

COMS 3998 Spring 2019 Final Report: Parson's Puzzle Game Library

Calvin Goah
Columbia University, New York, NY
cgg2126@columbia.edu

Table of Contents

1 Abstract	2
2 Introduction	2
3 Related Work	3
3.1 Two-Dimensional Parson's Puzzles: The Concept, Tools, and First Observations (Petri Ihantola and Ville Karavirta)	3
3.2 Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses (Dale Parsons and Patricia Haden)	3
3.3 A Mobile Learning Application for Parsons Problems with Automatic Feedback (Karavirta et al.)	4
3.4 How Do Students Solve Parsons Programming Problems? – Execution-based vs. liene based feedback (Helminen et al.	4
4 Implementation	5
4.1 Counting Loops	5
4.2 Sentinel Loops	6
4.3 Conditional Loops	7
4.4 Review	8
5 Bug Log	9
6 Future Work and Guidelines	9
7 Conclusion	10
References	11

1 Abstract

This paper focuses on the Parson’s Puzzle Library feature in the Gameful Direct Instruction Epic. It details how games for the Parson’s Puzzle Design: Loops User Story were constructed as well as some of the bugs encountered during game creation using the SAGE Dev. site. Several games ranging in difficulty from easy, to medium, to hard were created. These games covered the looping concepts of counting, sentinel, conditional, and fence-post looping. The method used to create these games is based on the concept that each puzzle needs to lead to a unique solution. Thus, ingredients for conditional statements were made apparent to the player so as not to distract her from the essential purpose of the games. Work, in the form of surveys and making key contacts, began in relation to creating an in-class testing scenario with a nearby public school.

2 Introduction

Parson’s programming puzzles are a family of code construction assignments where lines of code are given, and the task is to form the solution by sorting and possibly selecting the correct code lines [1].

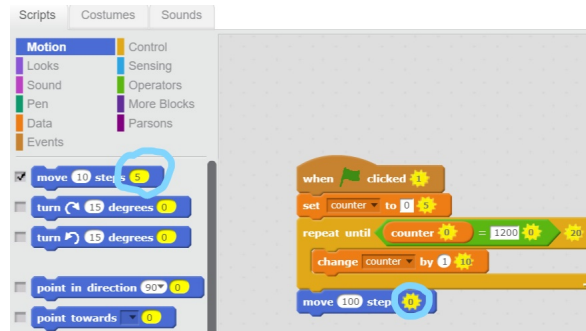


Figure 1: Example Parson’s Puzzle

The objective for this semester was to work on the Gameful Direct Instruction Epic on the SAGE backlog, with a special focus on the Parson’s Puzzle Library.

The previous state of the game library consisted of the following computational thinking concepts: Sequences, Loops, Events, Parallelism, Conditionals, Operators, and Data. The goal for each concept is to develop several Parson’s Programming Puzzles that instructors could use to properly teach that concept. The focus for this semester was primarily on the looping concept. Several puzzles were created for this concept ranging in difficulty from easy, to medium, to hard. Furthermore, the looping concepts were broken down into three primary components: counting loops, sentinel loops, and conditional loops. In addition, an extra game for fence-post loops was created.

During the game design phase, several bugs made game design and testing slightly difficult, and those are detailed in this paper. The layout of this paper is as follows. A brief discussion will be given concerning the various papers used in coming up with a model for game design, then we will discuss a few of the games from the looping library, detail some of the bugs from the development site, and then finally give some leads as to future works before concluding.

3 Related Work

3.1 Two-Dimensional Parson’s Puzzles: The Concept, Tools, and First Observations (Petri Ihantola and Ville Karavirta)

Ihantola and Karavirta survey the current landscape of approaches towards Parson’s Puzzle construction then present a new method for Parson’s Puzzle inspired by the Python programming language know as a Two-Dimensional Parson’s Puzzle. Essentially this new method uses the vertical ordering of lines to denote ordering as in traditional Parson’s puzzles; whereas, the horizontal dimension is used to define code blocks based on indentation, as in Python. Lastly, they discuss proper methods of designing puzzles, of which the most important design principles are avoiding ambiguity and utilizing distractors.

