

Final Project Report

Parson's Programming Puzzle Scoring System

Junyi Wang
Yilan He
Ningchao Cai
Yiming Guo

CONTENTS

Abstract	3
1 Introduction	4
2 Related Works	5
3 Architecture	6
4 Implementation	8
4.1 UI Improvement	8
4.2 Game Setting Configuration	9
4.3 Parson's Timer	12
4.4 Parson's Submit Logic	12
Time Up	13
Submit button	13
Reach the correct answer	14
4.5 Parson's Submit Explanation	16
4.6 Parson's Scoring	17
4.7 Parson's HUD Styling	17
4.8 Parson's Score Persistence	18
4.9 Block Point Configuration	19
4.10 Parson's Feedback Presentation	20
4.11 Parson's Question/Hint Presentation	21
5 Limitations and Future Work	23
Framework	23
Architecture	23
Scoring Panel	23
Block Point Configuration	23
Load Project from Database	24
6 Conclusion	25
7 References	26

Abstract

Parson's Puzzle is a game which aims to practice students' computational thinking ability. Throughout the semester, we have been focusing on improving the game design of Parson's Puzzle. We spent effort on integrating Parson's Puzzle with the website, stripping down Parson's Puzzle's dependency on the local environment and improving Parson's Puzzle's logic and user experience. The epic we focused on is Gameful Direct Instruction. We have completed all user stories listed in Parson's Puzzle 1.1 feature and some user stories listed in Parson's Coaching feature. The implementation of these features would bring users a cleaner and easier-to-use interface.

1 Introduction

Scratch was first introduced by MIT in 2002. It provides users an interface where they could program interactive stories, games, and animations by simply dragging and dropping the blocks [1][2]. Scratch also provides instructors with an easy-to-use interface to design programming questions for the students. Due to its ease of use, low entry barriers and efficiency in practice, it has been studied and improved widely by educational researchers.

Social Addictive Gameful Engineering (SAGE), proposed by Jeff Bender in 2015, was such a practice to “tooling Scratch with a collaborative game-based learning system that catalyzes teachers and (grade 6-8) students in immerse in computational thinking” [3]. Parson’s Puzzle is one of the main tools to help students practice computational thinking. Parson’s Puzzle is a game with direct instruction. The students would be provided with detailed questions, hints and building code blocks to reach the answer. Instructors would be able to set this information and put restrictions on the game. The game would automatically evaluate the students’ answer and generate a score as a metric for student’s performance.

The SAGE project designs a website where student/instructor would be able to enter all kinds of games. Once they enter a game, Scratch would be loaded as an isolated component and displayed on the website. SAGE has also made a great effort to create an interactive atmosphere to help students learn. An example of this type of attempt is the interactive lion avatar displayed on the website. However, since Parson’s Puzzle, like many other games, resides solely inside the Scratch, the game would not be able to use those interactive features. Also, at the current state of development, Scratch is working on the local environment, while SAGE project is designed as an online platform. Our project aims to solve this inconsistency and our goals are:

- 1) Integrate Parson’s Puzzle with the website. Bring some Scratch game elements to the outer user interface of the website.
- 2) Strip down Parson’s Puzzle’s dependency on local environment. Migrate data storage to database.
- 3) Improve Parson’s Puzzle’s logic. Make the play and design logic more clear and straightforward.

In this report we would describe the user stories that we have completed to achieve these goals. The report would formally describe the technical details of the features added and the motivation behind the implementation. We would also describe the limitations of the current work, as well as future works that could be accomplished.

2 Related Works

There exist some other platforms that show similar functionalities as the SAGE system. The most similar one is the one from *code.org* [4]. The basic functionality of *code.org* is to provide a platform for students to learn knowledge and for instructors to share their courses online, which mostly has the same concepts with the one of SAGE. And in the affinity space, students can drag blocks to finish the tasks assigned by their instructors. However, SAGE platform provides more interesting but helpful functionalities to make the student learning process smooth by adding intelligent hint and student assessment system. These are crucial and should be provided to the students, especially to the group of Grade 6-8 to learn how to write the code, to let them benefit more from the SAGE platform.

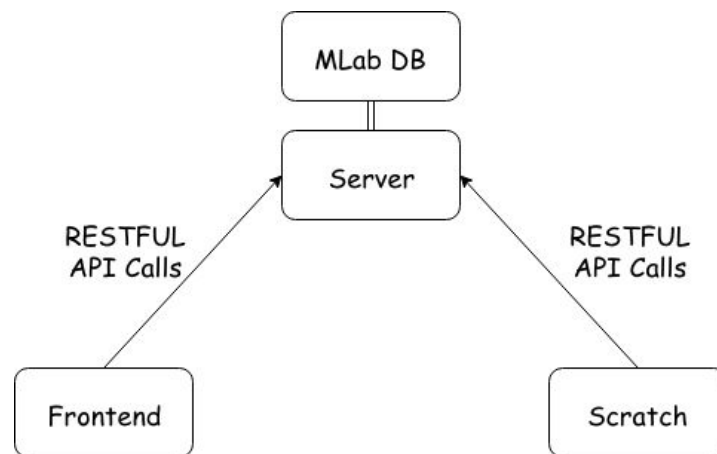
For the Parson's Puzzle part, some systems, such as *Hot Potatoes* [5], implement the idea to help different levels of users learning different concepts. *Hot Potatoes* provides the instructors with ways to build up their Parson's puzzles. However, the user experience of using the provided tool in *Hot Potatoes* is not great. In the SAGE platform, the system uses Scratch as the interface for the instructors to set up their assignments for the targeted students. The version of the SAGE system we implement also stores the information in the cloud, rather than saving the static local file. The different ways of interaction make the process of learning easier and bring convenience to the instructors to provide high-quality programming learning course to the students.

For the Scratch platform, some platforms such as *Scratch Studio* from MIT [6] provide the users with a way of learning various concepts. *Scratch Studio* shows users different courses, including programming courses. And the way of user interaction is better than the one in SAGE platform. But the version of the system we implement gives the students chances to show the self-explanations to their instructor about what they think of in the corresponding assignment and how they are reaching their answers. And the SAGE platform aims at letting students at Grade 6-8 learn how to program. Using Scratch rather than other plain text lectures can incent students to be interested in the process of programming, resulting benefits for both students and instructors.

The system combines Parson's Puzzle with Scratch in the implementation, which brings more interests to the students. Also, since Scratch uses some JSON-like format file (.sb2) to save the answer submitted by a student, the researchers can then retrieve these data to analyze and assess the answer to provide a more accurate hint to the student. This is the part that the SAGE platform differs from others.

3 Architecture

In the current implementation of Parson’s Puzzle, Scratch acts as an independent part which carries out all the logic execution, user interaction, and display. Only a few of its functionalities require one-way communication with the server. For the same reason, Scratch does not communicate with the frontend other than fetching query parameters from the URL. The frontend in most cases only needs to fetch information from the server. Therefore, simple API calls are implemented and used in the current code.



