Final project report

SAGE Gameful Direct Instruction and Gameful Constructionism: Parson's Puzzle

Veronica Woldehanna | vtw2108 | Spring 2019 COMS 3998

Table of Contents

Abstract

In this paper, a new method for scoring Parsons problems is presented. Furthermore, improvements are made to the automatic feedback given to students. This project is geared toward guiding students and helping them build their solutions by focusing on how they work.

Introduction

SAGE extends scratch, a drag and drop based programming language that allows users to develop their own games, animations and puzzles, and gives instructors the ability to author puzzles and students to gamefully solve them as they learn computational concepts (Bender, 2015). Parson's programming puzzles in particular is a type of game that allow students to reorder blocks of code to get the desired solution (Parsons & Haden, 2006). Parson's puzzle was created to avoid the problem of having syntactical issues block a student from learning concepts, to make rote learning of the syntax fun, to constrain the logic of a potential solution so students are guided to the solution and are kept from being side tracked, and to make it engaging for students to learn.

Parson's puzzles in SAGE (Mohan, 2017) allows the instructor to author the puzzle, incorporates "Self-Explanations" that help reinforce learning, animates sprites to indicate success or failure, has a live scoring system, and allows the teacher to differentiate between student performances and assess mastery of a concept.

As mentioned, Parsons Puzzles has live scoring and immediate feedback for the purposes of guiding a student to a solution without extraneous cognitive load. The current scoring system is based on a linear comparison of a student's code to a model solution each time a student makes a move. With each move, starting from top of the model solution, lines are compared one by one and points are deducted from their score for any out-of-order lines. This, however, forces students to work linearly going against the way experts are known to program. In addition, it is possible for a solution that is very similar to the model solution, but has one line missing, to get a score equal to if not worse than a score given to a solution that is completely out of order. This also means they continue to get negative feedback after one wrong move regardless of how correct the other moves are relative to each other. This project makes improvement to the scoring system to avoid the constraints mentioned and produce a score that is indicative of how close a student is to the model solution. It also contributes to properly guiding students by making improvements to the feedback system.

Related Works

Epplets, a Java web application for solving Parsons Puzzles, gives as feedback the first line of code in the student solution that is out of order as compared to the model solution. This was done for the purpose of helping the student solve the puzzle one line at a time (Kumar, 2018). This is similar to the implementation in SAGE scratch except that Epplets only provides feedback once the student solution is submitted. This way it avoids the problem we have encountered in SAGE's Parsons puzzles, which is inaccurate feedback, as it points out the first line of code that is incorrect and makes no comment on the rest of the lines. However, this has been known to also encourage students to solve the puzzle in a linear manner (Helminen et al, 2012). Just as in SAGE's version, however Epplets takes in to account the number of actions the student took to assemble the lines of code in to the score.

It can also be observed that students don't ask or use automated feedback either maybe because they are too proud or they don't know when they are stuck (Ihantola & Karavitra, 2011). Ihantola and Karavitra suggest that constant feedback is the best way to tackle this problem as opposed to having a student ask for it by clicking a button or submitting their solution. With this approach however, problems like trial and error behavior in students, inaccurate feedback, and constraints on the puzzle solving approach that students must take (i.e. linear) arise.

Karavitra et al (2018), have come up with a way to provide feedback to students based on the longest common subsequence (LCS), the longest sequence shared by the model and student solution. They highlight the lines in the student solution that do not belong to the LCS indicating to students to modify only the highlighted portion of their solution. To prevent students from using this feedback system to solve the puzzle in an undesired manner (using trial and error), they disable the button for a time based on how frequently and successively a student has used it. This prevents students from abusing said button.

In an effort to make it more engaging, Karavitra et al also suggest assigning scores to solutions and/or awarding students for achievements like taking less time to solve the puzzle or using less feedback. This is already implemented in SAGE Parsons puzzles and in fact students get a penalty for every extra move they make. Since we give immediate feedback, this has a huge part to play in preventing trial and error, as their score is related to how they solve the problem.

