

Increasing the Variety of Problem Designs and Human Computer Interaction of the MathSpring Intelligent Tutoring System for Students

An Interactive Qualifying Project Report
Submitted to the faculty of
WORCESTER POLYTECHNIC INSTITUTE

By

Paris Lopez

Approved: _____
Professor Ivon Arroyo, Project Advisor

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see <http://www.wpi.edu/academics/ugradstudies/project-learning.html>

Table of Contents

Abstract	2
1. Introduction	3
2. Background Research	5
2.1 Assessments in America	5
2.2 Past Research of MathSpring.org	6
2.3 Improvements of MathSpring.org	7
3. Methodology	9
2.1 Definition of Task	9
2.2 Programming Language	10
2.3 Method of Classification and Development	10
4. Results	12
3.1 Results of Classification	13
3.2 Process of Development	19
5. Discussion	40
References	42

Abstract

The goal of this research study is to investigate and develop a new variety of problem types for the online intelligent tutor MathSpring.org as opposed to the more traditional multiple choice or short answer items currently available. During this study, I also looked to understand whether this new variety of problem type design can assist students in comfort and familiarity with the problem types and improve education and standardized test taking ability. Throughout this project, I researched seven designs for new problems including Drop Down Menu, Coordinate Plane, Check All that Apply, Make a Model, Number Line, Drag and Drop, and Construct an Equation. Of these seven problem types, I developed the first five with the application of HTML, CSS, and ReactJS. By creating one or two problems of each type, future developers are able to apply my work to the pre-existing math content authoring tool.

1. Introduction

The goal of this research study is to investigate and develop a new variety of problem types for the online intelligent tutor MathSpring.org as opposed to the more traditional multiple choice or short answer items currently available. During this study, I also look to understand whether this new variety of problem type design can assist students in comfort and familiarity with the problem types and improve education and standardized test taking ability.

This is an important development for Mathspring.org because as the presence of computers and online testing pervade the classroom, schools should be able to provide practice for standardized testing in a format similar to the exams. By developing a new environment to match the updated testing environments, students will be able to not only learn the necessary material in an exciting way but will become experienced in how to interact with new forms of designs of the exam, demonstrating their math skills in a variety of ways, and learning new ways of interaction with math learning and assessment activities. Similar to SAT preparation courses, the advancement of problem design on Mathspring.org presents an opportunity for students to gain a deeper understanding of the types of problems that will appear in high-stakes tests.

In order to understand whether this new variety of problem type designs available on the intelligent tutor MathSpring.org can assist students in comfort and familiarity with the problem types and improve education and standardized test taking ability, I first gathered a variety of novel types of problems available in computer-based statewide standardized tests, in particular the Massachusetts Comprehensive Assessment System (MCAS) to replicate for Mathspring.org. With a folder of past MCAS problems to base my work on, I developed multiple new templates for different kinds of question types that require a variety of forms of human-computer interaction.

In order to answer the question about whether this would be feasible, I developed these problem types: “Drop Down,” “Coordinate Plane,” “Check All,” “Make a Model,” and “Number Line.” Respectively, the problems include one or multiple drop down menus, a coordinate plane on which to place a point, boxes to check all that apply, the construction of a shape divided into equal parts, or a number line on which to mark a point. The next sections will better describe the development of these problems, their use as templates, the results obtained, and conclusions drawn from this research study.

More specifically, the GOALS of this research study are to:

- Investigate the most current types of math problems used by standardized tests within the current years, specifically MCAS and National Center for Education Statistics (NCES), as opposed to the more traditional multiple choice or short answer items.
- Develop these new kinds of problem designs for MathSpring.org, generating new opportunities for student interaction with computer tutoring systems, and maybe deeper thinking while taking tests;
- Start to understand whether this new variety of problem type designs available on the intelligent tutor MathSpring.org can assist students in comfort and familiarity with the problem types and improve education and standardized test taking ability.

The next sections will explain the research method, the results obtained, and conclusions drawn from this research study.

2. Background Research

2.1 Assessments in America

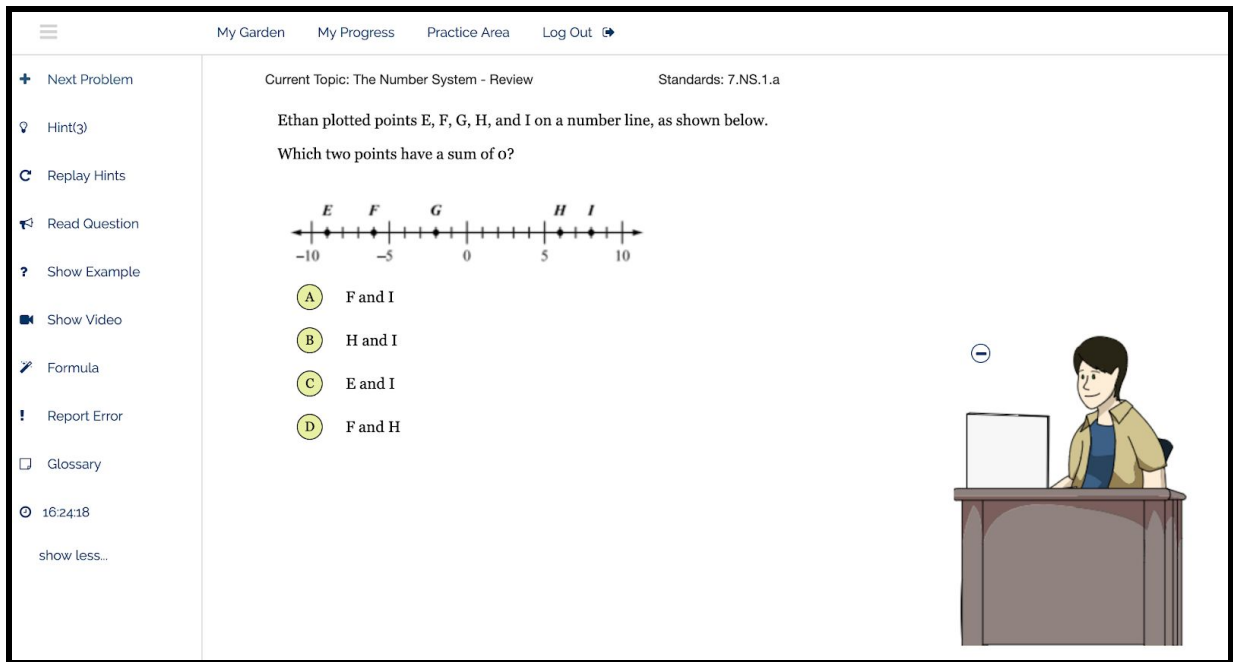
Beginning with the No Child Left Behind Act of 2001, state and national assessments within schools direct students and teachers toward setting high standards and establishing measurable goals. These educational directives aim to bring all students to a particular proficiency levels, especially in subjects such as math and reading. As the presence of digital technology grows in classrooms, educational departments transfer various forms of high-stakes testing from paper-based to digital formats. In 2003 a validity studies panel of one national assessment, the National Assessment of Educational Progress (NAEP), evaluated the implications of electronic technology in the testing of students and addressed concerns stemming from the growing use of devices in the classroom (Durán, 2003). Within years of this study, the NAEP “began piloting digital reading and math assessments in 2015 by having small groups take the online reading and math assessments while the majority of students were still using the old format” (Jacobson, 2017). As of 2018, not only are basic reading and math a part of online assessments, but the NAEP introduces civics, geography and U.S. history tests to the digital world (Jacobson, 2017).