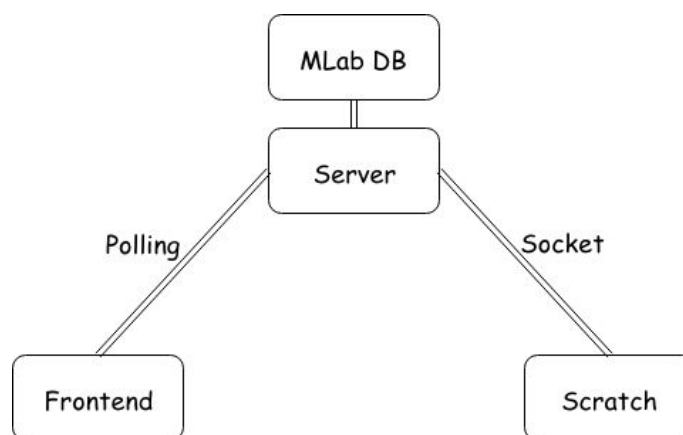
One of our major goals is to improve the user experience and to provide students with straightforward ways to interact with the game. We need to move all game statistics (i.e. point, time and proficient levels) and function buttons (i.e. question, hint and submit) to the frontend. However, this would require a whole new schema of communication between these parts. On one hand, scratch needs to inform frontend of newly generated game statistics. On the other hand, frontend needs to inform Scratch when the buttons are pressed. Since most of these communications contain valuable game information, we decided that instead of connecting frontend and Scratch directly, we use the node server as a medium. By doing this, these game information could be stored for later analysis.

In order to save the game information in the database, we need to set up some basic configurations. The most important part is to design some collections for the system to save the corresponding data in the cloud. In the current system, we add collections such as *student submissions* for storing the result of a student finishing an assignment, *gamestats* for saving the configuration of an assignment which is set up by an instructor, and so on. Storing the game information on the cloud means that each part of the system can interact with each other by using the real data. What’s more, right now most of the user data can be retrieved from the cloud database, the system does not need to retain the functionality of saving related files locally which means more sense based on the scheme.

However, there still are some issues such as real-time data transfer. Since we need to show the score changing process on the frontend page, which means the changing score has been stored in the database, we need to ensure that the data is stored successfully. We think about this part and implement the functionality in our system to resolve the problem. Basically, we use *promise* clause to ensure that we can have next step operation until the corresponding data is saved in the cloud.

Now, Scratch information would be passed to the server, stored in the database and sent to the frontend. Frontend user action would be passed to the server, stored in the database and sent to Scratch. Now we have to deal with the server push problem. It is easy for clients (frontend and Scratch) to send information to the server but it is not trivial to do the opposite. The easiest way to achieve this is polling. But considering the number of users, there might be and various types of data to poll, polling may not be the first choice. The other option would be using sockets. Once the socket connection is established, the data could flow freely in both directions. The concept of Socket is widely used in the online game industry for its stability, efficiency, and scalability.

Compared with traditional polling mechanism, Socket is also a better fit for the data we use, which has the property of small size, large amount and need for timely delivery. From the perspective of code maintenance, socket codes are easier to maintain and to scale. As a result, we chose to implement socket communication. However, due to Angular 1's environment limit, we failed to establish a socket connection between the server and the frontend, so we fell back to using the polling implementation. A socket connection is established between the server and Scratch only. An architecture overview is attached below.



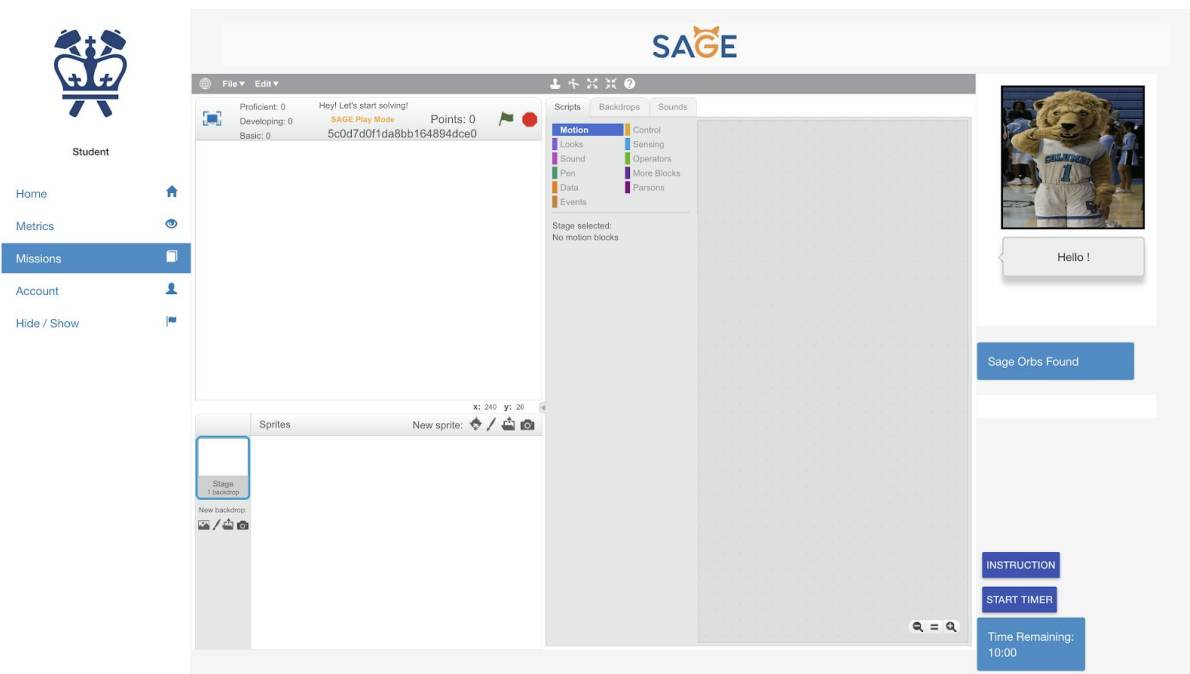
Every time a user enters a game, a socket connection would be established automatically. After that, any game data update or frontend update would be sent using the socket. The socket would be closed and cleaned from the memory once the user exits the game. To overcome the potential deficient of polling, the frontend would only start polling when a predefined event is invoked.