Lastly, ViLLE (Rajala, Laakso, Kaila, & Salakoski, 2007) is a Java application/applet whose version of Parsons Puzzle uses error messages from the execution of a student's code as feedback. A student can also walk through visualization of their code execution. Code visualization is already built into SAGE scratch and auto-execution, a feature implemented last semester, allows the student to visualize their code automatically with every step. However, the feedback given in SAGE's Parsons Puzzle does not come from these partial-code executions.

Design and Implementation

In this section we provide the design and implementation of the improved scoring and feedback system.

Scoring system

The goal is to move away from having students build their solutions linearly and allow them to be able to work on different areas of their solution without getting negative feedback. To allow this, a scoring system that accounts for multiple segments of code in the ScriptsPane s implemented. This algorithm is similar to the LCS algorithm mentioned but finds the LCS across multiple segments in the scripts pane.

To differentiate between lines of code that have errors ranging from few to numerous, a heuristic based on Manhattan distance was integrated into the score. The goal is to award a high score to sequences of lines that could become an LCS with only a few modifications and a low score to lines of code that had a long way to go before resembling any area of the model solution and so are completely out of order. In addition, if students correctly constructed an area of the solution made of blocks with higher point values, they'd get a higher reward as the scoring system also considers each line's point values, which are set by the instructor during time of design.

In the previous implementation of the scoring system, for each out of order line the student would get a point deduction equal to the point value of the correct line as determined by the model solution. In the new system, they would still get this deduction but the point value of the correct line is now scaled(multiplied) by how far away the correct line is in the student's solution from where it should've been as determined, again, by the model solution. This guarantees that the less out of order a student's solution is, the smaller the point deductions and vice versa.

In the end, only one segment actually determines a student's score and that is the segment that most resembles a certain area of the model solution. To find this segment, each segment is aligned with the model solution for maximum score and the segment with the maximum score is used to determine the student's score. This amounts to guessing which part of the solution the student is working on and scoring them based on how correctly they've implemented that part.

Of course, when students decide to merge their code segments together to eventually produce one segment that is their solution, assuming that the separate segments are correct, their scores will increase. This happens because when computing the score for each segment, there is logic to account for the fact that the segment is still only a part of the solution and that it is still unfinished even if it is correct so far. This way as students build up more and more of their solution, if they merge segments in the correct order, their scores increase. If not, if they merge the segments together out of order, then their scores might decrease as the area of the model solution they are currently working on could be more out of order as measured by the weighted Manhattan distance heuristic solution.

Of course, this is a somewhat complicated scoring system and might not be as transparent to the students which leads to a need for explicit feedback.

Feedback system

Previously feedback indicated that either the student had a correct solution, an incorrect solution, or that they had just used a distractor. So as mentioned above, if a student was building a solution that closely resembled an area of the model solution as long as they had at least one line out of order, they would get negative feedback from that point on. To fix this, the new feedback was implemented to be specific to the move that was just made rather than taking in to account a student's entire solution.

This feedback indicates whether a move they just made is incorrect, correct, or neutral relative to the other lines of code that are around it. Students are made aware if they have just used a distractor line as in the previous system. To implement this, the only lines of code that are checked are the lines before and after the current line of consideration. If those happen to be correct according to the model solution then the student receives positive feedback, if they are incorrect the student receives negative feedback, if a student starts a new segment then the student receives a neutral feedback.

In addition, as a way to provide more advanced feedback we allow them to compare the run of their solution with the model solution and so make it easier for them to adjust their solutions. This is done by adding a button that will let them run the model solution whenever they wish. This will allow them to notice where the undesired behavior is happening and so give them an idea of what they need to fix. These visualizations can help a novel learner reason about the program execution helping them develop a skill that has been known to be a major challenge in introductory programming education (Sorva, 2012). It allows them to learn the debugging skills that are essential to any programmer as well as build their solution with the end goal in mind.

Bug fixes, Failed attempts, and Limitations

Pausing scoring

pausing scoring, a feature that was considered but was later decided against, was intended to allow the student to work and explore without any kind of feedback and consequently allow the student to work non-linearly. However, this was removed as a better solution was found for the problem it was intended to solve. The new scoring system is better in that students will continue to get feedback which avoids increasing the cognitive load of students.

