

Midterm Project Report

Parson's Programming Puzzle Scoring System

Junyi Wang
Yilan He
Ningchao Cai
Yiming Guo

COMS 6901 SECTION 028

Nov 8, 2018

CONTENTS

Abstract	3
1. Architecture	3
2. Implementation	4
User Story: Parson's Timer	4
User Story: Parson's Score Persistency	5
User Story: Parson's Submit Explanation	6
User Interface Improvement	7
5. Limitations	9
Front-end	9
Server	10
Scratch	10
5. Future Works	11
Front-end	11
Node Server	11
Scratch	12

Abstract

Throughout the semester, we have devoted time and efforts to the Gameful Direct Instruction Epic. The focus was mainly on Parson's Puzzle 1.1 feature and on Parson's Coaching feature. In the following report, we will discuss the architecture modification, the detailed implementation of the user stories, the limitation of our existing systems and what we try to achieve during the remaining semester.

1. Architecture

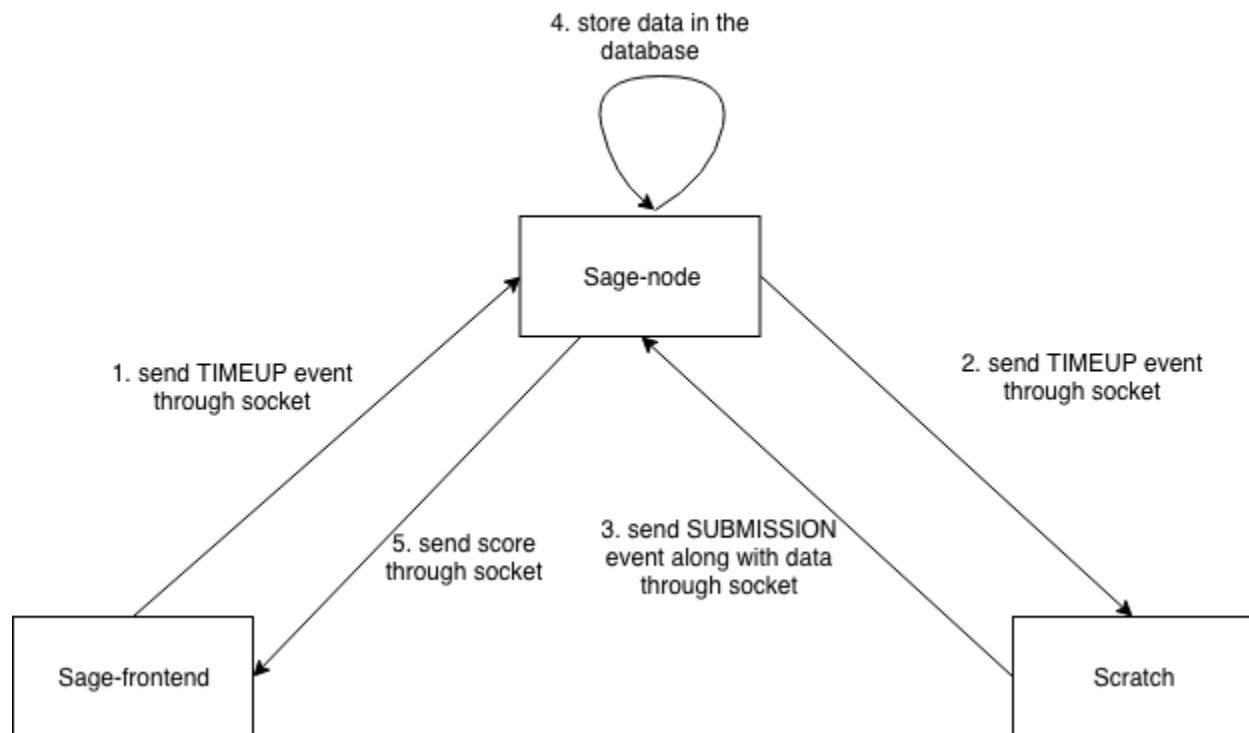
The architecture of the Parson's Puzzle becomes a little more complex compared with before. The main reason is that a number of Scratch functionalities such as displaying time, recording hint usage and displaying scores are now moved to frontend. Therefore, communication between sage-frontend and Scratch is needed. Also, previously, the system just work locally, where the score and time persistency were not implemented, so the communication between Scratch and Sage-node is required now.

Additionally, many of these data transfers need to be timely and reliable. For example, once the timer in the frontend is up, the Scratch should quickly submit the solution.

