Final project report

SAGE Gameful Direct Instruction: Parson's Puzzle

Veronica Woldehanna | vtw2108 | Fall 2018

COMS 3998

Table of Contents

Abstract ***3***

Introduction ***4***

Second of half of the semester **4**

First half of the semester **5**

Related works ***7***

Features ***8***

Step-explanations **8**

Feedback generation from peers **9**

Instructor feedback **11**

Auto-initialization **12**

Auto-execution **13**

Green-flag automation **13**

Implementation and Architecture ***14***

Step-explanation **14**

Feedback generation from peers **14**

Instructor feedback **15**

Auto-execution **15**

Green-flag as default block **15**

Auto-initialization **15**

Bug Fixes **16**

ParsonsLogic 16

Limitations ***17***

Wrong explanations **17**

Speed of auto-execution and auto-initialization **17**

Green Flag in Play Mode **17**

Future Work ***19***

Fix for fast speed of auto-execution and auto-initialization **19**

Filtering feedback from students and requesting feedback from instructors for specific mistakes **19**

Timing of "get advice from peers" button **19**

Conclusion ***20***

References ***21***

Abstract

This paper explains the goals and results of development on the Gameful Direct Instruction epic for the SAGE project. The focus throughout the semester was on Parson's Puzzle. During the first half of the semester, effort was put into incorporating visual programming with parson's puzzles and ameliorating possible student misconceptions of initialization in Scratch. The second half of the semester was focused on improving student coaching by providing helpful error feedback in parson's puzzles, student collaboration, and reinforcing learning.

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 scoring system that takes in to account not just the final solution but how students get there, and will allow the teacher to differentiate between student performances and assess mastery of a concept.

Second of half of the semester

The focus in the second half of the semester was to improve student coaching in Parson's puzzles. In some cases, giving students feedback of why their solution is incorrect is important in helping them learn. Currently Parson's Puzzles in SAGE gives students immediate feedback when they drag and drop a block from the palette on to the scripts pane. But this feedback is limited to telling them whether the step they just made is right or wrong. If it's wrong and a student is unable to figure out why, their next step would be to try another block, which leaves them without the opportunity to understand why the first block was wrong.

A study examined the effect of giving students binary feedback (right/wrong) on a programming exercise while allowing them to re-submit as many times as they wished (Kyrilov and Noelle, 2016). The study found that the majority of student failed to submit correct solutions.

When Dale Parson first wrote the paper on parson's programming puzzles, he notes that the majority of students wanted a better feedback system. They wanted specific explanations discussing their errors. A study also found that personalized feedback improves student engagement in a game (Lee & Ko, 2011). Studies in classroom settings also show that giving students immediate feedback during exams not only improves student retention but also decrease the negative effects of getting the wrong answer (Butler & Roediger, 2008). Garris et al.

(2002) argues that learning happens when students consider and evaluate every possible option. Therefore, students that are unable to know why their solution is wrong would benefit from feedback giving them an explanation of their mistake and a deeper understanding of concepts.

Usually when teaching students, this burden of providing personalized feedback falls on instructors or teaching assistants which is time consuming and makes it difficult to scale (Rivers and Koedinger, 2013). Another challenge, as Helminen et al., notes is that a student's process of solving is invisible to the teacher and so the teacher would be unable to provide guidance or feedback when they face problems.

One of the goals for the semester has been to ameliorate these difficulties and provide feedback to students using other student's solutions on Parson's Puzzles by taking advantage of its simpler structure. Combined with the work done during the first half of this semester (discussed below), feedback is meant to help students understand the different approaches, the wrong ones and the correct ones in Parson's puzzles.

A large part of this paper also includes efforts to increase student collaboration as well as student reflection. Sancho et al. states the importance of collaboration when learning programming in a game-based learning environment. This study argues that in this way slow-paced students can observe fast-paced students and learn from their behaviors and the judgments they make. Fast-paced students will also benefit from each other. Barker et al. also reports that student to student collaboration is a powerful predictor that learning is happening. In addition, reflection is also a very important part of learning that includes noticing and correcting one's mistakes. Self-Explanation in parson's puzzle forces students to reflect on their solutions, reinforcing what they learned. With this in mind, this paper seeks to encourage student collaboration and reflection.

First half of the semester

The focus for the first half of the semester, as noted in the midterm progress report, has been to incorporate visual programming to the extent that it can be incorporated into Scratch. As mentioned in the proposal, having a visual representation of their solutions can also be very useful in helping students spot and better understand errors they are making. Especially if they are able to see how that representation changes with changes they make to their solution.

In 2012, Bret Victor, in an article he wrote called "Learnable Programming", explains that a coding environment should be able to let the learner easily understand what the syntax means, understand what happens and when, what state the program is currently in, get to a solution by correcting mistakes and working towards it, as well as get comfortable enough with a piece of

code so a learner can abstract it. Visual representation lets the learner understand the code execution and what state it is in, modify the code to get to the goal state using logical reasoning, and trust the code enough that learners can comfortably abstract it and move on to greater problems. Student using visual aids are therefore better able to conceptualize computer science concepts that foster computational thinking like recursion, iteration, and abstraction because they learn them in context.

