Beth Simon. 2013. Retaining nearly one-third more majors with a trio of instructional best practices in CS1. In *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 165–170.

[21] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science* 12, 2 (1988), 257–285.

[22] Allison Elliott Tew, Charles Fowler, and Mark Guzdial. 2005. Tracking an innovation in introductory CS education from a research university to a two-year college. In *ACM SIGCSE Bulletin*, Vol. 37. ACM, 416–420.

[23] John Gregory Trafton and Brian J Reiser. 1994. *The contributions of studying examples and solving problems to skill acquisition*. Ph.D. Dissertation. Citeseer.

[24] Iman YeckehZaare, Paul Resnick, and Barbara Ericson. 2019. A Spaced, Interleaved Retrieval Practice Tool that is Motivating and Effective. In *Proceedings of the International Computing Education Research Conference (ICER '19), August 12–14, 2019, Toronto, ON, Canada*. ACM.