4 Implementation

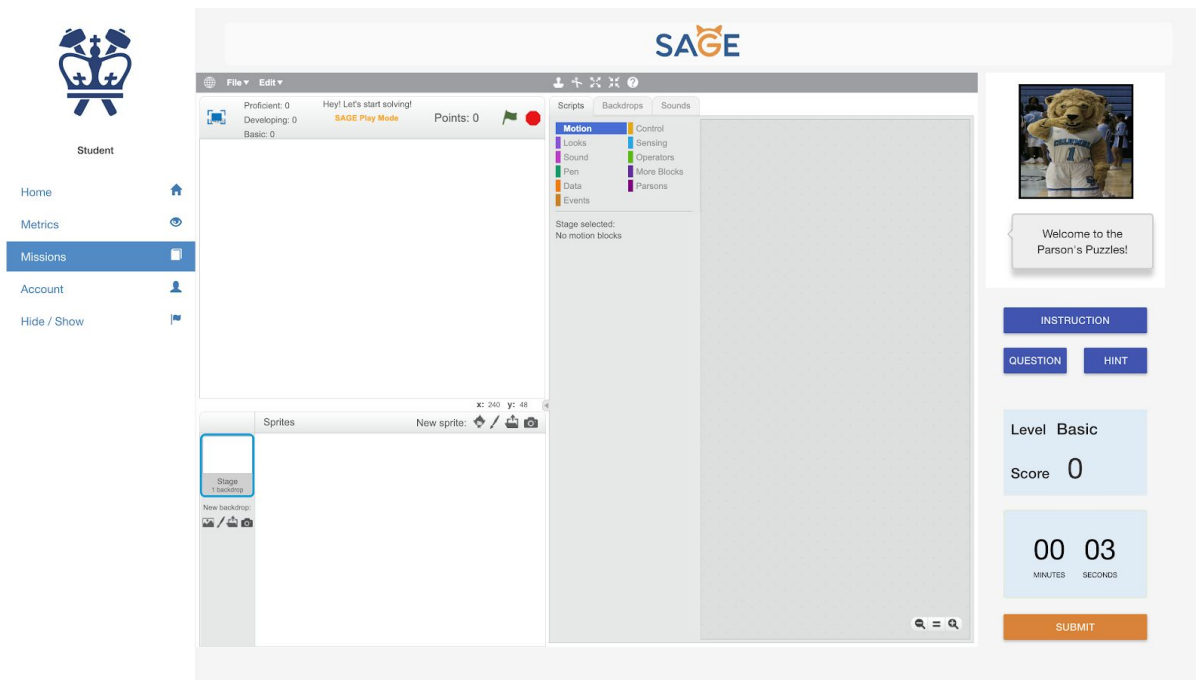
In this section, we will discuss the details of all the system components that we have made an improvement on. We will elaborate the reason for using the new design and how we implemented each of them.

4.1 UI Improvement

Here is the final UI design. We moved several functions from Scratch to the outer frontend, such as hint button, question button, scoreboard, timer board, and submit button. With this design, the website is more user-friendly and appears more obvious for users to see and interact with. In addition, we do not need to use the orbs in the Parson's Puzzle screen which has been implemented in the previous version of the system for all the games. Therefore, we used the provided course information to validate the type of the assignment and hide the orbs block only in Parson's Puzzle. In this way, the student would not be distracted by irrelevant components in the frontend. Below shows the old interface and the new one respectively.



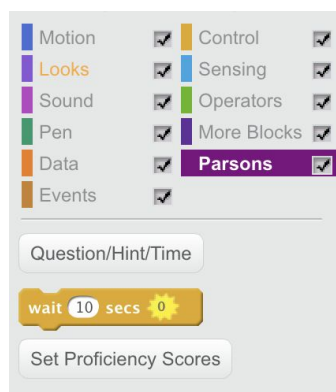
The old UI



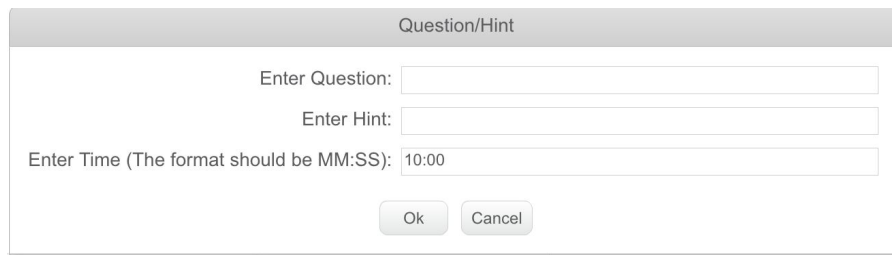
The new UI

4.2 Game Setting Configuration

Parson's Puzzle provides instructors with the option to configure the game setting. The instructor could set the question, hint, and proficiency levels for the game in the design mode. However, this setting was not persisted and only the instructor could use by switching to play mode. We implemented the storage of these settings by saving them in the database.

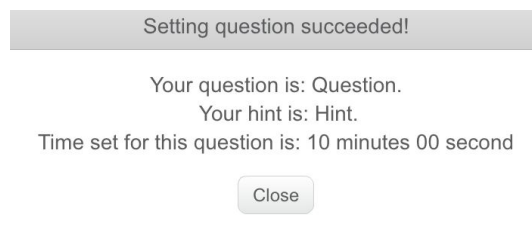


Same as the old interface, the instructor could set those parameters by clicking the buttons. By clicking the Question/Hint/Time button, the instructor would be shown a pop-up window asking for specific parameters.



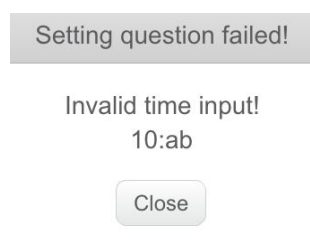
A dialog box titled "Question/Hint" with a light gray header. It contains three input fields: "Enter Question:" with an empty text box, "Enter Hint:" with an empty text box, and "Enter Time (The format should be MM:SS):" with a text box containing "10:00". At the bottom are two buttons: "Ok" and "Cancel".

Time is a new parameter we have added. And its value is pre-populated already with a default value of 10 minutes. Once the instructor entered the value and pressed ok, a confirmation would pop up if the input is valid.



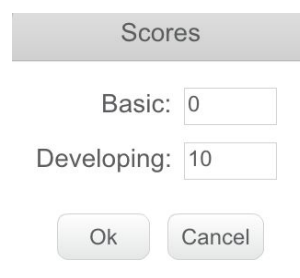
A confirmation dialog box with a gray header titled "Setting question succeeded!". The body contains the text: "Your question is: Question.", "Your hint is: Hint.", and "Time set for this question is: 10 minutes 00 second". A "Close" button is at the bottom.

If the instructor entered an invalid input, an error window would be popped up. Therefore, we can always verify that the time input is able to use.



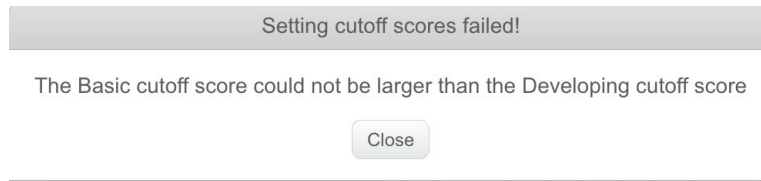
An error dialog box with a gray header titled "Setting question failed!". The body contains the text: "Invalid time input!" and "10:ab". A "Close" button is at the bottom.

Similarly, the instructor could set proficiency levels for the game by clicking “Setting Proficiency Scores”. Only Basic and Developing levels are required to be entered since the Proficient level cutoff score would be calculated by Scratch automatically.