Since most of the variables in Scratch, and parson's puzzles, are in the form of sprites whose attributes are visibly obvious, programming in scratch is similar to visual programming except for the few ways that they are different. While in traditional programming languages, variable initialization occurs during execution, in scratch it occurs during development when sprites are created with a default set of values (Franklin et al.). In parson's puzzles if a student script is run twice, the starting value in the second execution is the same as the final value at the end of the first execution. Although it is therefore convention to start the scripts on the green flag where initialization could occur, it isn't enforced.

Their study also finds that the carrying of state between executions, alters student ideas of initialization. Although it is important that variables be initialized at the beginning of a program, Franklin et al. reports that this is not obvious to students since they already see the sprite with certain values. Instead, most students try to initialize at the end of the program with the hopes of initializing the sprite for the next execution of the program. However, Franklin et al., reminds readers that program execution can be stopped at any time and there for initialization at the end of a program is therefore not guaranteed. The study also argues that initialization should occur at the start of the green flag, the point of execution by convention.

Related works

Gross et al, like the study done by Rivers and Koedinger, uses student solutions and clustering to provide feedback to students. However, although these methods use student solutions, they still don't provide detailed explanation of the errors if the current student solution happens to be wrong. As previously stated, intelligent hinting in SAGE also does not provide this detailed feedback but instead shakes a collection of suggested blocks for the student to try when they click the hint button.

Kyrilov and Noelle (2016) proposed using Automated Assessment systems to provide detailed feedback to students. They used clustering algorithms to group student submissions based on how similarly incorrect they are, which allowed the instructor to provide only one feedback per cluster. Their study found that this resulted in labor savings while allowing them to still provide detailed and instant feedback.

Regarding visual programming, a study of live programming environments on novice programmers showed students how variables change as each line of code is written, as well as the history of the variables along with their current values (Kang and Guo, 2017). The study yielded very promising results. It was found to help students form proper mental models and more clearly discuss a program's behavior.

Regarding misconception of initialization, Franklin et al., gathered four pieces of knowledge that they argued will result in robust implementation of a program in Scratch. The study argues that students should always initialize at the beginning, initialization should occur in the green-flag script, initialization should be hidden from the user, and that only absolute blocks should be used to initialize.

Features

Step-explanations

This feature was implemented in an effort to increase student learning reinforcement and crowd-source detailed feedback. Currently, if a student drops the wrong block from the parson's palette onto the scripts pane, the student would get feedback indicating that the step they just made was wrong. If the student manages to remove the wrong block and add the right block, an "explain step" button will show up below the blocks in the parson's palette. The student can then click this button which will pop open a dialog box that asks them to explain what their understanding was when they made that mistake and what advice they would give to students that are in the same situation as they were. This will force the student to think about why their step was wrong, leading to a deeper understanding of the concepts the puzzle is trying to teach.

In order to incentivize them to explain their step and make this process more Gameful, the student will get a point back if they submit their explanation. In addition, if a student manages to drop the correct block after dropping the wrong block more the once, the explain step button will not show up. This way the students are forced to really think about why they got a step wrong and what the correct one might be before trying a different block, if they want the extra point.

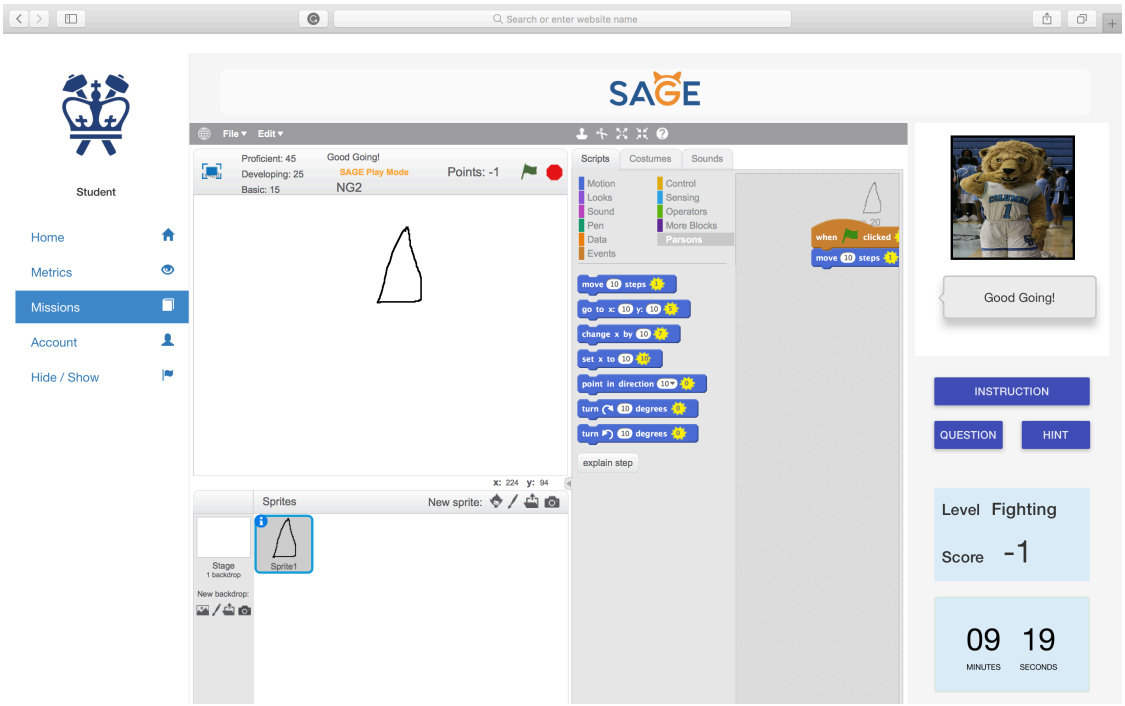

Fig 1: "explain step" button appears where a student can explain their previous wrong step