Delay or loss of data would bring terrible performance and user experience. What makes the design even harder is that now the frontend and Scratch need to actively pull information from the server. Traditional pulling methods would cast tremendous amount of pressure on the server. As a result, we decided to use socket to transfer data between frontend and server, and between Scratch and server.

A socket would be established between Scratch and the server during the Scratch initialization. There will be another socket connected between the server and frontend.

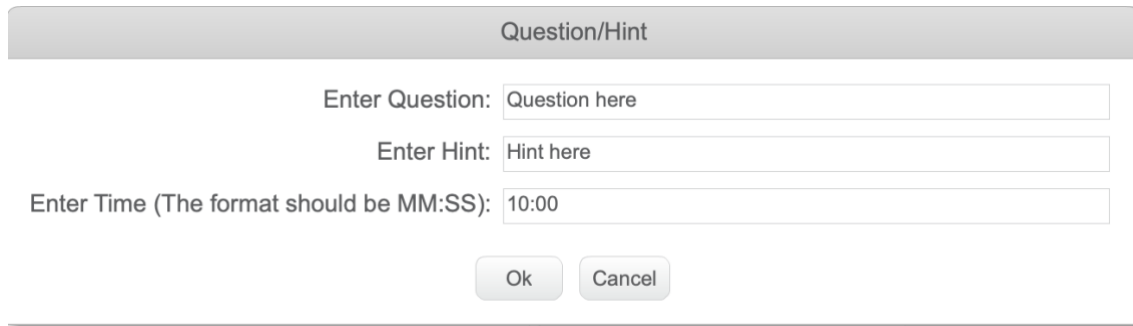
An example of the workflow is shown below:



2. Implementation

- User Story: Parson's Timer

We have successfully implemented the Parson's Timer functionality. Before, the timer only has a default value of ten minutes. Now, the teacher could specify the duration for each assignment when entering question and hint as shown in the figure.

A dialog box titled "Question/Hint" with a light gray header. It contains three input fields: "Enter Question:" with placeholder text "Question here", "Enter Hint:" with placeholder text "Hint here", and "Enter Time (The format should be MM:SS):" with placeholder text "10:00". At the bottom, there are two buttons: "Ok" and "Cancel".

Question/Hint

Enter Question: Question here

Enter Hint: Hint here

Enter Time (The format should be MM:SS): 10:00

Ok Cancel

After the teach specifies the duration, the time configuration for this assignment would be stored in the database. From there, when student enters the game, the front-end would fetch the time configuration from the database and starts to count down automatically.

When the students submit their answers or they reach the correct answer (thus triggering automatic submission), the timer would be stopped.

When the time runs out, a TIMEUP event would be sent to the server then to Scratch. After Scratch receives this event, the submission functionality would be triggered. One major implementation difference compared with before is that now Scratch is no longer responsible for monitoring time. But since the submission event could only be initiated by Scratch, the TIMEUP event needs to be sent from front-end all the way to Scratch.

- **User Story: Parson's Score Persistency**

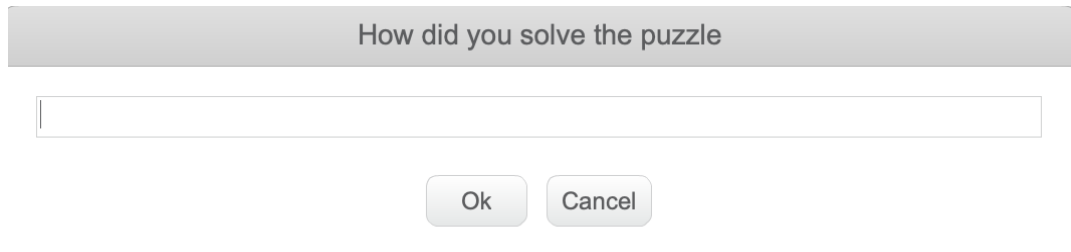
This user story has been mostly completed. To implement this story, we redesigned the submission logic. Before, the submission event could only be triggered by clicking the submit button. Currently, there will be three ways to trigger submission event:

1. Click the submit button
2. Time runs out
3. Student reached the correct solution

Once the submission event is triggered, a summary would be displayed in Scratch. And a json consisting of SUBMISSION event and assignment information would be sent to sage-node server using socket. The assignment information includes studentID, assignmentID, objectiveID, score, hint usage, submit message, self explanation, start time, end time, time remaining, and information of the blocks. Once the sage-node server received the information, the server would store the information in the database.