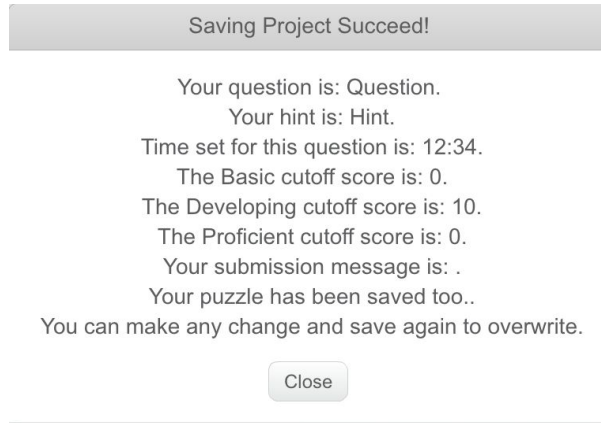


A dialog box titled "Scores" with a gray header. It contains two input fields: "Basic:" with a text box containing "0", and "Developing:" with a text box containing "10". At the bottom are two buttons: "Ok" and "Cancel".

Again, there is some basic checking for the proficiency scores as well. If the Developing score is lower than the Basic score, an error would be popped up.



Setting proficiency levels would be the last step before the entire game configuration is saved. If every entered value is valid, a final confirmation would be popped up.



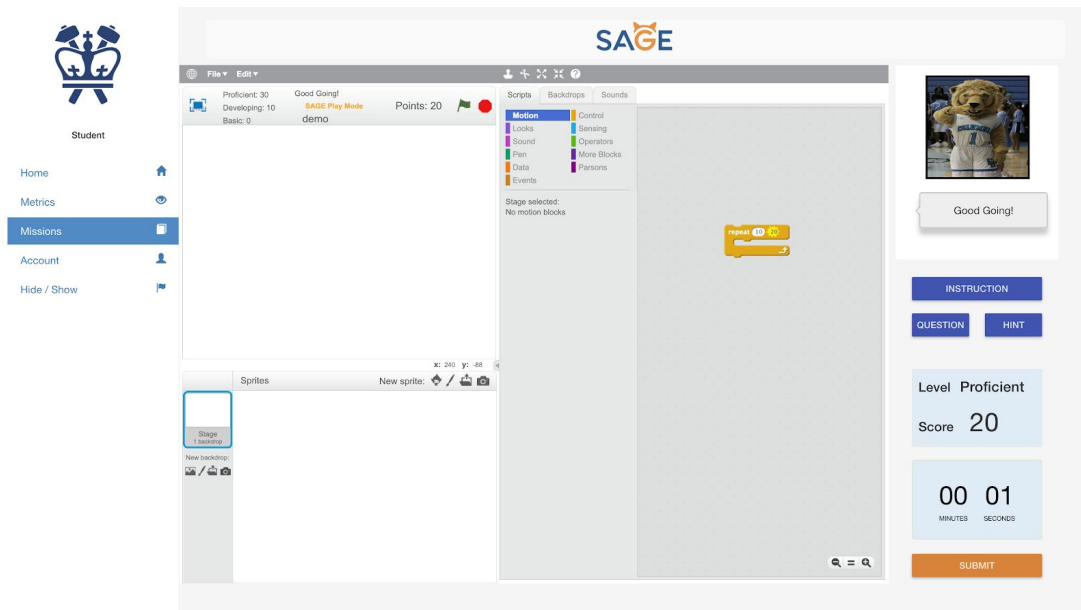
At this moment, Scratch would send all these game settings, along with the *assignmentID* to the database. The setting would be stored in the *gamestats* collection. Every time a student loads a game, the game would check whether the database has an entry for the *assignmentID*. If there is, then the game setting would be loaded into the current game.

Game Setting API

GET: /courses/:courseID/assessment/:aid/:sid

The frontend will get all the game setting information via the API call above. Thus, the saved game configuration information which includes assigned time, question, hint, and proficiency score levels will be fetched via *assignmentID*. After that, this information will be saved as variables and displayed at frontend.

4.3 Parson's Timer



As mentioned before, the timer setting is a newly introduced feature. In the first half of the semester, the timer implemented in the frontend was too small to see. The timing mechanism was not proper since it required the student to click the “start timer” button to start the whole game. The “start” button would become “pause” once the game began. Though the student might have the freedom to toggle the game timer, it was not useful when coming to the stage of fairly evaluating the student’s work with others since they might be completing the game within a different amount of time.

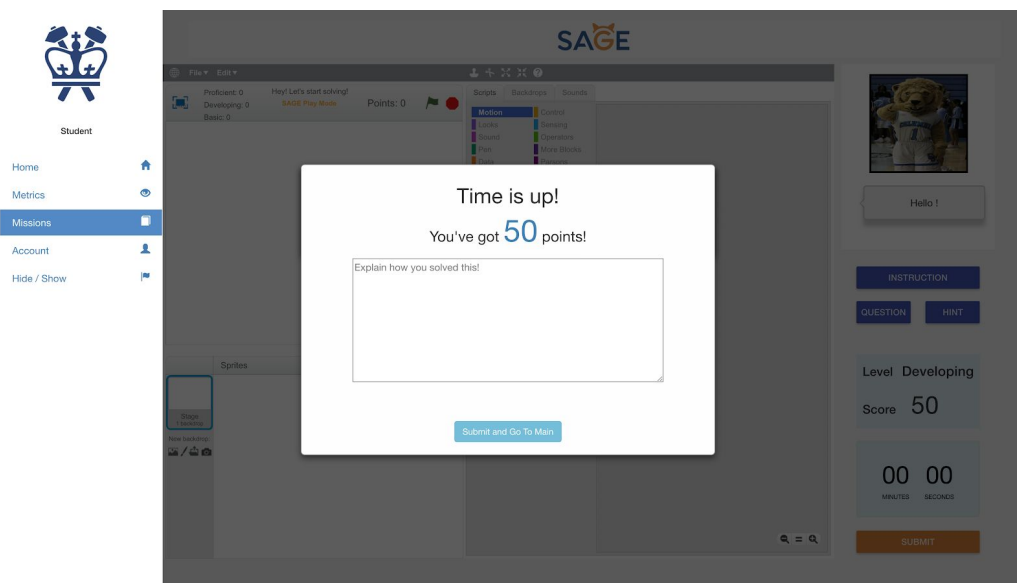
Therefore, to improve the current design, we remade the timer block and add more contrast by enlarging the dark numbers and using a light-blue background. So the timer will be more clear to see especially when it counts down and changes. Regarding the timing mechanism, the current timer will automatically start when a student opens one puzzle to solve. When the time is up, the result will be automatically submitted. Thus, it could be fairer to ensure all the students complete the game within the same time limit since they are forced to start and end the game without intervening the time by pausing it.

4.4 Parson's Submit Logic

There are three different ways to trigger the submission event:

1. Time Up

When a student opens the game, the timer will start automatically and when the time runs out, a popup window would be displayed on a blacked out background so the student would not have the access back to the Scratch game. Then, the student would be asked to enter self-explanation. Once a self-explanation is provided and the student clicks “Submit and Go to Main” button, the frontend would notify Scratch to submit.