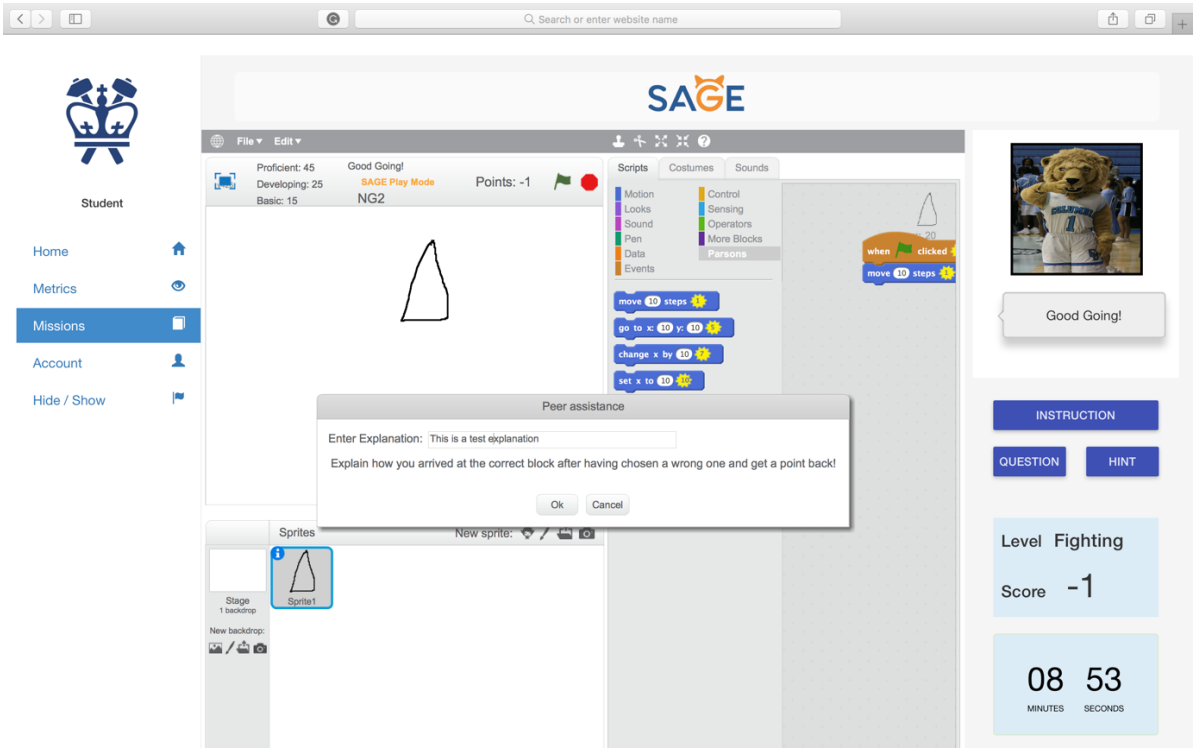

Fig 2: when the “explain step” button is clicked, a dialog box pops up

Feedback generation from peers

This feature was implemented in an effort to integrate student collaboration and detailed feedback into parson’s puzzles. If a student drops the wrong block, an “advice from your peers” button will show up which when clicked will pop open a dialog box with a list of student explanations that were submitted for the same mistake, if there are any.

The student can use these explanations to figure out why the step they just made was wrong and in doing so get closer to figuring out what the correct block is. This explanation, having come from students that were in the same situation as them and are better fit to explain how they got past their mistake and what their misunderstanding was, is meant to cultivate peer learning. This can also help overcome the “expert blind spot”, where an instructor, an experienced programmer, might have difficulty anticipating what students struggle with (Nathan et al., 2003).

To stop students from using this button too much, every time they use it they will get a point deduction.