- **User Story: Parson's Submit Explanation**

This user story is fairly trivial and has been completed. A screenshot of the result is shown here:

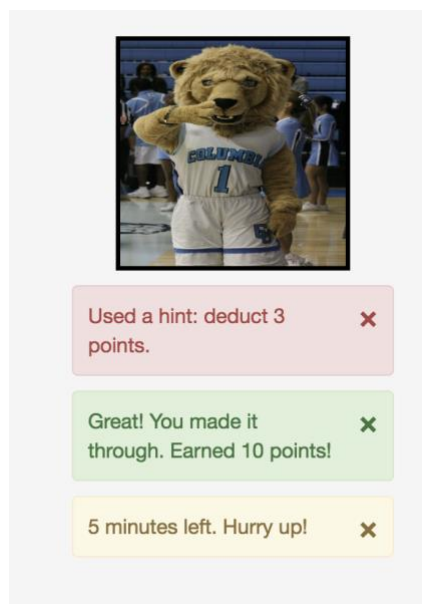
A screenshot of a web-based dialog box. The dialog box has a light gray header bar with the text "How did you solve the puzzle" in a dark gray font. Below the header is a large, empty white rectangular text input field. At the bottom of the dialog box, there are two rounded rectangular buttons: "Ok" on the left and "Cancel" on the right, both in a light gray color.

More styling would be applied in the future and we will discuss later, we intent to move explanation submission to the frontend.

- User Interface Improvement

- Message panel

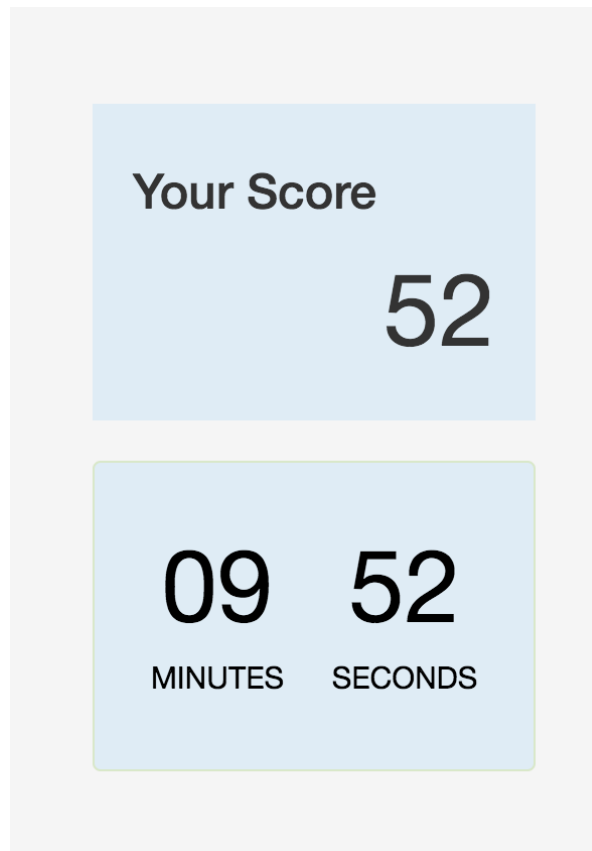
We have created a message panel for showing dynamic message as the students are playing. Some message can be triggered either with their action in the game or the time left. We also placed the lion icon to make it look like the lion is telling them.