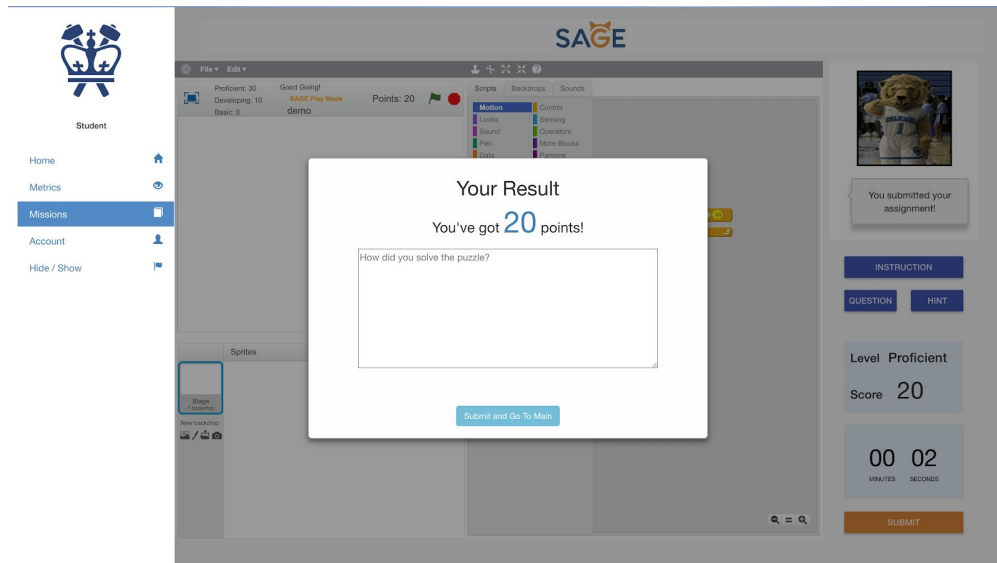


2. Submission Button

During the game, the student can click the submit button to manually submit his/her assignment whenever they want. Under this circumstance, the popup window will show “Your Result” and the final score the student gets. The message box will also show the information that “You submitted your assignment!”.

With this functionality, the student does not need to wait for time up to submit when he/she cannot solve this question or he/she reaches his/her limit and wants to prevent further point deduction.

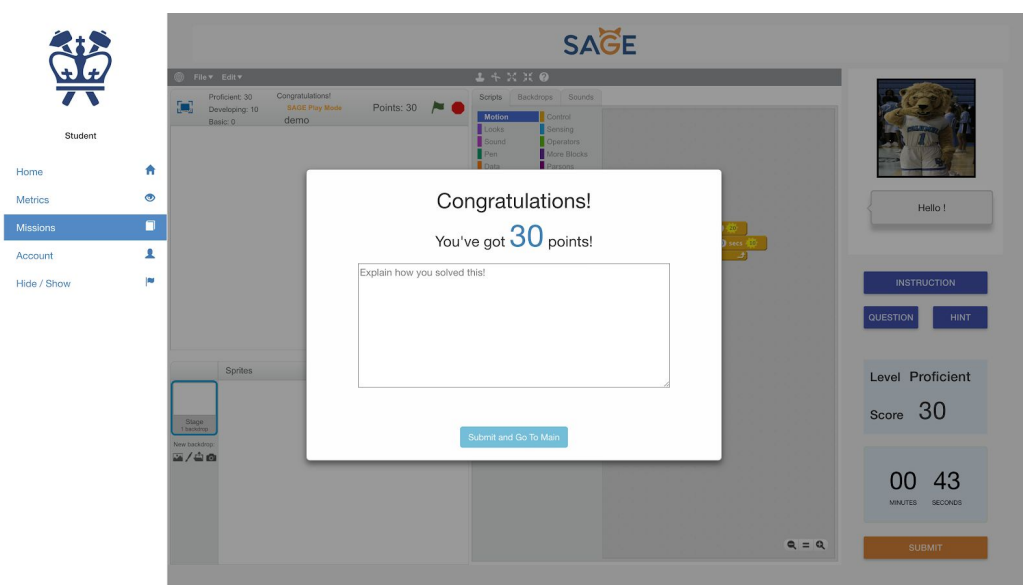
The submit button is designed in orange color and at the bottom right corner, which is easier for the student to see and reach, which satisfies Fitts’s Law.



3. Reach Correct Answer

When the student is playing the game, if he/she reaches the correct answer, Scratch will automatically initiate the submission event and the popup window will show “Congratulations!” and display the final score. We added this functionality since we believe this will make the problem solving process more like a game. Also, there is no need for the student to submit manually when they reach the correct answer since the feedback would give away a student’s progress.

To evaluate whether the student reaches the correct answer, we modified the *parsonsLogic* function in Scratch. Every time a student moves the block, we would compare his/her answer against the instructor’s.



Auto-Submit API

POST: /courses/assessment/:aid/:sid/autosubmit

To inform the frontend when a student has reached the correct answer, the backend would do a post and save a combination of *assignmentID* and *studentID* (*'aid-sid'*) in a set called *submitset*.

GET: /courses/assessment/:aid/:sid/autosubmit

To check whether a student's work has been submitted automatically from the backend, we will check the current combination *'aid-sid'* within the *submitset* mentioned above and poll this information every second. If we find it in the set, the frontend will pop up the submission window automatically to display a congratulation message and ask for the student's input for self-explanation.

Final Submission API

POST: /courses/:courseID/assessment/:aid/:sid/timeup

Any of the above three submission types will trigger the pop-up window. Final submission occurs only after the student enters the self-explanation in the pop-up window and click the "Submit and Go To Main" button. After that, the frontend will send a post request with the self-explanation information so the server can further process it by saving this record in the database.

When the actual submission takes place, Scratch would first calculate the score for one last time, send game information and the game itself (sb2 file) to the database *studentsubmissions* collection. A sample submission payload would look like this:

```
{
  "_id": {
    "$oid": "5c17f74258527c70ec26e708"
  },
  "startTime": "1545074489726",
  "score": "0",
  "hintUsage": "0",
  "remainingSeconds": "591",
  "submitMsg": null,
  "endTime": "1545074498265",
  "studentID": "59f369cc748499467c32a414",
  "assignmentID": "5c0f311e1c2ea6a9cf105cd1",
  "objectiveID": null,
  "selfExplanation": "that's how I get the solution",
  "blocks": [],
```

```

    "sb2File": binary sb2 File,
    "__v": 0
  }

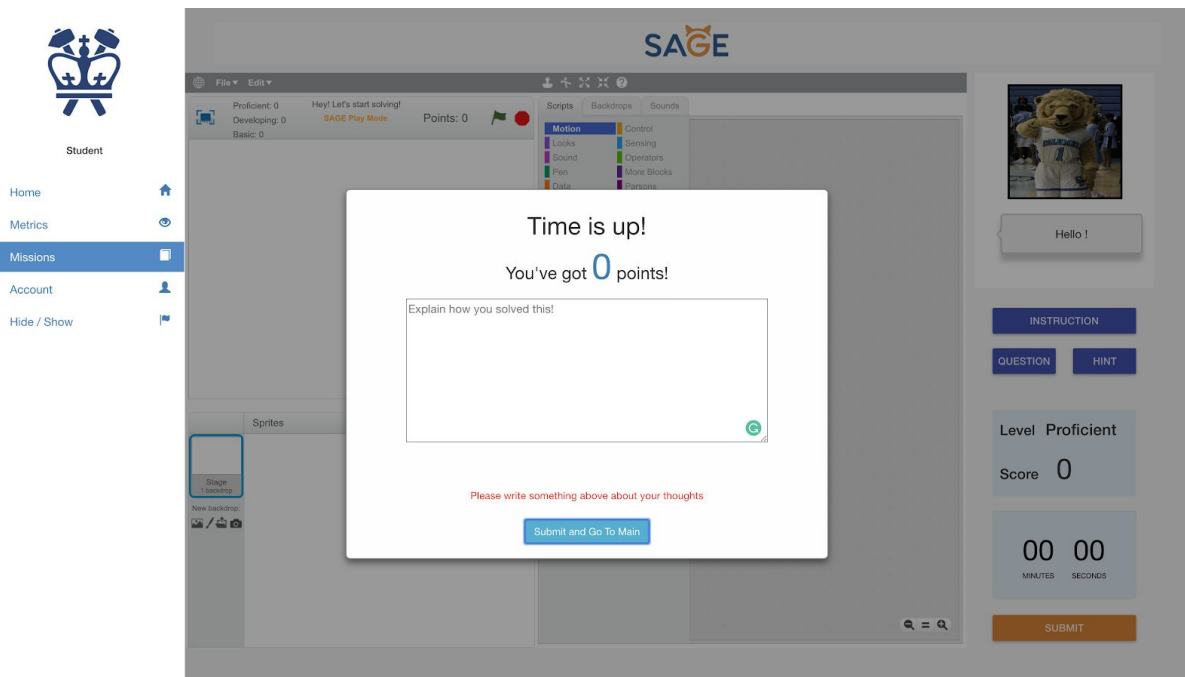
```