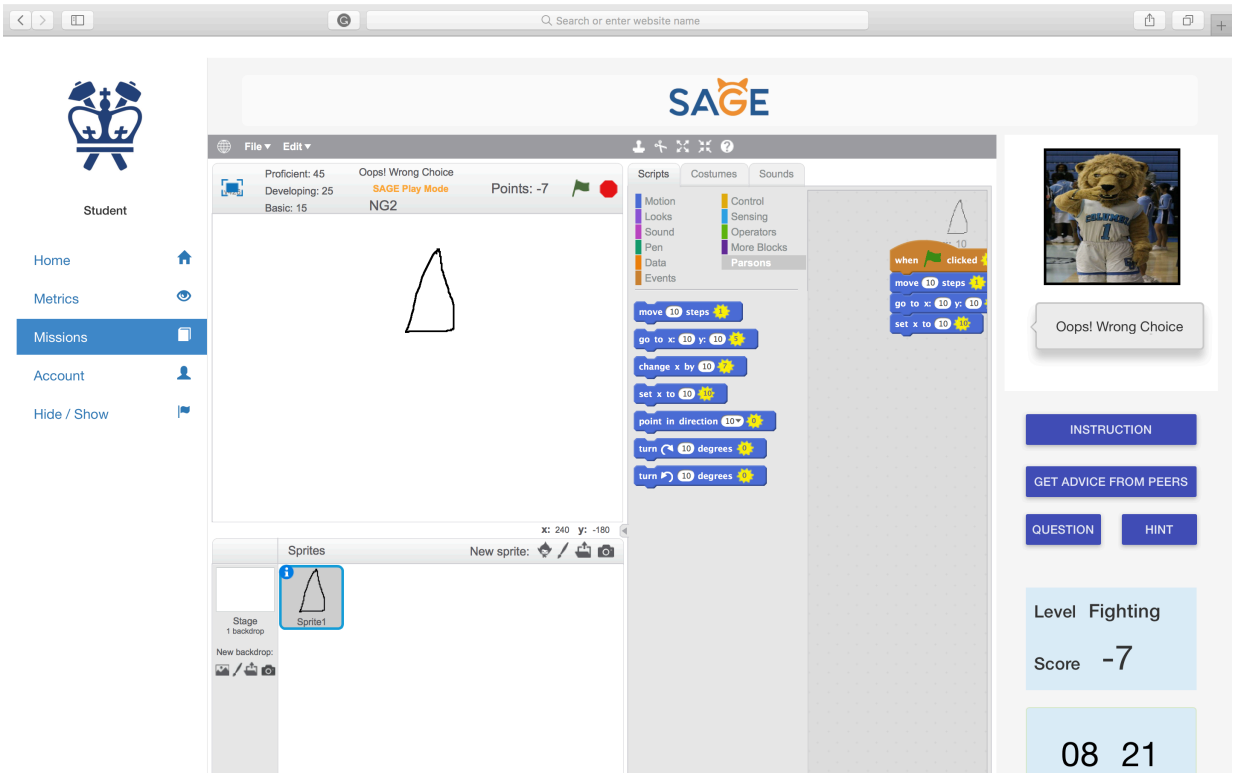

Fig 3: The “get advice from peers” button shows up when a student drops the wrong block.

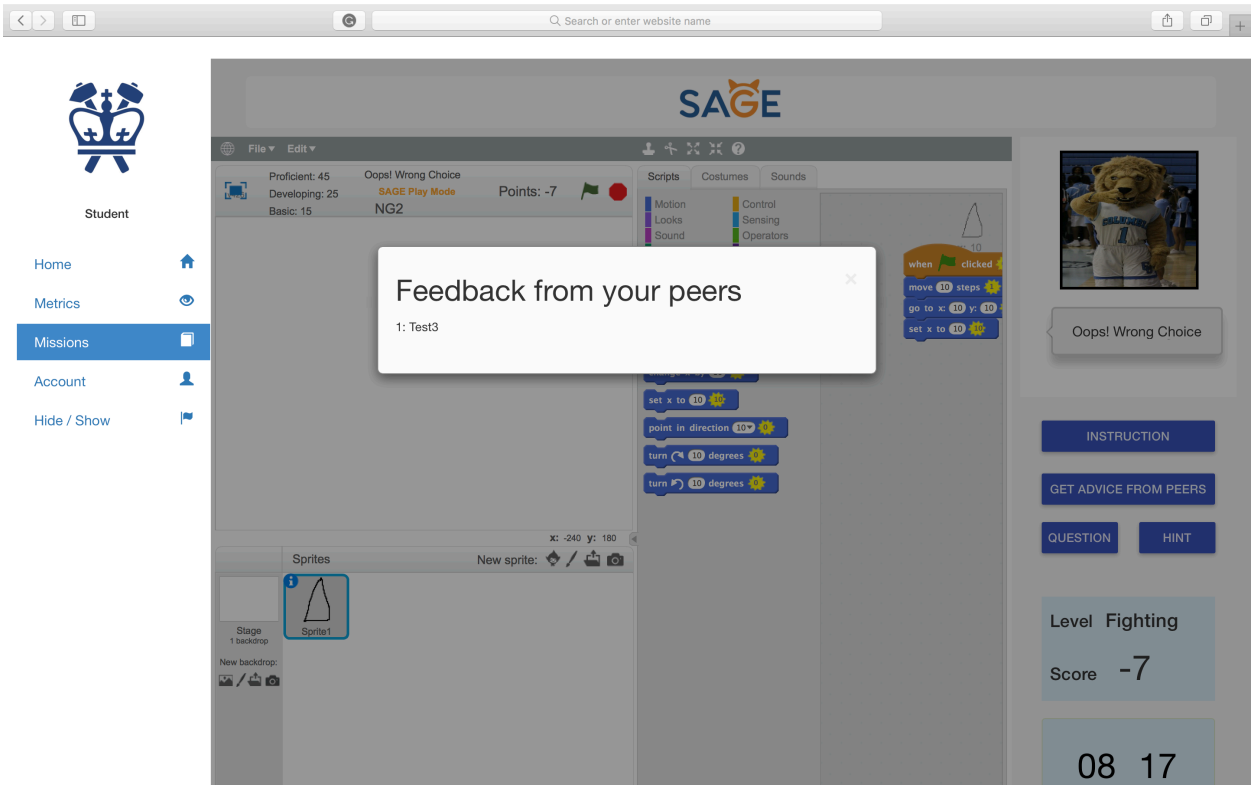

Fig 4: When the “get advice from peers” button is clicked, a dialog box pops up.