In addition to national assessments, state testing is being transferred to an online format. The Massachusetts Comprehensive Assessment System (MCAS) has been moved to a digital test with online practice problems and final test scores available on the website (Massachusetts Department of Elementary and Secondary Education, 2018). Similar to NAEP, MCAS provides an online format for reading, science and math high-stakes testing to evaluate the level of proficiency of students throughout Massachusetts. As these assessments continue to move onto the web, the level of interactivity and variety of problems become more expansive. With a

developing education system, the tools one uses to support students at this time should reflect the assessments testing those students.

2.2 Past Research of MathSpring.org

A supplementary tool for mathematical high-stakes assessments, MathSpring.org provides students with an intelligent system to practice their skills and refine their knowledge. Currently, MathSpring.org supplies only short answer and multiple choice questions which is inconsistent with the level of human-computer interaction presented through the previously mentioned MCAS and NAEP. While it is currently expanding to not only new topics but new languages, MathSpring is limited in its user interface by not providing other methods of expressing the answer to a mathematical problem.



The screenshot shows the MathSpring.org user interface. At the top, there are navigation links: "My Garden", "My Progress", "Practice Area", and "Log Out". The main content area displays the current topic as "The Number System - Review" and the standard as "7.NS.1.a". The question text reads: "Ethan plotted points E, F, G, H, and I on a number line, as shown below. Which two points have a sum of 0?" Below the text is a number line with tick marks from -10 to 10. Points are plotted at: E (-10), F (-7), G (-3), H (5), and I (7). Below the number line are four multiple choice options: A) F and I, B) H and I, C) E and I, and D) F and H. On the left side of the interface, there is a sidebar with various tools: "Next Problem", "Hint(3)", "Replay Hints", "Read Question", "Show Example", "Show Video", "Formula", "Report Error", "Glossary", and a timer showing "16:24:18". A "show less..." link is also present. On the right side, there is an illustration of a student sitting at a desk with a laptop.

Figure 1. A multiple choice question from MathSpring.org

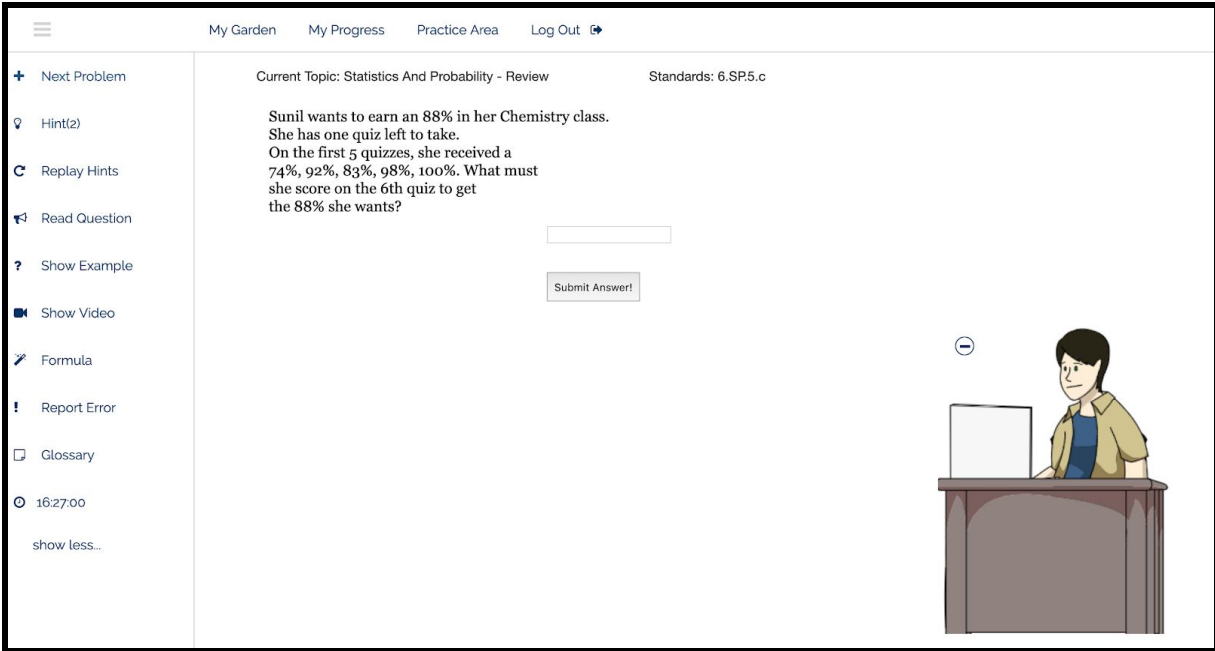


Figure 2. A short answer question from MathSpring.org

Past studies of MathSpring have discussed an upgraded version of the user interface to benefit students and teachers. The researchers from “Improving User Interface and User Experience of MathSpring Intelligent Tutoring System for Students” study the benefits of improving the system through the landing page, a place for students to access logins and forms; the login page, a place for students to either register or login; and the dashboard, where students may access their problem progress. As an extension of this research, the user interface of the problems themselves can also be updated to improve the learning experience of the students.

2.3 Improvements of MathSpring.org

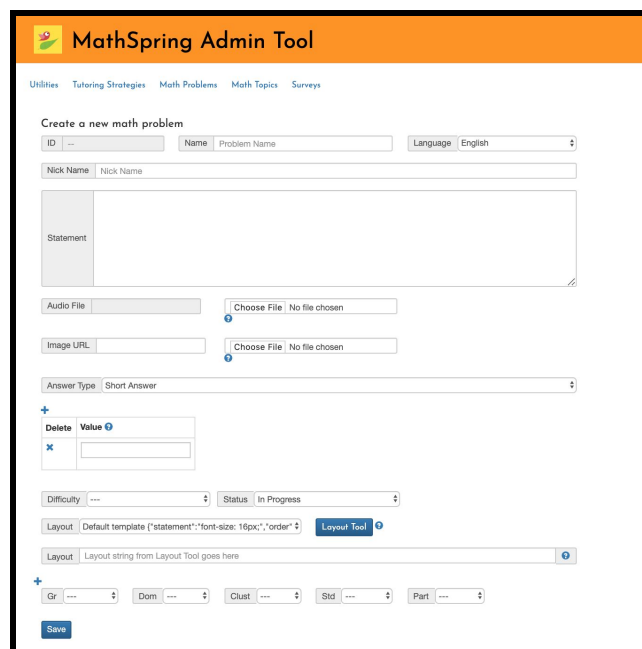
While the study succeeded in introducing a new interface to MathSpring.org, improvements to the websites can continue in the form of interactive problems. As previously

mentioned, the current multiple choice and short answer problems can be augmented by modelling them like those found on the MCAS or NAEP websites. These organizations constantly release problems from past examinations that allow teachers and students to review the structure and content of the questions. By applying these problems to MathSpring.org, the user experience can improve the testing experience for students everywhere.