- Score display & Timer setting

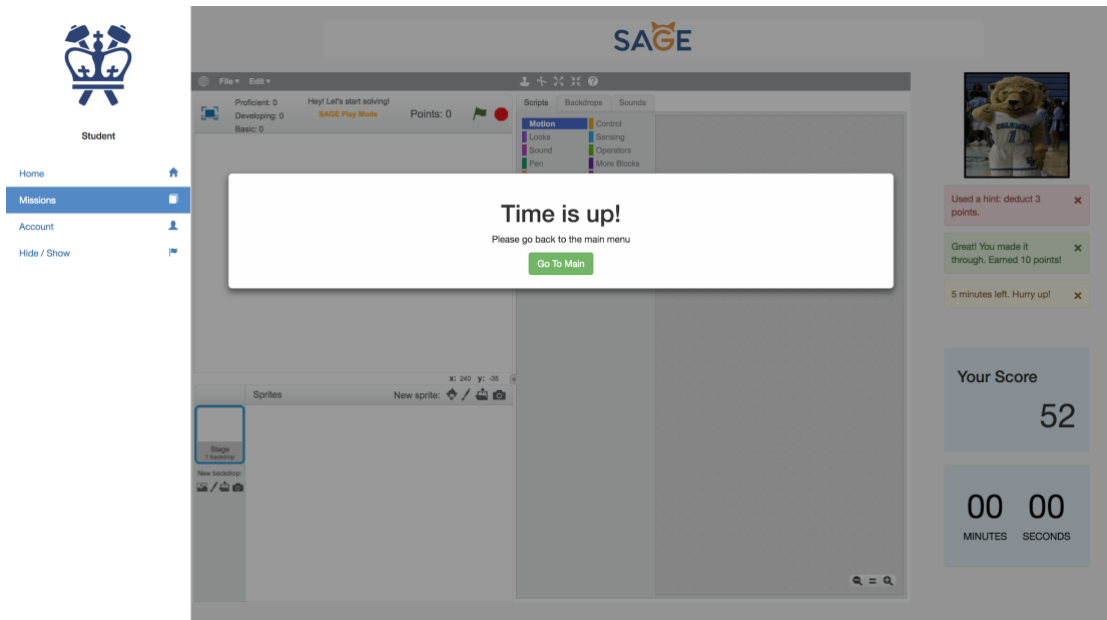
We have changed the layout of the scoreboard and timer section. Now the score is displayed at the outer UI together with the timer. Originally, the timer could be started and ended manually, but after several discussions, now we will let the timer automatically start after the game is loaded so the game can be more fair. Also, the student cannot pause on the game as long as it has started so that it makes more sense as a timed practice. The numeric values of scores and time

are displayed in bold as an emphasis. For now, the time can be fetched from the backend and displayed dynamically.



- Popup panel

We added a pop up panel when the time is up telling the student their session has ended. This increases the interactions between the time and the game, and also prevents the students from making further actions on the game board so that the scores can be fixed. We will only allow them to go back to the main menu after seeing their score.



5. Limitations

- Front-end

For now, we want to implement the real-time score changing using socket.io. As we implemented that, the socket can run in the server side but not worked at Angular side. There is not lots of information about how to solve this problem in Angular 1. So we may need more time to work on that.

Message panel is also static for now. As the socket.io is not working now, the message related to the scoring cannot be shown dynamically.

- **Server**

The server needs to keep the socket connection in memory. A large volume of connection could potentially affect the performance.

In addition, we need to figure out whether the information from the frontend can be sent to the corresponding student client side. Right now we come up with using a global map variable to store the relationship between the student identification and the socket object. However, the concurrency might happen and the same student can be logged in from different places (since right now the authentication part doesn't take care of the double logging issue). We hope we can think out some feasible solutions based on the current situation.

- **Scratch**

Many functionalities are moved to the frontend. Therefore, a number of extra communications with the frontend are required, causing overhead.

Also, we need to check out whether the version of scratch we modify right now is corresponding to the latest version that is given on the cloud.

What's more, we have taken the hint into account in the frontend part, which means that students can click the hint button to get the hint based on the specific assignment. But what we need to take care of is that we need to concern about the final score. It should relate to the times of a student clicking the hint button from the front end. Therefore, after the socket can be established from the frontend to the server, we can implement this functionality into the system so that the scoring system is more reasonable.

Last but not least, we think the current version of the system is not so user-friendly. We want to make some detail modifications to make the user experience better, like adding a new window that can help instructors to configure what blocks need to be used in a specific assignment when they want to set up a new assignment, and so on. But the difficulties of implementation of each detail might vary due to the limitations of database collection definitions and efficiency issue.

5. Future Works

- Front-end

1. Scoreboard implementation

Fix the problem of socket.io in the angular. Make sure it will connect and receive score information from backend and display on the page dynamically.

2. Message panel

After the socket.io worked, the message panel will also display dynamically. Such as when the students do a wrong step, the score will deduct, the message will pop up as the lion said that. Some message about the time will also be implemented.

- Node Server

1. Finishing socket connection establishment for both sides.

We will also try to finish the socket connection between the node server side and the front end part. Thus, we can ensure communication regarding scores and messages.

2. Cleaning up code

We have added some redundant codes and some codes that previously could not work. We will do code cleaning after we finish the socket communication issue.

● Scratch

1. Removing the student answers locally store option.

Since we want to the student result in the database, we don't need to have the local storing functionality. This includes removing the option after we click the file tab in the scratch panel, and provide correct storing when students submit their answers.

2. Cleaning up the code.

3. Block point configuration user story

Basically, we want to pop up a window for an instructor to configure blocks in the parson's puzzle. And the details of the procedure of a teacher configuring a block will be discussed in our following sprint meetings.

4. Testing (depending on the remaining of the time after we implement all functionalities provided above)