Instructor feedback

If an instructor wants to add an explanation in addition to the explanations that students will or have submitted, they can do so by editing the design of the game or when authoring the solution during the initial design of the puzzle. In design mode, there will always be an “add feedback for error” button that when clicked will pop open a dialog box prompting the instructor to add feedback for when a student reaches the sequence of blocks the instructor has authored in the scripts pane at that moment. This can come in handy if an instructor thinks a certain mistake is common and is worth providing an explanation for, or the other students have not explained it well enough.

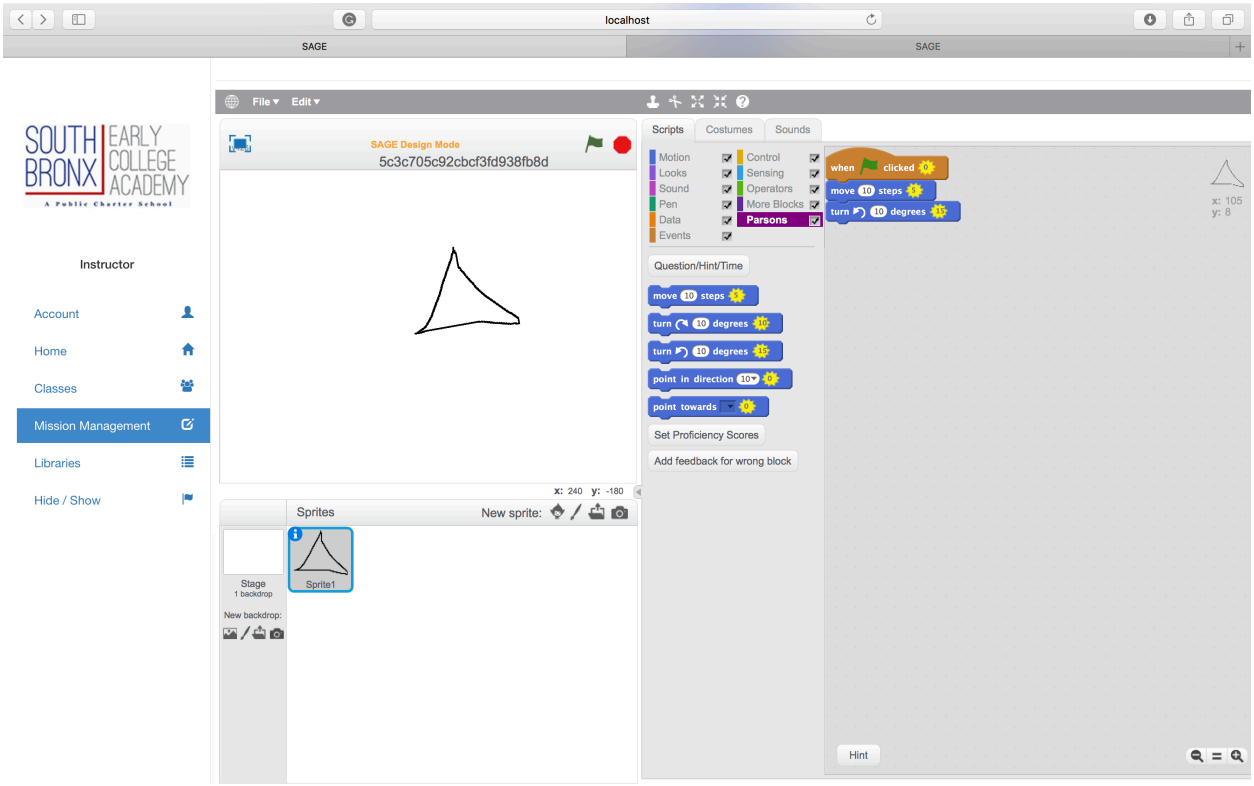

Fig 5: “Add feedback for wrong block” button in parson’s palette when in design mode.

If there is an instructor explanation present for the mistake that the student makes, the “get advice from peers” button will show the instructor feedback first before showing explanations from other students.