3. Methodology

2.1 Definition of Task

This project aims to provide a variety of new problems to Mathspring users which will better prepare them for high-stakes testing. To complete this goal, I considered which programming language and libraries to utilize and the design of these new problem types. After evaluating the design of these problems, I created multiple versions of a problem type to find similarities in the code. This process will allow other developers to extract these similarities and add them to the problem designing tool. Already available for short answer and multiple choice problems, rather than a math content developer programming a new problem each time from scratch, a problem authoring tool (currently at: <http://rose.cs.umass.edu/msadmin>) allows content developers to define some parameters and automatically generate new instances of math problems using repeated code that is stored in a database.



The screenshot displays the 'MathSpring Admin Tool' interface. At the top, there is a navigation menu with links for 'Utilities', 'Tutoring Strategies', 'Math Problems', 'Math Topics', and 'Surveys'. The main heading is 'Create a new math problem'. Below this, there are several input fields: 'ID' (with a dropdown arrow), 'Name' (with a sub-label 'Problem Name'), and 'Language' (set to 'English'). A 'Nick Name' field is also present. A large text area is labeled 'Statement'. Below the statement area are two file upload sections: 'Audio File' and 'Image URL', each with a 'Choose File' button and the text 'No file chosen'. The 'Answer Type' is set to 'Short Answer'. There is a 'Delete' button and a 'Value' input field. Below that, there are 'Difficulty' and 'Status' (set to 'In Progress') dropdowns. A 'Layout' section shows a 'Default template' with a 'Layout Tool' button. At the bottom, there are several dropdown menus for 'Gr' (Grade), 'Dom' (Domain), 'Clust' (Cluster), 'Std' (Standard), and 'Part' (Part), followed by a 'Save' button.

Figure 3. The MathSpring Problem Authoring Tool

2.2 Programming Language

Because this project only includes the front-end development rather than fully integrating new problems into the website, I needed primarily Hypertext Markup Language (HTML) and Javascript. I also use ReactJS, a Javascript library, because it includes a virtual DOM which allows for quicker rendering of objects on a single webpage. ReactJS also permits a component-based structure which simplifies the design of each problem as they gain complexity. Because I am inexperienced with ReactJS, I use online tutorials to help understand the library. On websites like Codecademy.com and w3schools.com, I am able to take short courses and answer my questions pertaining to ReactJS. When I felt comfortable in my ability to create these new problems, I continued to the design aspect of this project. Because I aimed to make problems similar to those found on standardized tests, I visited the MCAS website to review past computer-based problems.

2.3 Method of Classification and Development

To construct a variety of new problem types for Mathspring.com, I first needed to gather the design of these problems from a preexisting high-stakes tests. By observing already present questions from the MCAS, made available by the Massachusetts Department of Education at <https://mcas.digitalitemlibrary.com/home>, I was able to analyze and classify the different kinds of questions that I could potentially create, which would resemble items that a student might see in a future exam.

After gathering approximately twenty examples with a variety of mathematical topics, I was able to classify them based on the topic and their theoretical code structure. After generalizing the basic idea of the question, I first specified which of the twenty questions

gathered from the standardized tests would be best to develop. Second I classified the problem types based on the complexity of the interface and the human-computer interaction. Because certain forms would be more lengthy and difficult to complete, I began with the simplest of the problem types and develop each problem. When a problem type had no more examples to replicate, I moved onto the following type as noted in a spreadsheet.

With a system for development in place and the problems classified and placed in the spreadsheet, I began to code problems using ReactJS within the following classifications: Drop Down Menu, Coordinate Plane, Check All that Apply, Make a Model, and Number Line. Each of these are described in the Results Section. For each problem, I iteratively produced the code. I began with the simplest design possibly and gradually increased the complexity of the problem until it fully represented the design. By generating problems one feature at a time, I simplified the development of each problem. Throughout this process, I recorded any issues or errors that could be resolved at a later date into a spreadsheet.

4. Results

Throughout this project I gathered examples of problems from past standardized tests to replicate for the use of MathSpring.org and stored them in a spreadsheet with information including their level of difficulty to create, their category, the link to the original problem, and any problems I experienced when writing the code. A simplified spreadsheet tracking the problems I created is included in Figure 4.

Difficulty	Category	Problem Name	Problem Description
1	CheckAll	CheckAll Problem Number One	Multiples of 4 and 7 Table
1	CheckAll	CheckAll Problem Number Two	Choose from A-F
1	DropDown	DropDown Problem Number One	Greater/Less/Equal
1	DropDown	DropDown Problem Number Two	Complete Correct Statements
2	MakeModel	MakeModel Problem Number One	Square Model - fractions
2	MakeModel	MakeModel Problem Number Two	Circle Model - fractions
2	NumberLine	NumberLine Problem Number One	Label a number on a numberline
3	CoordPlane	CoordPlane Problem Number One	Place Point (9,5)
3	CoordPlane	CoordPlane Problem Number Two	Make Line
3	CoordPlane	CoordPlane Problem Number Three	Make Triangle
3	CoordPlane	CoordPlane Problem Number Four	Make Shape (close-shape-button)
3	ConstructEquation	ConstructEquation Problem Number	Make Equation
4	DragDrop	DragDrop Problem Number One	System of Equations
4	DragDrop	DragDrop Problem Number Two	Complete the Expression
4	DragDrop	DragDrop Problem Number Three	Pumpkin Line Plot
4	DragDrop	DragDrop Problem Number Four	Equivalent Expressions
4	DragDrop	DragDrop Problem Number Five	Complete the Table
4	DragDrop	DragDrop Problem Number Six	Triangle Rotation
4	DragDrop	DragDrop Problem Number Seven	Evaluate Equation

Figure 4. A spreadsheet noting each of the problems in order of completion, their category, and their difficulty with 1 being the easiest and 4 being the most difficult.

Note: Problems listed below the thick line were not completed within project time frame.

3.1 Results of Classification

Figure 4 marks the classifications of each problem I acquired from the MCAS database and decided to recreate. First deciding the category, I divided each problem based on their structure and the common components they would require during development. Following this division, I based the difficulty ranking of the categories on the number of components and the level of interactivity compared to other categories. I decided the “Check All” category would be easiest to code because it only requires checkboxes next to a series of options. With few components and little interactivity, the “Check All” category was decisively easiest, while “Drop Down” followed. This category involved a drop-down menu being placed within a problem, prompting appropriate symbols to be chosen. Based on its increased amount of interactivity compared to the prior problem while still maintaining a fairly limited amount of variability in results, “Drop Down” was also ranked as a “1” in difficulty.

For each number in the table, select the box for any multiple that describes the number. You may select one or two boxes per row.

Number	Multiple of 4	Multiple of 7
16	<input type="checkbox"/>	<input type="checkbox"/>
28	<input type="checkbox"/>	<input type="checkbox"/>
42	<input type="checkbox"/>	<input type="checkbox"/>

Figure 5. An example of a CheckAll problem type.

Statistics

7.SPA.1.6

A large company has offices in cities across the country. The facilities director of the company was asked to survey employees about their office furniture. Rather than survey all employees in the company, the director decided to take a sample of employees. Which groups would be most representative of the opinions of all employees in the company?

Select **each** correct answer.