3.2 Parson’s Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses (Dale Parsons and Patricia Haden)

Parsons and Haden describe the motivations for Parson’s Programming Puzzles. They touch on best practices for design and state that since each puzzle solution is a complete sample of well-written code, use of their tool exposes students to good programming practice. They also describe a web-based authoring tool used to build the puzzles. In their introduction, they describe two main problems associated with drill-based exercise. For one, they are boring and lack engagement. For two, it is difficult to remove a single syntactic unit from the logical context in which it occurs. To address these issues their proposed tool attempts to maximize engagement, constrain the logic by providing good code structure, permit common errors allowing students to make direct comparisons between what they commonly do wrong with what the correct syntactic construct looks like, model good code, and provide immediate feedback. The puzzle described is in a drag-and-drop style in which a student is given a selection of code fragments and the student is tasked with dragging choices into the indicated answer locations. Lastly, they discuss built-in distractors that are meant to provide nuance and subtleties in each puzzle for a stronger understanding.

3.3 A Mobile Learning Application for Parsons Problems with Automatic Feedback (Karavirta et al.)

Karavirta et al. present a mobile application tool called MobileParsons that facilitates the learning of programming for Parson's programming puzzles. Their argument for this approach is that Parson's problems are attempted in small chunks and as a result are suited to a mobile application. They present issues with automatic feedback associated with trial-and-error behavior in the context of frequent feedback requests. Strategies that are currently applied in attempting to solve these issues include limiting the number of submissions allowed and/or limiting the details of feedback. Karavirta et al. propose feedback for partial solutions, such that a student is given continuous feedback for every possible state he or she gets in. To do this, they implement a feedback algorithm known as the longest common subsequences (LCS). This algorithm searches for the longest common subsequence shared by the model and the student solution that has the greatest number of consecutive code fragments in the student's solution. Another implementation Karavirta et al. devised that is intended to improve automatic feedback for trial-and-error behavior was to implement a functionality that recognized aimless exploration resulting in loops (revisiting previous states of the solution along the solution path). Lastly, they propose a solution to gaming the system for constant feedback by instilling penalties for the frequency with which students request feedback; thus, encouraging exploration as opposed to hacking.

3.4 How Do Students Solve Parsons Programming Problems? – Execution-based vs. line based feedback (Helminen et al.)

Helminen et al. discuss a field study in which a group of students was split in half in order to determine the most efficient means by which students learn. Specifically, the paper presents a study on how the type of automatic feedback in Parsons problems affects how students solve them. It was discovered that the type of feedback had an effect on how students constructed their programs and how quickly they were able to complete them. With feedback based on execution as opposed to the visible arrangement of code, the programs were more frequently executable when feedback was requested and, overall, feedback was requested less frequently. The two primary contributions to the existing knowledge of automatic assessment and feedback on Parsons programming problems presented in this paper include an extension of js-parsons with execution-based feedback where tests are run on learner's code and then checked for expected results, and the second is a comparison between this model and the existing line-based feedback.

4 Implementation

In this section I will be discussing how I went about designing each game. As stated previously, I mainly focused on the looping concept and broke it down into three main components: Counting Loops, Sentinel Loops, and Conditional Loops. Each of which had an easy, medium, hard level.

The screenshot shows a web interface for 'Quest Management'. At the top, there's a blue header with the text 'Quest Management'. Below it, there's a form for editing a quest. The quest title is 'Basics of Looping'. The quest description is a text area containing the following text: 'Counting Loops: Easy, Medium, Hard', 'Sentinel Loops: Easy, Medium, Hard', 'Conditional Loops: Easy, Medium, Hard', and 'Extra: Fencepost Loop'. Below the text area is a blue button labeled 'UPDATE QUEST'. Below the form, there's a section titled 'Games and Objectives in this Quest'. This section contains a table with the following columns: 'Order', 'Game Description', 'Design', 'Objective', 'Instruction', and 'Delete'.

Order	Game Description	Design	Objective	Instruction	Delete
1 ▼	Counting Loops (Easy)	UPDATE	UPDATE	UPDATE	DELETE
1 ▼	Counting Loops (Medium)	UPDATE	UPDATE	UPDATE	DELETE
1 ▼	Counting Loops (Hard)	UPDATE	UPDATE	UPDATE	DELETE
2 ▼	Sentinel Loops (Easy)	UPDATE	UPDATE	UPDATE	DELETE
2 ▼	Sentinel Loops (Medium)	UPDATE	UPDATE	UPDATE	DELETE
2 ▼	Sentinel Loops (Hard)	UPDATE	UPDATE	UPDATE	DELETE
3 ▼	Conditional Loops (Easy)	UPDATE	UPDATE	UPDATE	DELETE

Figure 2: List of Games

4.1 Counting Loops

For these types of loops I had to make sure that each puzzle accurately conveyed the idea that counting loops always repeat/execute the same number of times. Moreover, sometimes the program calculates the number of repetitions based upon user input. keeping [3] in mind, I created a simple mapping of the blocks I wanted to use for this concept. The issue with these types of games, and parson's puzzles in general is that the puzzle creator should avoid blocks that aren't imperative to the learning of the concept. So, for some most of the finished questions, I provided hints as to what to input for each conditional if user input is required, thus keeping the focus on the ordering of the blocks.