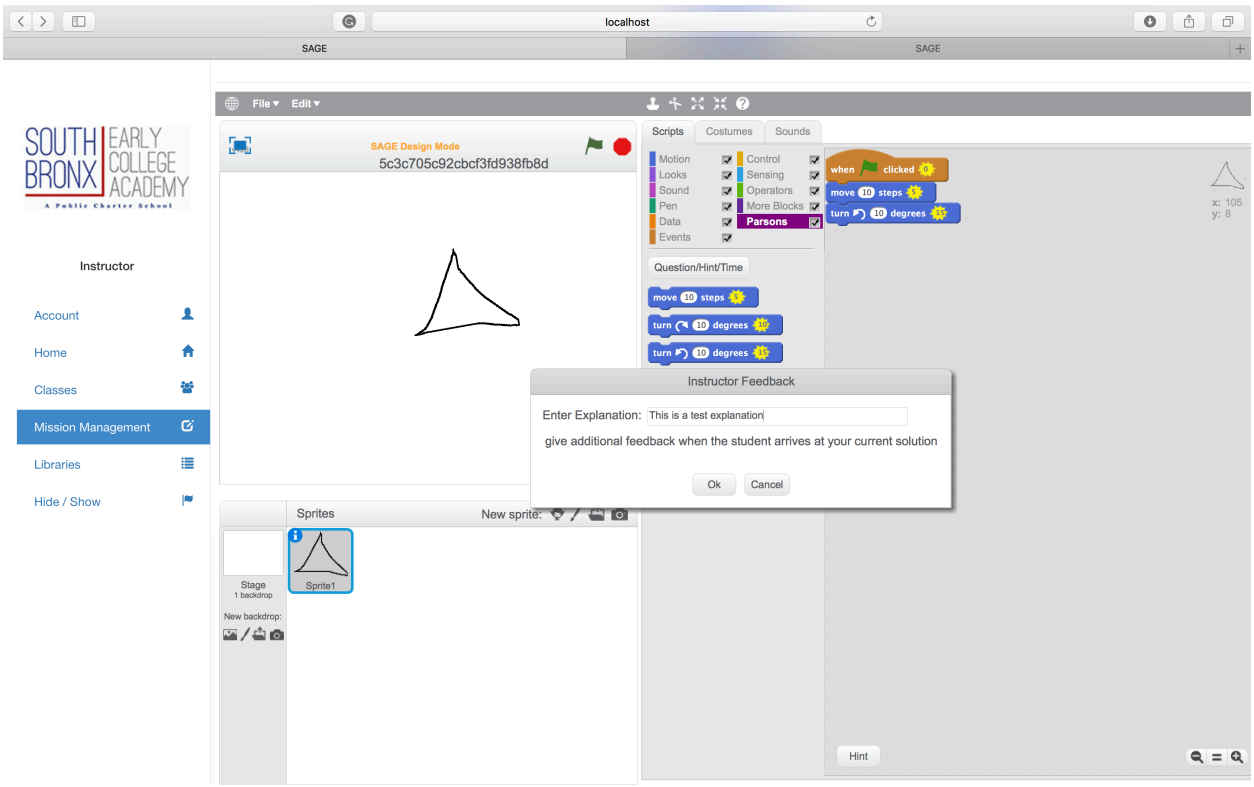

Fig 6: Instructor feedback dialog box

Auto-initialization

When a student's solution is run, if their solutions actually involve moving the direction or position of sprites, at the end of the execution, their sprites will potentially be in a new location with new direction. However, with auto-initialization, the sprites will start from their initial position every time their solution is run.

Auto-initialization was implemented with the intention of eliminating the misconception that students may get about initialization when sprite states carry between executions, as explained before. Auto-initialization forces students to initialize at the beginning of the program because they can no longer rely on the end state of the previous execution.

This also has the added benefit of showing them how the history, in this case the path of the sprite, and final result of the sprites change with changes they make to their solutions. This way, at the end of each execution, they'll see the state of the sprite as changed by their entire solution being applied to the initial state of the sprite. This makes it easy for them to compare the execution of the code they saw with and without the changes they make, allowing them to see if they are heading in the right direction.

The initial sprite location is just the initial random location the sprite gets during creation but teachers are able to change this initial position by either using scripts or by dragging the sprite. In addition, since sprites now always start from a fixed state that doesn't rely on previous executions, students are able to initialize with absolute blocks as well as relative blocks.

Auto-execution

This task was motivated by wanting to show students how the changes they make to their solution, dropping a block in the scripts pane, affects the intended outcome of their solution more immediately. Now, as soon as a student drops a block on to the scripts pane, their new solution would run automatically instead of them having to press the Green Flag button themselves.

This is similar to the work done by Kang and Guo, where they implemented visual programming by showing students how each line of code changes not only the final states of the variables but also the history of the variables as it is written. In Parson's Puzzles' case a line of code is a block and the automatic execution at the drop of a block is comparable to the execution of code as each line is written. Together with auto-initialization, since the variables in Parson's puzzles are sprites which move from a fixed initial state to a final state with each execution, the history of variables is comparable to the path of the sprite.

Green-flag automation

Following convention as Franklin et al. stated, Auto-execution requires they have a "When green flag is clicked" block at the beginning of each program. Consequently, this block has been added as a default block in any new sprite that is created whether in design mode or play mode. This was implemented with the intention of systematically enforcing said convention.