- A. employees with office phone numbers ending in 3 and 7
- B. randomly selected employees in the cafeteria of one of the offices
- C. employees who have worked for the company for more than 10 years
- D. 5% of randomly selected employees from each office location
- E. employees answering phone calls in the company's customer service department
- F. employees who are randomly selected by a computer from a list of all company employees

Figure 6. An example of a CheckAll problem type.

Figure 7. An example of a DropDown problem type including possible student response.

Figure 8. An example of a DropDown problem type including possible student response.

Because the “MakeModel” category includes a few more components, such as a shape, the division of that shape, and several buttons, and these components each have their own interactions available, this category is labelled a “2.” This category requires the division of a specific shape and shading of those divisions based on a given fraction. Comparing “MakeModel” to the next problem type, “NumberLine,” there are no discernable differences in

difficulty based on the number of components and level of interactivity; but because the latter has large similarities in code structure to “CoordLine,” I decided to build the one “NumberLine” problem after the “MakeModel” problems.

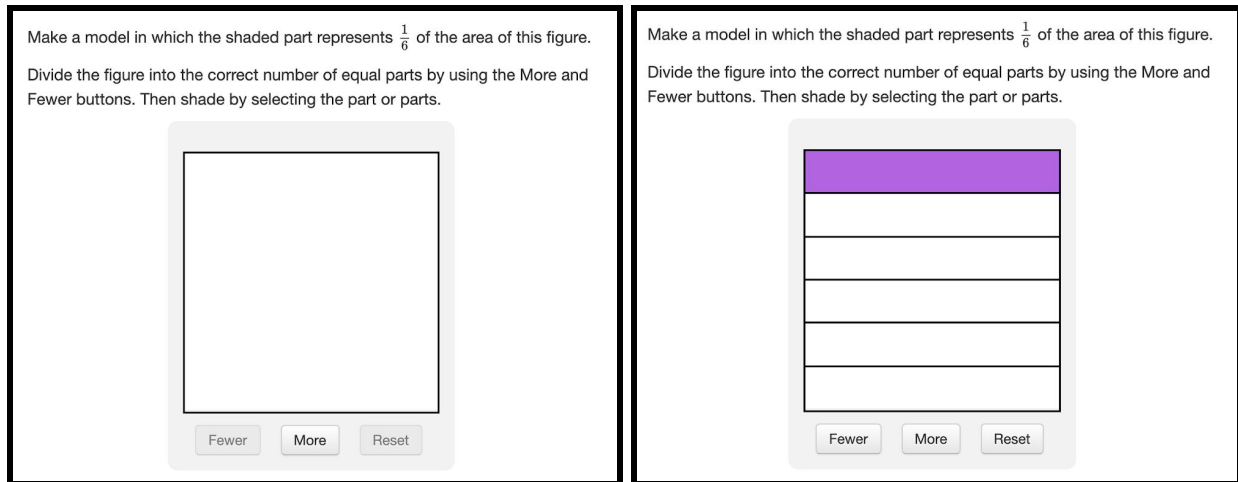


Figure 9. An example of a MakeModel problem type including possible student response.

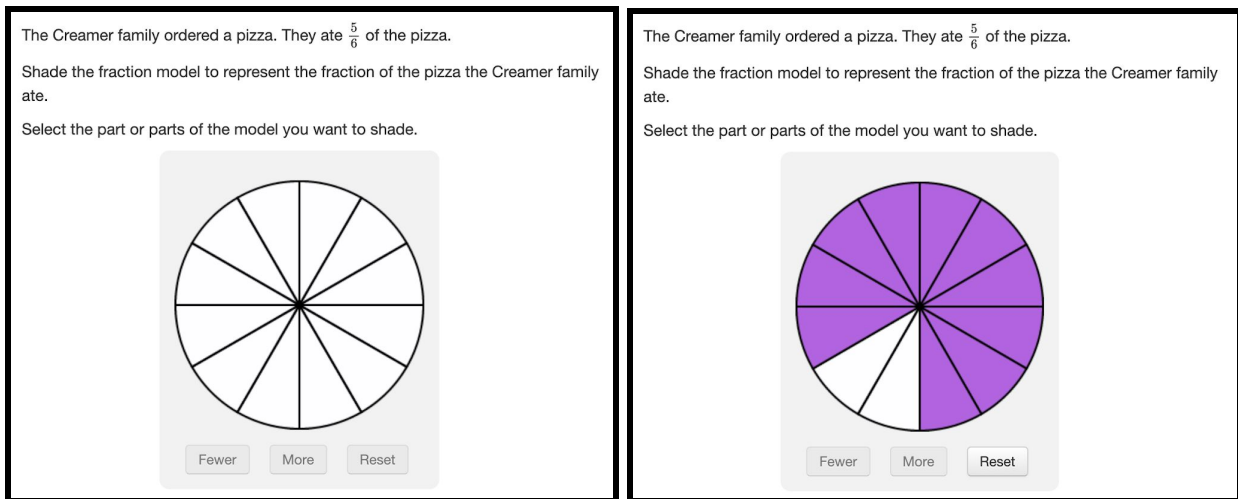


Figure 10. An example of a MakeModel problem type including possible student response.

The “NumberLine” category consists of a simple number line with the ability to mark a given number. Slightly more complex, the “CoordPlane” category introduces problems that

require the user to plot a point, line, or shape. With a complex design of interactivity and the most components necessary thus far, this category earns a “3” in difficulty.

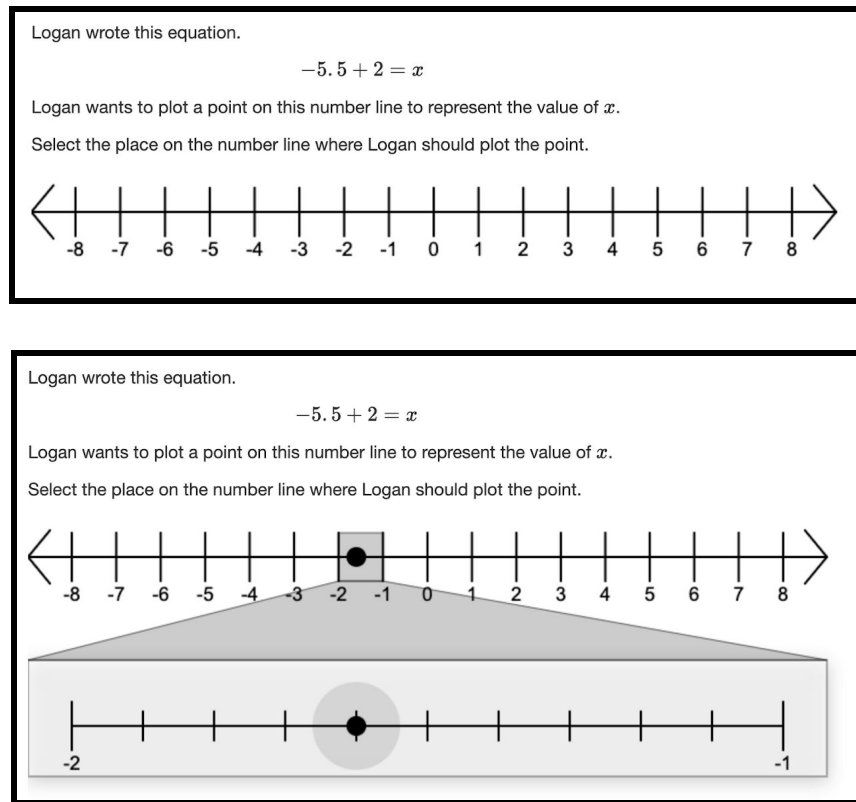


Figure 11. An example of a NumberLine problem type including possible student response.