Watermark of sprite

Since the execution for any solution happens very fast, having a watermark of the sprite in the initial location of the sprite would help students provide a frame of reference for the final location of the sprite according to the student and model solutions. Unfortunately, this was not implemented as adding the UI component for the watermarks proved to be taking more time than there was. Hence this is instead listed as an item under the future works section.

Bug fixes

This semester's work has also included cleaning up bugs present in Sage-Scratch, Sage-Node, and Sage-Frontend for the purposes of having everything ready for field study. This list includes preventing model solution from being shown to the student, fixing pages/data that weren't loading properly, removing palette/block restriction in Parsons Puzzle, and fixing glitches in adding/removing blocks from Parsons palette.

Future work

More advanced feedback

As a solution to the problem of encouraging students to solve the puzzle in a linear manner, Helminen et al (2012) suggest highlighting the fragments of code that are out of order and informing the student that these are wrong relative to each other. Karavitra et al (2018) also highlight the lines that are not in LCS to provide more specific feedback.

Although we have the penalties in place for students who exhibit trial and error behavior, it might be more beneficial to also add to our explicit feedback that the lines are wrong only in relation to the blocks around it and perhaps also highlight the rest of the out of order lines in the student's solution. By integrating this kind of hint into our feedback system we could more efficiently point students in the right direction and avoid them having to break apart parts of their solution they have correctly implemented. We have the ability to not only highlight lines of code that aren't in the LCS but also lines of code that are in the LCS but are out of order.

Point scaling

As per the implementation of the new scoring system, because each point deduction is equal to the point value of the correct line of code scaled by how far away that line of code is in the student solution, scores can increase very rapidly. On the one hand, this might make the puzzle more Gameful to students but on the other hand, huge increases and decreases in score might be confusing to the students. Therefore, the deductions should be scaled by some amount deemed appropriate for Parsons puzzles.

More specific feedback

Due to time constraints, the current feedback system only considers the move a student has just made in relation to the model solution but not the solution the student already had. For example, a student can move a correct line to another correct position, perhaps in another segment and get a positive feedback even though the student did not make a difference on how close they are to the model solution. For that case, it would be better to consider what the surrounding lines of the line in question are, not just in the model solution vs current state of the student's solution but also the previous state of the student's solution.

Conclusion

The focus of this semester has been on guiding students to the correct solution by taking in to account how students work and trying to decrease their cognitive load as they work through the puzzle. In this paper, problems to the current implementations of the scoring and feedback system have been explained and the design and implementation of the new scoring and feedback system have been presented. The new systems now avoid the problem of inaccurate feedback, constraining students to solve the puzzle in a linear manner, and inaccurate scores.

References

Bender, J. (2015). Developing a collaborative Game-Based Learning System to infuse computational thinking within Grade 6-8 Curricula.

J. Helminen, P. Ihantola, V. Karavitra, and L. Malmi. How Do Students Solve Parsons Programming Problems? – An Analysis of Interaction Traces. In Proceedings of the Eighth Annual International Computing Education Research Conference, pages 119–126, 2012.

J. Sorva. Visual Program simulation in introductory programming education. Doctoral dissertation, Department of Computer Science and Engineering, Aalto University, 2012.

Karavitra V., Helminen J., Ihantola P. (2018). A Mobile Learning Application for Parsons Problems with Automatic Feedback.

Kumar, Amruth. (2018). Epplets: A Tool for Solving Parsons Puzzles. 527-532. 10.1145/3159450.3159576.

P. Ihantola and V. Karavitra. Two-Dimensional Parson's Puzzles: The Concept, Tools, and First Observations. Journal of Information Technology Education: Innovations in Practice, 10:1–14, 2011.

Rajala, T., Laakso, M.-J., Kaila, E., & Salakoski, T. (2007). ViLLE — A language-independent program visualization tool. In R. Lister & Simon (Eds.), Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007), Vol. 88 of CRPIT, ACS, Koli National Park, Finland, pp. 151–159.