Implementation and Architecture

Step-explanation

The implementation and execution of this feature starts from parsonsLogic method in Scratch.as in Sage-scratch. Whenever a student drops the wrong block, there is logic in the parsonsLogic method that will keep track of whether this is the first time a student has dropped a wrong block since the last time the student had a correct partial solution. This logic also keeps track of when a student drops the right block after a wrong one, and only then will it make the explain-step button visible. After the explanation is taken in, Scratch will send a signal with the updated point (after having incremented one point) to sage-frontend so that the student's score will be incremented by one in the front-end UI as well. Subsequently, it sends another signal to sage-front-end with the incorrect block previously tried by the student and the correct block right above the incorrect one, along with the student explanation. Sage-frontend receives this signal and saves this in the peerFeedbackModel model in the database under the current assignment and student id. This particular student explanation is uniquely identified by the wrong block they dropped and the correct block found at the end of their correct partial solution before they dropped the wrong block.

If a student drops a wrong block more than once, they will lose the opportunity to explain their step when they eventually drop the right block. But after they do eventually manage to drop the right block, the logic is reset and they can have the opportunity to explain their steps thereafter.


```
Received: {
        "event": "SEND_FEEDBACK",
        "correctBlock": "forward:",
        "studentID": "59f369cc748499467c32a414",
        "firstBlock": "whenGreenFlag",
        "assignmentID": "5c0d7d091da8bb164894dcde",
        "feedback": "Explaining",
        "wrongBlock": "heading:"
}
```



```
Received: {
        "event": "UPDATE_POINT",
        "objectiveID": "undefined"
        "correctBlock"
        "timestamp": "1547841178352",
        "studentID": "59f369cc748499467c32a414",
        "wrongBlock": "heading:",
        "assignmentID": "5c0d7d091da8bb164894dcde",
        "feedback": "Good Going!",
        "isFinal": false,
        "newPoint": "4",
        "firstBlock": "whenGreenFlag"
}
```


Scale document down

Fig 7: the send feedback JSON event sent from scratch to frontend and Fig 8: The update point JSON event sent to frontend right before the send feedback event

Feedback generation from peers

In line with the efforts to move frontend UI to sage-frontend, when a student drops the wrong block, sage-scratch sends the wrong block, the correct block from the solution, and the last correct block in the student solution along within the UPDATE_POINT JSON event to sage-

frontend. Sage-frontend will retrieve the list of explanations for the current blocks from the peerFeedbackModel and inserts it into the tempPeerFeedbackModel. When sage-frontend checks for score updates every second in the controller for the page that contains the scratch game, if it finds that the current state of the student solution is wrong, it will also query for feedback updates every second which will return the current list of explanations saved in the tempPeerFeedbackModel. This allows the student to have the most updated list of explanations specific to the wrong state their solution is in at any time.

Instructor feedback

When the instructor is in design mode and has authored a solution that has at least 2 blocks, an "Add feedback for wrong block" button is used to take in the feedback and send a "INSTRUCTOR_FEEDBACK" event signal to sage-frontend. Sage-frontend then saves this feedback in peerFeedbackModel under the assignment ID. When a student requests feedback, the instructor feedback is collected and added to the tempPeerFeedbackModel and consequently shown to the student along with the other student explanations.

Auto-execution

Pressing the green-flag button is simulated whenever the parsonsLogic method in Scratch.as is executed. This method is executed when a student makes any kind of move, drops a block onto the scripts pane or drags a block off from the scripts pane, which allowed for easy implementation of auto-execution.

Green-flag as default block

Whenever a new sprite is added, the green flag block is automatically added to the list of scripts for that sprite as default.

Auto-initialization

When a sprite is first created its initial location, direction, and rotation style is saved. In play mode, after the execution of the blocks in the scriptsPane, the position and direction of the sprites are change to their initial state. This will take effect the next time the script is run, allowing the script to make changes to the initial state of the sprite as opposed to the previous final state.

Bug Fixes

ParsonsLogic

Previously parsonsLogic only gave feedback based on whether the last block the student has in the scripts pane matched the block at the same position in the solution scripts. This was because even if parsonsLogic found an incorrect block in the solution, if the last block the method checks after that is correct (meaning it matched the block in the solutions script at the same position), positive feedback would override the constructive feedback no matter how many mistakes there were before the last correct block. This caused confusing feedback messages for the student in different cases. Now, in a situation like this the feedback will stay "Try removing few blocks!" until the student has removed all blocks including the first incorrect block they dropped.

ParsonsLogic also previously did not account for blocks in for or while loops. It used to only check to make sure the first block inside of a loop was correct. Now, it considers all blocks even in loops and gives appropriate feedback.

In addition, if the number of blocks in the scripts panes was the same number as that of the solution, then parsonsLogic would incorrectly congratulate the student and submit the assignment. Now, parsonsLogic make sure every block is correct before submitting assignment.

Lastly, when an instructor updates the design of the game, if his authored solution matches his previous one, the puzzle would incorrectly congratulate and submit the project. This has now been fixed by calling parsonsLogic only when in play mode.

Limitations

The first half of the semester was largely spent trying to understand the code-base for sage-scratch and coming up with ideas to implement. The ideas implemented also have some limitations.