As can be seen in Figure 3 below, the objective is to create a nested counting loop. This is one of the more difficult counting loop games. In the question



Figure 3: Counting Loop Example

for this loop, I would tell the student what to set the counter blocks to. For example:

Question: Using the set blocks, make counter and clock zero, then make the repeat until block repeat 5 times (i.e. use the counter variable and the green block). Within the repeat until block, change the clock variable by 1 five times. Show the counter variable at the end of your loops.

This question does two things very well. For one, it provides the student with the necessary information to fill out the nonessential components of the concept, and, for two, it keeps the key idea, that of changing the counter by one after the repeat block, hidden from the student. As this is meant to be a hard question pertaining to counting loops, it should take some time to solve this particular question. Of course an issue with this question is that it gets wordy to try and explain every non-essential component to the student. It would be nice to have a feature where the instructor could fill these parts out and just have the student work on dragging blocks that have preset values.

4.2 Sentinel Loops

With this particular concept, I tried to convey the following information about Sentinel Loops:

1. They are used for reading an unknown amount of input. In the real world,

this tends to be the most flexible way of reading user data.

2. If it is necessary to verify that a user's input falls within a certain range, then a sentinel loop is required.

In accord with that mission, I also wanted to make the games a little bit more fun, as opposed to just simply dragging and dropping, so I tried using a little bit more of the Scratch palette options.

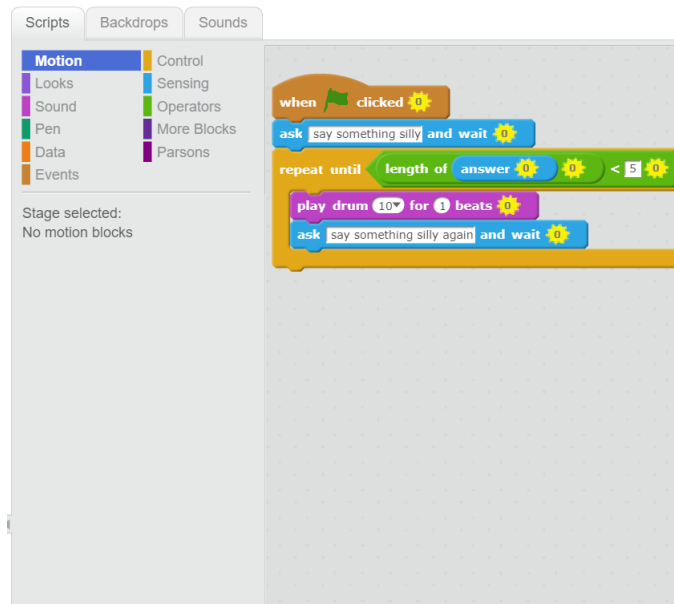


Figure 4: Sentinel Loop Example

Figure 4 shows a medium level puzzle. Again, we have the issue of the nonessential elements, that being the input questions, but as stated earlier this is something that needs more work, so in the main time simply telling the student these inputs ahead of time should suffice. But something to be aware of in Figure 4 is the presence of the “play drum” block. Incorporating such blocks from the “sound” palette is a good area to explore if looking to make the games more engaging.

4.3 Conditional Loops

For conditional loops, the essential idea I tried to convey was that they are like sentinel loops in that it is unknown beforehand how many times they will repeat, and they are also like counting loops in that they terminate as the result of a calculation.

To drive home this duality element of conditional loops, I essentially just combined the puzzles from Figure 3 and Figure 4 in creating the example puzzle

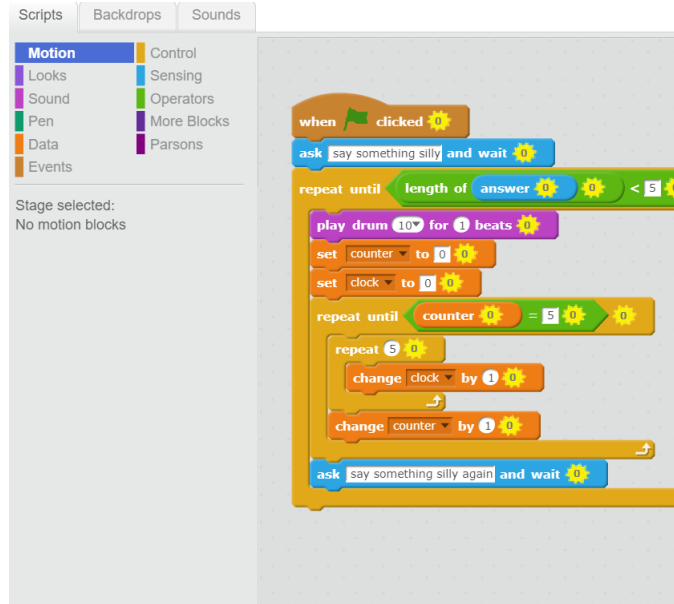


Figure 5: Conditional Loop Example

in Figure 5.