4.5 Parson's Submit Explanation

Previously, the self-explanation is prompted when student clicks the submission button. The self-explanation would be collected using a small dialog box, which is not quite user friendly. Also, since we moved the submission button to the frontend, it would be more natural to collect self-explanation directly on the frontend. We also made it mandatory for the student to provide this self-explanation information.

Empty Check

When the student clicks the submission button without the feedback or explanation, he/she cannot proceed on, and a warning message in red will show to alert them to fill in this field.



4.6 Parson's Scoring

We made some modification to the scoring algorithm. To begin with, the points deducted for moving an incorrect block would be associated with that block's point. Secondly, if the student moves a distractor (a block that is not in the solution), the penalty would be doubled. Also, a "You selected a distractor block" feedback would be provided to the student.

Point is the most important metric in the game. Previously, the point was displayed on Scratch, which is not obvious at all. To improve, we moved the scoreboard from scratch to frontend and made it bigger so students could notice the score changing easily. Every time

Scratch calculates a new score, that new score would be sent to the server and server would push the new score to the frontend to display. To adopt the same method for increasing the contrast as the timer, we used the black big font and the light blue background for the scoreboard as well.

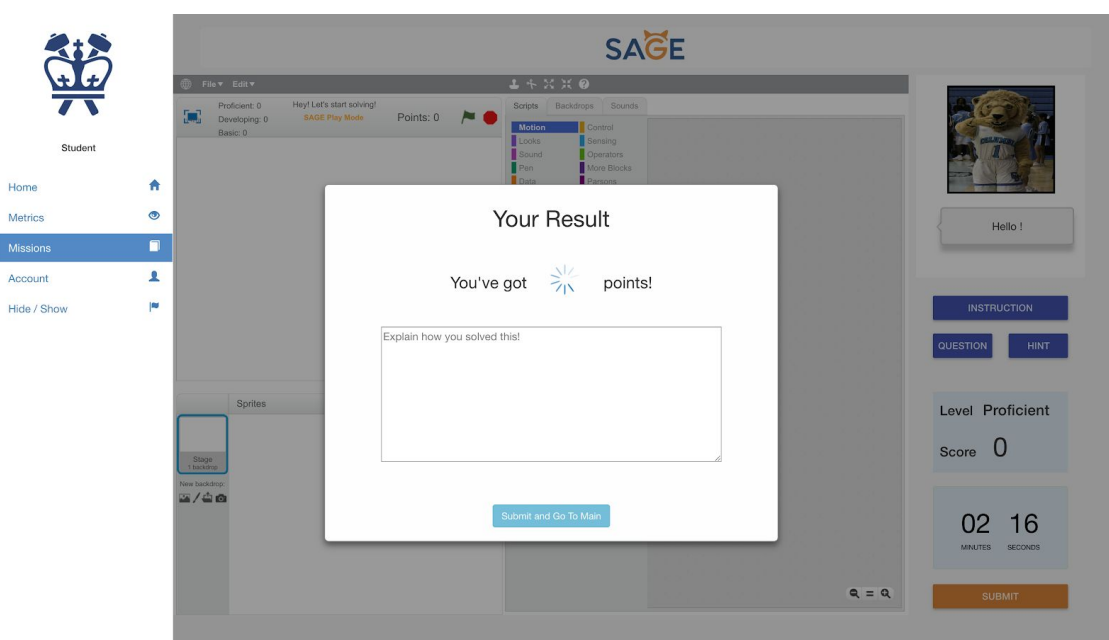
4.7 Parson's HUD Styling

We moved the proficiency levels from Scratch to the frontend. Previously, the proficiency levels are simply listed on Scratch. Students need to do comparison themselves to know how good they are. We believe that this is not user-friendly. The students may not understand what those scores are for at all. And it is not necessary for students to know the specific cutoff of each level. In order to improve user experience, we moved proficient levels to the frontend, right above the score. Instead of giving out the cutoff scores, we only display a student's current proficiency level. If the student's score exceeds or drops below a cutoff, the proficiency level would be updated automatically.

4.8 Parson's Score Persistence

Loading Animation

After submission, it takes time to read the final score from the database. To ensure a consistent view of the student's final score, during this fetching time, a circling animation will show at the position of the score which will tell the user to wait for the final score. When we fetched the final score, which will be indicated by a field called *isFinal*, we will replace the loading icon with the actual and final number that the student earns.



Score Fetching API

GET: /courses/:courseID/assessment/:aid/:sid/score

To update the score in real-time, we do polling on the scores every second. To indicate the final score, we have an additional attribute called *isFinal*, which can indicate the frontend whether this score is a final submitted one. The fetched score will be displayed on the score panel in real time. If anything triggers the popup window, the score section will show a loading icon until it gets the final score.

4.9 Block Point Configuration

Previously, the instructor could configure points for the blocks and save to or load from the local filesystem, which is not optimized for SAGE. To begin with, as SAGE project aims to provide an online platform, we should avoid local IO as much as possible. Saving the configuration into the database would make the usage more flexible. Additionally, point configuration is a commonly used handy configuration. The point configuration could be reused for several assignments and a carefully designed configuration could be shared in the community. Therefore, we have the motivation to create a point configuration library, where the users could share their configurations.

To achieve this purpose, the first step would be storing the point configuration online. We provide instructors with options to name the configuration so they could search for the configuration easily later. If the instructor clicked “Save Point Configuration” from the menu dropdown, a popup window would be displayed asking for configuration name.



The configuration name needs to be unique to the user. But it could be duplicated across different users. Once the user picks the name, Scratch will check the server for name duplication.