Wrong explanations

Students submitting explanations for mistakes they corrected might be a great opportunity for reflecting and gaining a deeper understanding, but since these explanations are also being used for the benefit of other students, it might also be the case that incorrect explanations end up confusing the students it was meant to help.

On the other hand, since feedback requesting students have the opportunity to view all feedback submitted, a confusing explanation among other helpful ones might not have a negative effect. However, this might not always be the case.

In addition, currently if a student is not guided by the feedback returned by parsonsLogic (for example, if they remove a block from the middle but manage to go back to an early correct solution by removing almost all the blocks in one try), their feedback might be requested. Even though this feedback will never be shown to other students because of the way the feature was implemented, they will still get a point back for it.

Speed of auto-execution and auto-initialization

As previously explained, when a student edits their solution script, auto-initialization will reset the sprite's position, location, and rotation style to its initial values and auto-execution will run the student script. However, since Scratch executes a student's solution in a matter of seconds, students might miss the sprite going back to its initial position right before their solution is applied to the sprite's initial state.

Green Flag in Play Mode

In addition to adding the Green Flag block in any new sprite that was created, the intention was to also show only the Green Flag block whenever Scratch was in play mode, excluding the rest of the solution. However, there is currently a bug in Scratch that shows the solution when Scratch

is opened in play mode and since I was unable to fix this, I was also unable to complete said implementation.

Future Work

Fix for fast speed of auto-execution and auto-initialization

The limitations of these features can perhaps be removed if the fixed initial position of the sprite was marked somehow (by maybe a watermark of the sprite) to make it clear to the student where the sprite's initial state is. In addition, to make visualization easier, a button could be implemented that shows a path between the initial and the final position of the sprite.

Filtering feedback from students and requesting feedback from instructors for specific mistakes

Although only students that took not more than one step to correct their mistake are given the chance to give feedback, more filtering might be helpful to avoid confusing explanations. For example, we could consider the number of moves a student has made so far relative to how far the student has gotten in the solution. Another approach might be to consider performance on past assignments as well.

Instructor feedback could also be requested for the most common mistakes or for the mistakes that students can't solve with less than an arbitrary number of tries.

Timing of "get advice from peers" button

Just like the "hint" button, a student's performance or behavior could be considered to decide when (or after how many tries) to display said button to the student and allow them access to explanations.

Conclusion

In conclusion, applying visual programming to Parson's puzzles' simpler structure, auto-execution and auto-initialization are meant to act as a feedback mechanism such that as every block is positioned in the script pane, the sprites state (position and direction) is shown updating from its initial state. By looking at how the sprites change, students can see that the code is heading in an undesired direction. They'll understand better what their code is currently doing and have a better idea of what their next step will be because they're more likely to know what the problem they have to overcome is. This also ameliorates misconceptions student can have of initialization when sprite states carry over executions.

A large part of the efforts made in the second half of the semester went towards including student collaboration and detailed feedback for mistakes into Parson's puzzle, again by making use of its simpler structure. Combined with auto-initialization and auto-execution, detailed feedback and reflecting on mistakes is meant to help students understand the different approaches and have a deeper understanding of concepts.

References

Barker, L. J., McDowell, C., & Kalahar, K. (2009). Exploring factors that influence computer science introductory course students to persist in the major. *SIGCSE Bulletin*, *41*(1), 153–157. doi:10.1145/1539024.1508923

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

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

Franklin, D., Hill, C., Dwyer, H., Hansen, A., and Iveland, A. Initialization in Scratch: Seeking Knowledge Transfer.

Garris, R., Ahlers, R., & Driskell, J. E. (2002). Games, motivation, and learning: A research and practice model. *Simulation & Gaming*, *33*(4), 441–467. doi:10.1177/1046878102238607.

Gross, Sebastian, et al. ”Cluster based feedback provision strategies in intelligent tutoring systems.” Intelligent Tutoring Systems. Springer Berlin Heidelberg, 2012.

Helminen, J., Ihantola, P., Karavitra, V., and Malmi, L. How Do Students Solve Parsons Programming Problems? — An Analysis of Interaction Traces. 119.

Kang, H. and Guo, P. (2017). OmniCode: A Novice-Oriented Live Programming Environment with Always-On Run-Time Value Visualizations.

Kyrilov, A., and Noelle, D. (2016) Do students need detailed feedback on programming exercises and can automated assessment systems provide it?

Lee, M.J., & Ko, A.J. (2011). Personifying programming tool feedback improves novice programmers' learning. *ACM ICER,* 109-116.

Mohan. (2017). Gameful Direct Instruction (Parson’s Puzzle).

Nathan, M.J., and Petrosino, A. Expert blind spot among preservice teachers. *American educational research journal, 40(4), 905-928, 2003.*

Parsons, D. and Haden, P. (2006). Parson’s Programming Puzzles: A fun and effective learning tool for first programming courses.

Rivers, K., and Koedinger, K. (2013). Automatic generation of programming feedback: A data-driven approach. 50-59.

Sancho, P., Gomez-Martin, P. P., & Baltasar, F. M. (2008). Multiplayer role games applied to problem-based learning. In *Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts*, Athens, Greece (pp. 69-70).

Victor, B. (2012). Learnable programming: Designing a programming system for understanding programs.