4.4 Review

The previous sections detailed examples from each of the sub-concepts of looping and how I went about designing them. There was an additional looping concept which I added as an extra game called fencepost loops. These are loops where you have n -elements that are of concern but your operation only needs to run $n-1$ times. An example of this would be printing n letters but only needing $n-1$ commas. An example game is given in Figure 6 below.

All of these concepts could be represented very differently in Scratch. As such, it is important to list a few areas that could use some improvement. The games at the moment tend to stand on their own. That is to say that there are not many illustrations or graphics that aid in making the games more transparent and even better mediums of conveying computational thinking. I will detail some possible avenues pertaining to how future SAGE researchers could go about creating illustrations for these and future puzzles.



Figure 6: Fencepost Loop Example

5 Bug Log

During the game design process, I encountered a few bugs that made it a little difficult in testing the puzzles I had created. The two biggest bugs were the one that prevented the games within the quest I was working on (from the instructor SAGE DEV. site) from showing up on the student SAGE DEV. site. As a result, I had to do most of the testing from the instructor side. The other major bug was that towards the end of the semester, the games I created on the instructor SAGE DEV. site wouldn't save properly, so I had to save them locally to my computer and upload them every time I needed to test or demo them. I have since uploaded all the games I had on my computer to a Google Drive folder shared with Jeff Bender.

6 Future Work and Guidelines

There is a lot of area to improve in terms of game construction and ensuring that each game has a unique output and is easily understood by students. On this front, I attempted to get in contact with the principal at PS76 in conducting field test. They tend to move pretty slowly and as a result I was unable to go in for a test trial, but I will connect Jeff Bender with the current principal so if future researchers would like they could set up a meeting to demo some games and get feedback. I've also created some questions that are in a shared folder and easily accessible via Jeff Bender, so those could be used as extra resources.

Another area of improvement is in illustrations for each game. Currently there are no illustrations due to the delay they posed when initially trying to create the actual parson's puzzles. My advice for this would be to attempt to create independent illustrations for each game. It doesn't have to be a full-fledged adventure game of any kind, just something simple dealing with emoji-like sprites would suffice. With that said, if coming up with illustrations seems to hinder the process of creating/populating the game library with actual puzzles, I would recommend focusing on creating the actual puzzles as opposed to worrying about game illustrations.

7 Conclusion

In this paper I talked about creating games for the computational thinking concept of looping, primarily the three sub-components of looping: Counting loops, Sentinel Loops, and Conditional loops. There are a lot more games that need to be created for the Parson's Puzzle library. This is just a small snapshot of what these games could look like. Future development is important and I hope new researchers will embrace every aspect of the journey.

























Child (12)		
	 486 changing the title of a Game Des...	
	Updated 4/18/2019,	● New
	 500 Adding students to class	
	Updated 4/18/2019,	● New
	 496 Code blocked by dialogue box	
	Updated 4/18/2019,	● New
	 501 Deleted Work	
	Updated 4/25/2019,	● New
	 504 Extra game created	
	Updated 4/25/2019,	● New
	 503 Freezing	
	Updated 4/25/2019,	● New
	 505 Negative Points	
	Updated 4/25/2019,	● New
	 497 Nonautomatic point update	
	Updated 4/18/2019,	● New
	 498 Not able to get rid of variables i...	
	Updated 4/18/2019,	● New
	 499 Text description not properly dis...	
	Updated 4/18/2019,	● New
	 502 Variables can't be dragged	
	Updated 4/25/2019,	● New
	 468 Point Deduction On Drop	
	Updated 3/21/2019,	● Resolved

Figure 7: Bug Log, Note: I did not log bug 468

References

- [1] Petri Ihanola and Ville Karavirta. *Two-Dimensional Parson's Puzzles: The Concept, Tools, and First Observations*
- [2] Dale Parsons and Patricia Haden. *Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses.*
- [3] Karavirta et al. *A Mobile Learning Application for Parsons Problems with Automatic Feedback*
- [4] Helminen et al. *How Do Students Solve Parsons Programming Problems? – Execution-based vs. liene based feedback*