Point Configuration Exists

The point configuration name you entered: points_config.json has been used for this game. Do you want to overwrite?

OverwriteGo back

If the name exists already, a popup window would ask the user whether he/she wants to overwrite the existing configuration. If the user chooses to overwrite, the point configuration would be sent to the server in JSON format. The point configuration would be stored in *assignmentpointconfigs* collection. A sample entry is provided here:

```
{
  "_id": {
    "$oid": "5c0fc619530c94adb7eb6eef"
  },
  "id": "59f8c6fdc1bfb23c4ced8e20",
  "configName": "demo_point_config",
  "pointConfig": "{\n\t\t\"wait %n secs\": 10,\n\t\t\"repeat %n\": 5\n\t}",
  "__v": 0
}
```

We only associated the configuration with the instructor’s ID, since the configuration is not tied to any specific assignment.

When the instructor wants to load point configuration, he/she could click the “Load Point Configuration” button in the menu dropdown. Then he/she would be provided with all the point configurations available to him/her.

Load Config List

points_config.json by you

demo_point_config.json by you

demo_point_config by you

another_point_config by 5bd214be4b407f0d58ad54d5

yet_another_point_config by 5bd214be4b407f0d58ad54d5

Prev PageNext PageCancel

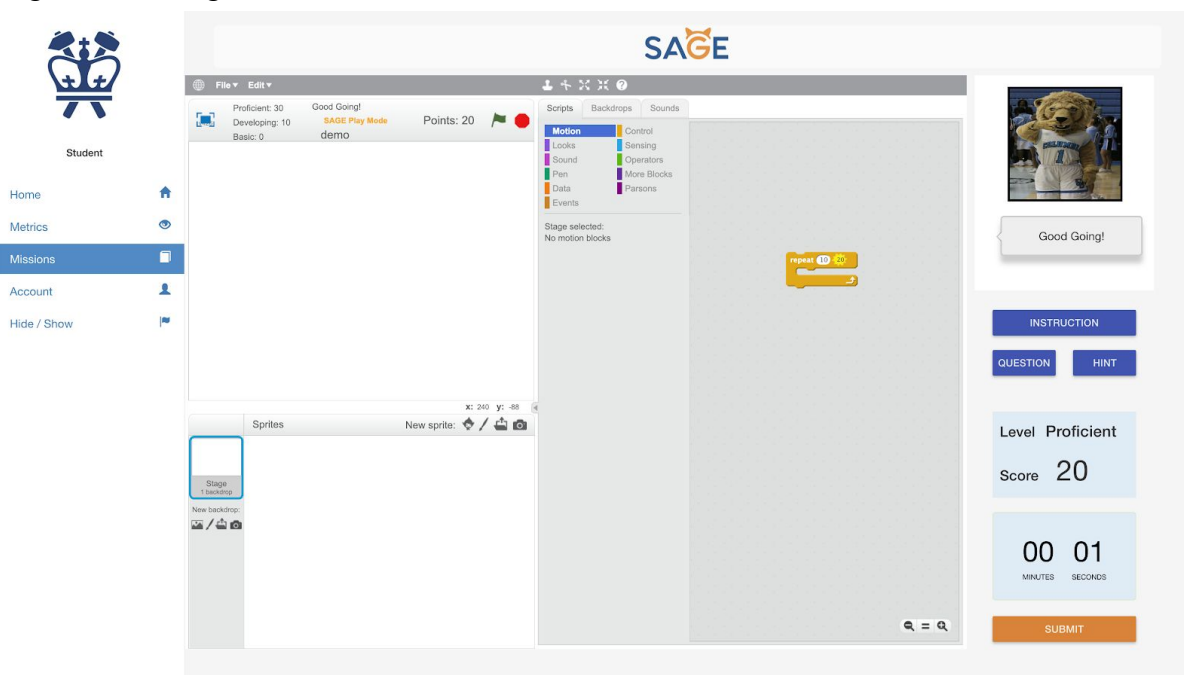
The available configuration would be displayed in the format of “*configName by userID*”. If the configuration is created by the user, it would be shown as “configName by you”. If the list contains a large number of configurations, the instructor could navigate by clicking “Prev

Page” or “Next Page” buttons. The instructor could select the desired point configuration by simply clicking the item in the list. After the click, the point configuration would be applied automatically on the current blocks.

4.10 Parson’s Feedback Presentation

Lion Dialog Box

When the student is working on the assignment, some feedback will show in a dialog format and appear under the lion. This will make the lion appears to be talking with the student. Thus, having the words said by the lion helps increase the student’s interest, makes the assignment more gameful and interactive.

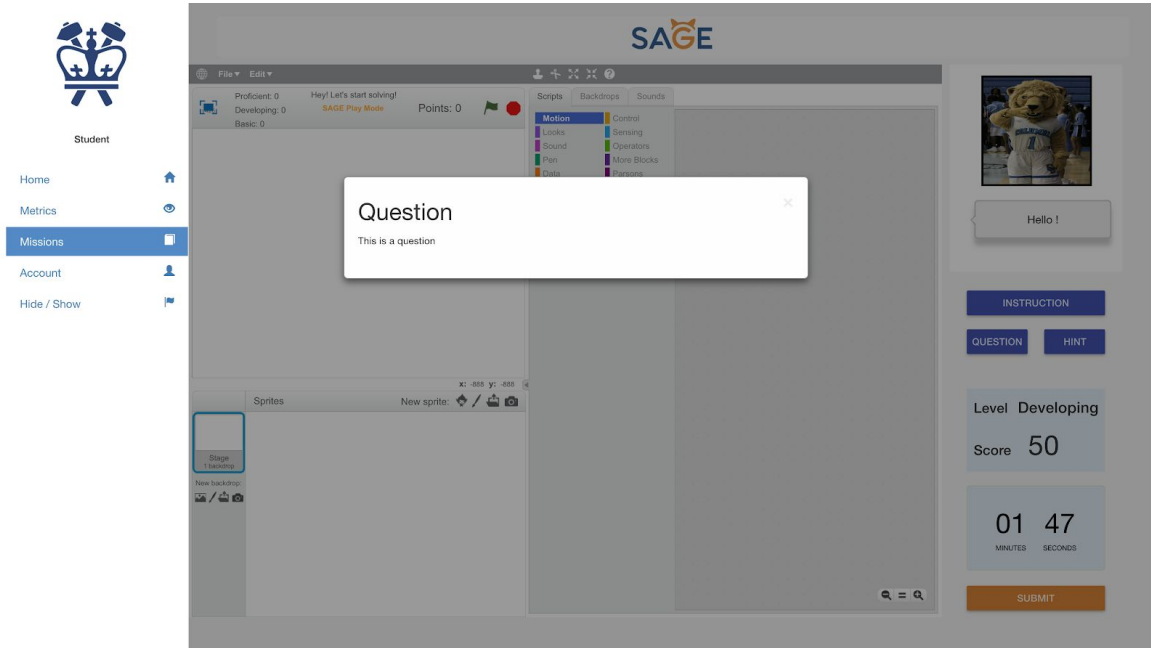


4.11 Parson’s Question/Hint Presentation

Previously, the question and hint button located at the Parson’s palette, which only possesses around one-fifth of the game panel. The buttons are relatively small and the popup windows are not noticeable. To improve the user experience, we would like to move these two buttons from Scratch to the frontend, so we could apply styling to them. Below we present the newly designed buttons. As you can see, the buttons located on the right side panel of the game, with a blue background. It would be easier for students to notice and locate them. Since sometimes the question and hint might be long, and the space of outer UI is limited, we decided to use a pop-up window to display this information so that the length would not be a problem for display. Once the button is clicked, an eye-catching pop-up window would be shown. The game page’s brightness would be turned down, so the student’s attention would be directed to the pop-up window.