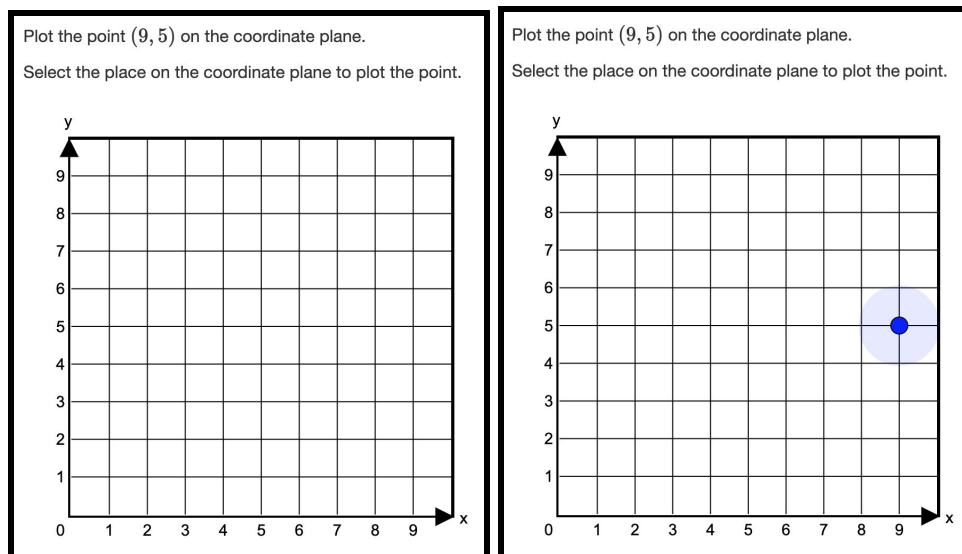


Figure 12. An example of a CoordPlane problem type including possible student response.

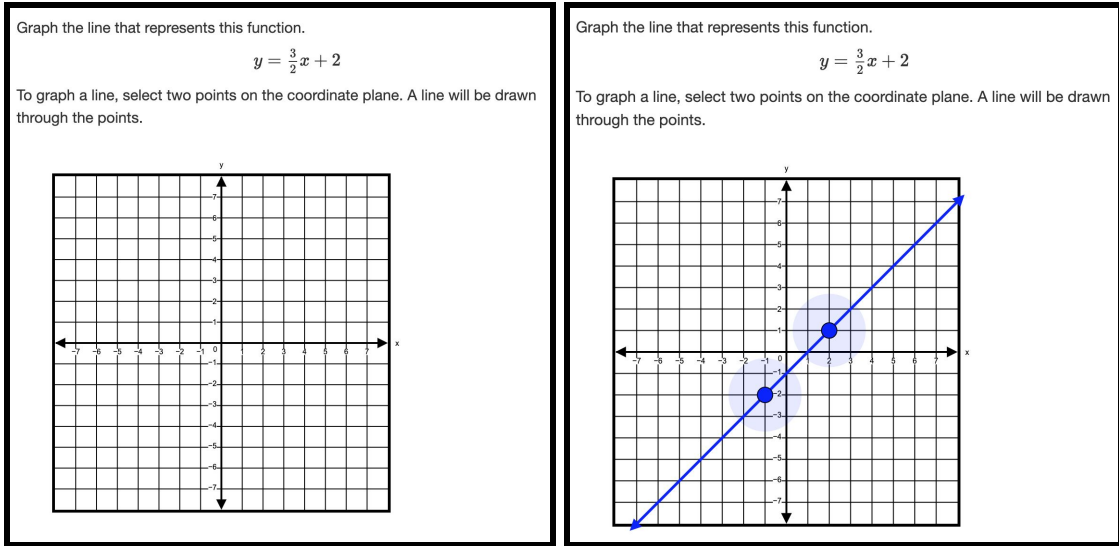


Figure 13. An example of a CoordPlane problem type including possible student response.

The “ConstructEquation” category includes “short-answer questions that allow users to include specific symbols from a table that would not otherwise be accessible. This inclusion of symbols in a textbox requires a large number of buttons, each with their own response, thus sustaining a larger number of components and level of interactivity consistent with a difficulty ranking of “3.”

Part A

Kevin cuts each orange into fourths. He has already cut 12 fourths.

How many oranges has Kevin cut so far? Show or explain how you got your answer.

Enter your answer and your work or explanation in the space provided.

▼ Math symbols

+	-	×	÷
$\frac{\square}{\square}$	\square^{\square}	(·)	[]
=	<	>	≠
\$	°	?	

Figure 14. An example of a ConstructEquation problem type.

The final category, “DragDrop,” asks that users drag an object with the mouse pointer to an appropriate location to answer a given question. This category allows for a great amount of variation between problems and their possible components, as well as a high level of interactivity. Each problem requires several mouse events dependant on the object with which the user is interacting. Though the resulting data itself would look similar to the responses to a multiple choice question, the visual effects would be the most difficult to create. Compared to the other categories, this one would require the most work which earns this problem type a “4.” After comparing and ranking each of these problems, I was able to conclude the classification and begin coding each problem.

Complete the expression to write this number in expanded form.

four hundred sixteen and eighty-two hundredths

Drag and drop a number into each box.

× 100 + × 10 + × 1 + × $\frac{1}{10}$ + × $\frac{1}{100}$

Complete the expression to write this number in expanded form.

four hundred sixteen and eighty-two hundredths

Drag and drop a number into each box.

× 100 + × 10 + × 1 + × $\frac{1}{10}$ + × $\frac{1}{100}$

Figure 15. An example of a DragDrop problem type including possible student response.

3.2 Process of Development

By classifying these examples, I was able to use ReactJS, HTML, and CSS to generate several problems that will provide other developers with an idea of how to incorporate abstracted versions of the problem types into MathSpring’s question designing tool. Each problem includes a main description as well a submit button outlined by a light gray border with a box shadow. With the necessary components included in the App.js file located in each appropriate project, each problem is rendered by the statement in Figure 16.

```
7 ReactDOM.render(<App />, document.getElementById( 'root' ));
```

Figure 16. A line of code that uses the ReactDOM to render the App object.