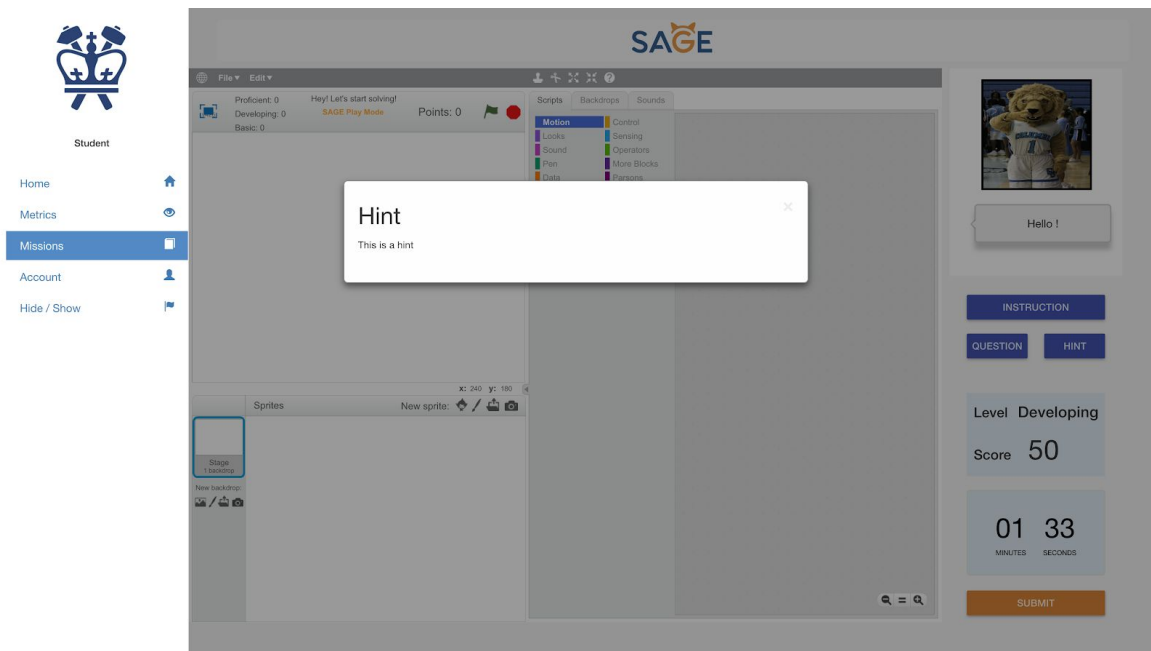
Question Button

When a user clicks this button, a pop-up window shows the description of this assignment.



Hint Button

When a user clicks this button, a popup window will show the hint from the database associated with this assignment.



The frontend will notify Scratch of the hint usage of the student, and Scratch will update the score and send the updated score to the frontend. The frontend will communicate with the Scratch backend for updating the student's score using the following API.

Hint Usage API

POST: `/courses/:courseID/assessment/:aid/:sid/hint`

Whenever the student clicks on the hint button, the frontend will notify the server to do a calculation of the student's score in terms of the hint used. The server will thus store a new score into the database. So by the next second of score fetching, a new score which reflects the hint usage will be shown back to the frontend.

5 Limitations and Future Work

5.1 Framework

The current framework used in sage-frontend is a little outdated. The Angular 1 framework creates large resistance in using a new library. We recommend the usage of AngularJS in the future so that library such as SocketIO and some fancy animation could be used, which could largely improve the user interface and performance.

Currently, Scratch is written in ActionScript which has a small developer community. This creates lots of trouble during debugging and the number of libraries could be used is limited. We highly recommend migrating to Scratch JS when available. This will allow the usage of SocketIO and most importantly, allow better integration with the website.

5.2 Architecture

The architecture is greatly affected by the framework we choose. With the current framework, little improvement could be made. But if we decide to move on later frameworks, we would recommend the usage of SocketIO. Ideally, the frontend, server, and Scratch are connected using SocketIO. This will largely boost the performance and make the code easier to maintain.

5.3 Scoring Panel

Currently, the scoring metrics are shown on a separated page so the student has to manually check that section. We have come up with an idea to show the metrics immediately after the student finishes a game with their score displayed on the pop-up submission window. Thus, a student's score could be integrated with the scoring metrics to give them an idea of their performance at a simple sight.

5.4 Block Point Configuration

Currently, the point configuration lists are displayed in Scratch. Since Scratch is written in ActionScript, the styling is greatly limited. In the future, Save/Load Point Configuration could be moved to the frontend, so the display is more natural and better looking.

Also, right now the displayed configuration is using user's id as part of the name, which is not user-friendly. In the future, we should replace that id with username or email address. But to achieve that step, we first need to have a database collection recording the mapping from user id to username/email address.

5.5 Load Project from Database

Due to time constraint, we did not implement loading project from the database. It should be similar to loading point configuration from the database.

6 Conclusion

In this report, we show our main work in Fall 2018 regarding the Parson's puzzle part in the SAGE platform. We first mention some background and related works in the first part of our report. After that, we mention the revised architecture of the current version of the system.

In the whole semester, we develop the system by using the Agile software development methodology. We use Trello and TFS backlog as the requirement reference for our sprint meetings which are held twice a week.

The major changes are shown as followings:

1. We successfully establish the connections between the frontend, the node server, and the scratch component.
2. We bring some Scratch functionalities to the frontend and improve the user experience by implement some basic UI design principles.
3. The data is stored online, and some collections in the database are designed based on the functionalities of the corresponding parts.
4. The scoring mechanism becomes more reasonable since we take some factors in the process of a student finishing an assignment into account.
5. We improve the user experience for the users with the instructor identities. They can now set up their assignment from their portal and save the configurations in the cloud.

The improvements mentioned above are the ones that we mainly focus on during the whole semester. Some other small details are also mentioned in the previous sections. While we make some significant improvement of the system, some limitations are discovered and remain to be solved in the future work. More potential enhancement could be made on the whole system.

7 References

- [1] Parson's programming puzzles: a fun and effective learning tool for first programming courses, Parsons, Dale & Haden, Patricia. (2006)
- [2] Scratch Programming for Teens By Jerry Lee Jr. Ford (2009)
- [3] Tooling Scratch: Designing a Collaborative Game-Based Learning System to Infuse Computational Thinking within Grade 6-8 Curricula, Jeff Bender (2015)
- [4] Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. Computers in Human Behavior, 52, 200-210.
- [5] <http://web.uvic.ca/hrd/halfbaked/>
- [6] <https://scratch.mit.edu/>