CheckAll Problem Number One

Because I was not familiar with ReactJS at the beginning of this project, I created the first problem with only HTML and CSS. This first problem, including a table of checkboxes, inquired whether three numbers were multiples of 4, multiples of 7, or both. Using HTML, I constructed a simple table with these options in the appropriate locations. Inside each cell, I included an input component with its type defined as a “checkbox,” as shown in Figure 17.

```
29 <form action="action_page.php" method="get">
30 <table>
31 <tr>
32 <th>Number</th>
33 <th>Multiple of 4</th>
34 <th>Multiple of 7</th>
35 </tr>
36 <tr>
37 <td>16</td>
38 <td><input type="checkbox" name="option1" value="16a"></td>
39 <td><input type="checkbox" name="option2" value="16b"></td>
40 </tr>
41 <tr>
42 <td>28</td>
43 <td><input type="checkbox" name="option1" value="28a"></td>
44 <td><input type="checkbox" name="option2" value="28b"></td>
45 </tr>
46 <tr>
47 <td>42</td>
48 <td><input type="checkbox" name="option1" value="42a"></td>
49 <td><input type="checkbox" name="option2" value="42b"></td>
50 </tr>
51 </table>
52 <br>
53 <input type="submit" value="Submit">
54 </form>
```

Figure 17. HTML representing a table of checkboxes regarding the multiples of 4 and 7

With a submit button included below, these components comprised a form that could produce an output when the submit button was clicked. Because the HTML in Figure 17 defines only the format and content of the table, I included style tags at the head of the page to complete the desired format as shown in Figure 18.

```

3 <head>
4 <meta charset="UTF-8">
5 <title>MathSpring Problem</title>
6 <style>
7   table, th, td {
8     text-align: center;
9     padding-left: 15px;
10    padding-right: 15px;
11    border-collapse: collapse;
12    vertical-align: center;
13  }
14  th, td {
15    border: 1px solid #DDD9E0;
16  }
17  div {
18    text-align: left;
19    height: 200px;
20    width: 600px;
21    font-size: 16px;
22    color: black;
23    border: 2px solid lightgray;
24    box-shadow: 5px 5px lightgray;
25    padding: 1%;
26  }
27 </style>
28 </head>

```

Figure 18. The head of the HTML page including the style tags.

With the style tags and their child components in place, the final result of the first problem is visible in Figure 19.

For each number in the table, select the box for any multiple that describes the number. You may select one or two boxes per row.

Number	Multiple of 4	Multiple of 7
16	<input type="checkbox"/>	<input type="checkbox"/>
28	<input type="checkbox"/>	<input type="checkbox"/>
42	<input type="checkbox"/>	<input type="checkbox"/>

Figure 19. The final rendition of CheckAll Problem Number Two made with HTML.

CheckAll Problem Number Two

Using ReactJS, I developed the next problem by creating a list of options and iterating through them while placing a checkbox next to each option. I began this process by creating a list of the possible options as shown in Figure 20. Each item on this list becomes the label for a designated checkbox.

```
5  const items = [  
6    'A. employees with office phone numbers ending in 3 and 7',  
7    'B. randomly selected employees in the cafeteria of one of the offices',  
8    'C. employees who have worked for the company for more than 10 years',  
9    'D. 5% of randomly selected employees from each office location',  
10   'E. employees answering phone calls in the company's customer service department',  
11   'F. employees who are randomly selected by a computer from a list of all company employees'  
12 ]  
13 ;
```

Figure 20. A list of strings that represent each checkbox option.

With the options stored in the list, I created a separate file to define a “Checkbox” component. This component class includes a label attribute, a key attribute, and a callback function attribute shown in Figure 21. The label is the text that appears next to the checkbox, and the key will be useful when the answer must be checked for correctness. The callback function allows the “Checkbox” component to communicate with its parent component the results of the clicked checkbox.

```
36  createCheckbox = label => (  
37    <Checkbox  
38      label={label}  
39      handleCheckboxChange={this.toggleCheckbox}  
40      key={label}  
41    />  
42  );
```

Figure 21. The creation of a checkbox component with the label, callback function, and key attributes.

If the checkbox is clicked, the function “toggleCheckboxChange” within the Checkbox component switches the isChecked value to true if it was unchecked and false if it was already checked. Additionally, the function calls the defined callback method, referenced in Figure 21. Shown in Figure 22, the attribute “handleCheckboxChange” calls “toggleCheckbox” from its parent component to store the label of the selected checkboxes in a list if it does not already exist and remove the label if it does. This list will be used later to confirm the correctness of the student user’s answers.

```
9 toggleCheckboxChange = () => {
10   const {handleCheckboxChange, label} = this.props;
11   this.setState( state: ({isChecked}) => (
12     {
13       isChecked: !isChecked,
14     }
15   ));
16   handleCheckboxChange(label);
17 }
```

Figure 22. The function “toggleCheckboxChange” that uses a callback function to share information with a parent function.

When combined, the App and Checkbox components render a problem with a question, choices accompanied by checkboxes, and a button for submission of the chosen answers.

A large company has offices in cities across the country. The facilities director of the company was asked to survey employees about their office furniture. Rather than survey all employees in the company, the director decided to take a sample of employees. Which groups would be most representative of the opinions of all employees in the company?

Select each correct answer.

- A. employees with office phone numbers ending in 3 and 7
- B. randomly selected employees in the cafeteria of one of the offices
- C. employees who have worked for the company for more than 10 years
- D. 5% of randomly selected employees from each office location
- E. employees answering phone calls in the company's customer service department
- F. employees who are randomly selected by a computer from a list of all company employees

Figure 23. The final rendition of CheckAll Problem Number Two built with ReactJS.

DropDown Problem Number One

After researching the components made available from the React library, I applied the react-dropdown module to my own code. With this module I added three dropdown menu objects each in between two numbers with options to define the equality of the values, as shown in Figure 24. Following a problem description, each dropdown menu and its accompanying numbers create an interactive problem.

```
44 <form className="Problems" onSubmit={this.handleSubmit}>
45   <div className="Problem">
46     <label> 2.09 </label>
47     <Dropdown className="One"
48               options={options}
49               onChange={this.handleChange}
50               placeholder="Select an option"
51               value={this.state.value}/>
52     <label> 2.12 </label>
53   </div>
54   <br />
55   <div className="Problem">
56     <label> 8.10 </label>
57     <Dropdown className="Two"
58               options={options}
59               onChange={this.handleChange}
60               placeholder="Select an option"
61               value={this.state.value}/>
62     <label> 8.1 </label>
63   </div>
64   <br />
65   <div className="Problem">
66     <label> 6.78 </label>
67     <Dropdown className="Three"
68               options={options}
69               onChange={this.handleChange}
70               placeholder="Select an option"
71               value={this.state.value}/>
72     <label> 6.7 </label>
73   </div>
74   <br />
75   <input type="submit" value="Submit"/>
76 </form>
```

Figure 24. The main code necessary to render three dropdown menus along with appropriate labels.

Each dropdown menu required options and a function that would handle the user's selected option. To create the dropdown menu, I created a list of options, as shown in Figure 25, each with a symbol as the label and a textual representation of the symbol as the value. With the value written as a word rather than a symbol, there will likely be less trouble when checking the response against the correct answer that will be stored in the database.


```
6  const options = [  
7    {value: "greater", label: ">"},  
8    {value: "less", label: "<"},  
9    {value: "equal", label: "="}  
10 ];
```

Figure 25. A list of objects each with a value and a label.

When a student clicks a specific choice, the dropdown menu calls the function “handleChange,” shown in Figure 26. This method takes the value chosen by the user and adds it to a list of selected responses. When the student finishes selecting options from the dropdown menus, they can submit their answers with the submit button.

```
31  handleChange = (event) => {  
32    this.responses.add(event.value);  
33  };
```

Figure 26. A function that adds chosen values to a list in the App.js file.

The use of the react module “dropdown,” labels to clarify the question for the student, and a submit button all exist in the App javascript file to render the problem shown in Figure 27.

Select from the drop-down menus to correctly complete each comparison.

2.09 2.12

8.10 8.1

6.78 6.7

Figure 27. The final rendition of DropDown Problem Number One.

DropDown Problem Number Two

Similar to DropDown Problem Number One, Problem Number Two utilizes the dropdown React module with a set of options defined in a list, as shown in Figure 28. The only difference between these two problems are the labels that define the question. The student would still choose a value measuring equality; but rather than only comparing two numbers in each of the three statements, the user will also use a second dropdown menu to explain this inequality.

```
6  const options = [  
7    {value: "less", label: "less than"},  
8    {value: "greater", label: "greater than"},  
9    {value: "equal", label: "equal to"}  
10 ];
```

Figure 28. A list of objects each with a value and a label.

The final product of DropDown Problem Number Two is shown in Figure 29.

Select from the drop-down menus to correctly complete the statements.

The product of $6 \times \frac{5}{3}$ will be 6 because the fraction $\frac{5}{3}$ is 1.

The product of $7 \times \frac{6}{6}$ will be 7 because the fraction $\frac{6}{6}$ is 1.

The product of $3 \times \frac{2}{3}$ will be 3 because the fraction $\frac{2}{3}$ is 1.

Figure 29. The final rendition of DropDown Problem Number Two.

MakeModel Problem Number One

To create this problem, I created a static square in a new component class called “SquareGroup” that is referenced in the App.js file. At the start this component comprised of a simple square to visualize the structure of the problem. From there I developed a divisible square by creating another module called Square shown in Figure 30.

```
3  class Square extends Component {
4    constructor(props) {
5      super(props);
6      this.state = {
7        background: 'white',
8        width: 300,
9      };
10
11      this.changed = 0;
12      this.fillSquare = this.fillSquare.bind(this);
13    }
14
15    componentDidUpdate() {...}
16
17
18    shareColor = () => {...};
19
20
21    fillSquare = () => {...};
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39    render() {
40      return (
41        <div className="square"
42          onClick={this.fillSquare}
43          style={{background: this.state.background,
44                width: this.props.width}}>
45        </div>
46      );
47    }
48  }
49
50  export default Square;
```

Figure 30. The Square component class from the Square.js file.

The Square component includes a fixed height and a mutable width; so when a Square is added to SquareGroup, the Square's width can change to maintain the correct size of the whole. Additionally, when the user clicks on a Square, it will change color. If the Square began unshaded, it will turn blue; but if it was blue before, it will become unshaded. This interactivity, shown by the code in Figure 31, allows users to create the numerator of a fraction. To create the denominator, the user must change the number of parts through the buttons included in the SquareGroup component.

```
30  fillSquare = () => {
31    if(this.state.background === 'white')
32      this.setState( state: {background: 'blue'});
33    else {
34      this.setState( state: {background: 'white'});
35      this.changed = 1;
36    }
37  };
```

Figure 31. A function that fills a square blue if it was originally white and white if it was blue.

Through the inclusion of three buttons in the SquareGroup class, the user controls the number of parts of the square model. There is a “More” button to increase the number of parts, as well as a “Less” button to reduce the number of parts. The final button is labelled “Reset” to restart the fraction modelling process. The creation of these buttons are shown in Figure 32.

```
58 render() {
59   return (
60     <div>
61       {this.makeSquares()}
62     <br/>
63     <button className="Reset" type="button" onClick={this.reset}>Reset</button>
64     &nbsp;
65     <button className="More" type="button" onClick={this.splitSquare}>More</button>
66     &nbsp;
67     <button className="Less" type="button" onClick={this.mergeSquare}>Less</button>
68   </div>
69 );
70 }
```

Figure 32. The code necessary to render the Squares and the buttons that control them in the SquareGroup.js file.

With the provided ability to create more or less shadeable components, the App file renders a final problem that allows users to effectively model a fraction as shown in Figure 33.

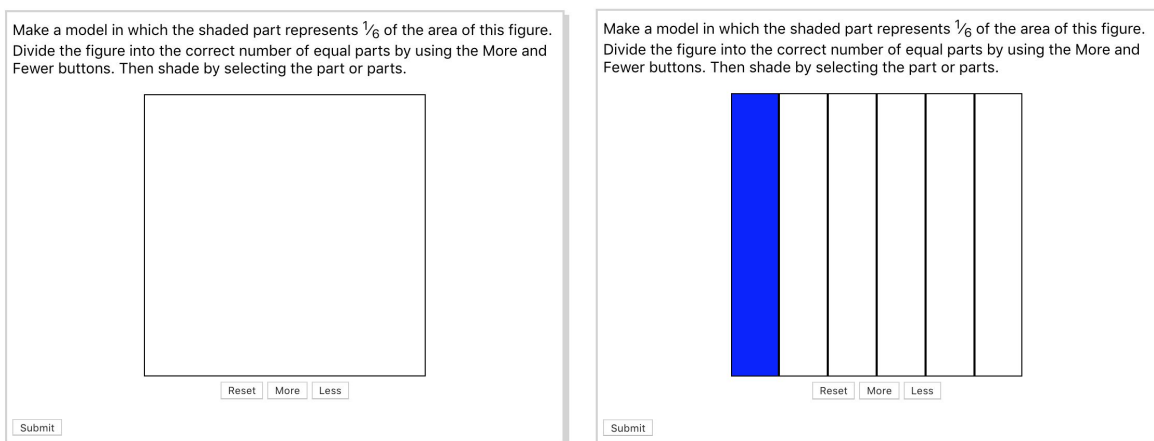


Figure 33. The final rendition of MakeModel Problem Number One with a possible student response.

MakeModel Problem Number Two

Similar to MakeModel Problem Number One, this problem began with a static circle rendered below the question. While I included the buttons here as well, they are disabled to reflect the MCAS problem on which I was working. Much like the previous problem, this one consists of a class called Circle with Arc components to represent the pieces of the whole. The code to render the Circle component class is shown in Figure 34.

```
53 render() {
54   return (
55     <div>
56       <div className="circle">
57         {this.makeArcs()}
58         <br/>
59       </div>
60     <div>
61       <button className="Reset" type="button" onClick={this.reset}>Reset</button>
62       &nbsp;
63       <button className="More" enabled="true" type="button" onClick={this.splitArc} >More</button>
64       &nbsp;
65       <button className="Less" enabled="true" type="button" onClick={this.mergeArc} disabled>
66         Less
67       </button>
68     </div>
69   </div>
70 );
71 }
72 }
```

Figure 34. An excerpt from the Circle.js file to render the Circle component.

Rather than a fixed height and variable width, the radius is fixed and the degree of the arc would change if the buttons were enabled. Similar to the Square of the former problem, the Arcs can be shaded individually to represent the numerator of the desired fraction. The code that renders the Arc is available in Figure 35.

```
24 render() {
25   let trans = "rotate("+this.props.rotate+"deg) skewY("+this.props.skew+"deg)";
26   return (
27     <div className="arc"
28       onClick={this.fillArc}
29       style={{background: this.state.background,
30         transform: trans}}>
31     </div>
32   );
33 }
```

Figure 35. The render function for the Arc component from the Arc.js file.

With these components working together, the user can model a desired fraction with a circle and its pieces as shown in Figure 36.

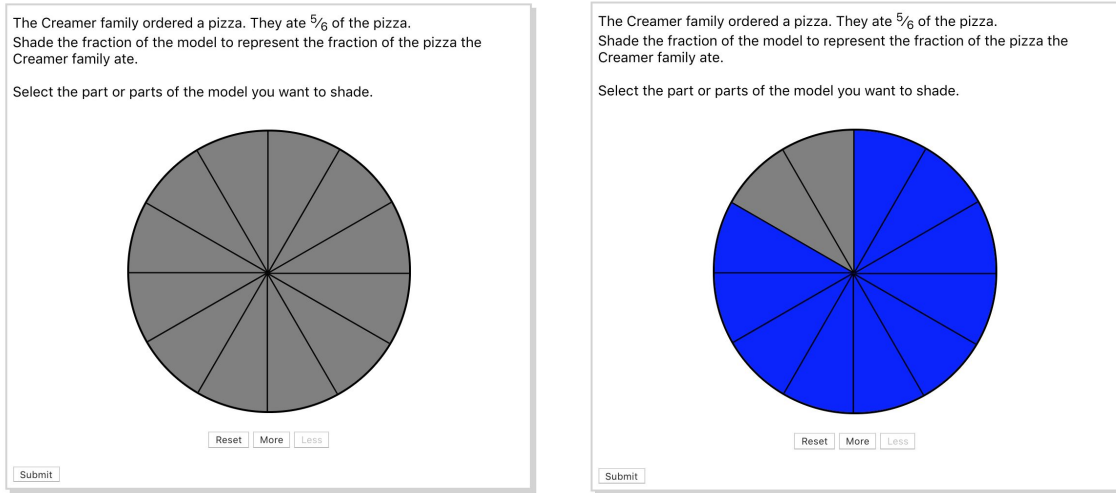


Figure 36. The final rendition of the MakeModel Problem Number Two with a possible student response.

NumberLine Problem Number One

To begin this problem, I created static elements such as the line to represent the number line and multiple tick marks to represent each number. When the static outline was complete, I began to abstract certain elements. I created a new class called TickMark to simplify the use of that component and its interactivity. The function that renders this component is shown in Figure 37.

```

47 render() {
48
49   // x-position of text changes based on number of digits and if negative
50   let textPosX = (this.props.number > 9 || this.props.number < -9) ? this.props.x - 15 : this.props.x - 10;
51   let textNegX = (this.props.number > 9 || this.props.number < -9) ? this.props.x - 10 : this.props.x - 5;
52   let textY = this.props.lineY - 15;
53
54   //tickMark information
55   let tickTop = this.props.lineY - 10;
56   let tickBottom = this.props.lineY + 10;
57
58   return (
59     <svg height={this.props.height} width={this.props.width}>
60       {(this.props.number < 0) ?
61         <text x={textPosX} y={textY}> {this.props.number} </text> :
62         <text x={textNegX} y={textY}> {this.props.number} </text>}
63       <line x1={this.props.x} y1={tickTop}
64            x2={this.props.x} y2={tickBottom}
65            style={{stroke: this.state.fill}}
66            onMouseEnter={this.hoverOver} onMouseLeave={this.hoverLeave} onClick={this.makeCircle}/>
67     </svg>
68   );
69 }

```

Figure 37. The render function for the TickMark component from the TickMark.js file.

With a TickMark component available, I created a loop to create multiple evenly-spaced TickMarks on the number line. Seeing that the number line required a few static elements to be complete, I also created a separate class for the NumberLine that would include the arrowheads, TickMarks, and the line itself. The code that creates the NumberLine is shown in Figure 38.

```

55 render() {
56   let arrowEnd = this.props.lineLength - 5;
57   let arrowStart = this.props.lineStart + 5;
58   let arrowEndPoint = arrowEnd + " " + (this.props.lineY - 5) + " "
59     + this.props.lineLength + " " + this.props.lineY + " "
60     + arrowEnd + " " + (this.props.lineY + 5);
61   let arrowStartPoint = arrowStart + " " + (this.props.lineY - 5) + " "
62     + this.props.lineStart + " " + this.props.lineY + " "
63     + arrowStart + " " + (this.props.lineY + 5);
64   this.makeTicks();
65
66   return(
67     <div className="numberline">
68       <svg height={this.props.windowHeight} width={this.props.windowWidth}>
69         <line x1="5" y1={this.props.lineY}
70            x2={this.props.lineLength} y2={this.props.lineY}
71            style={{stroke: this.props.color}}/>
72         <polyline points={arrowStartPoint} style={{stroke: this.props.color}}/>
73         <polyline points={arrowEndPoint} style={{stroke: this.props.color}}/>
74         {this.renderTickMarks()}
75         {this.renderDot()}
76       </svg>
77     </div>
78   );
79 }

```

Figure 38. The render function for the NumberLine component from the NumberLine.js file.

Once the structure of the number line was complete, I added the desired interactivity to the problem through the TickMarks. As shown in Figure 39, a TickMark includes three functions that allow the user to interact with the number line. When a student hovers over a TickMark, a dot appears and will disappear when the pointer leaves the proximity of the mark. When clicked, the dot will remain.

```
16   hoverOver = () => {
17     this.setState( state: {showCircle : true});
18     this.setState( state: {hover: true});
19     this.props.callbackHover(this.props.x);
20
21   };
22
23   hoverLeave = () => {
24     if(this.state.hover === true) {
25       this.setState( state: {showCircle: false});
26       this.props.callbackLeave();
27     }
28     this.setState( state: {hover: false});
29   };
30
31   makeCircle = () => {
32     this.props.callbackHover(this.props.x);
33     if(this.state.hover && this.state.showCircle){
34       this.setState( state: {showCircle : true});
35       this.setState( state: {hover: false});
36     }
37     else if(!this.state.hover && this.state.showCircle){
38       this.setState( state: {showCircle : false});
39       this.setState( state: {hover: false});
40     }
41     else {
42       this.setState( state: {showCircle : false});
43       this.setState( state: {hover: false});
44     }
45   };

```

Figure 39. The functions from TickMark.js file necessary to create a Dot on a number line.

There is a Dot class to assist in the functionality of the problem and avoid repetition of code when rendering a dot. This class also includes two functions that assist in the interactivity mentioned in the TickMark class called “hoverOver” and “hoverLeave.” The render function of the Dot class is included in Figure 40.


```

41  render() {
42      return (
43          (this.state.showCircle) ?
44              <circle cx={this.props.x} cy={this.props.y} r={this.props.r}
45                  style={{fill: this.state.circleFill}}
46                  onMouseEnter={this.hoverOver} onMouseLeave={this.hoverLeave}
47                  onClick={this.changeCircleVisibility} /> :
48              <circle cx={this.props.x} cy={this.props.y} r={this.props.r}
49                  style={{fill: 'none'}}
50                  onMouseEnter={this.hoverOver} onMouseLeave={this.hoverLeave}
51                  onClick={this.changeCircleVisibility} />
52          );
53      }

```

Figure 40. The function that renders a Dot from the Dot.js file.

As I drew to the conclusion of this problem and finalized the App file, I began to abstract qualities of each component and set them as global variables in App to show future developers where variables can be set in the problem authoring tool. These variables are shown in Figure 41.

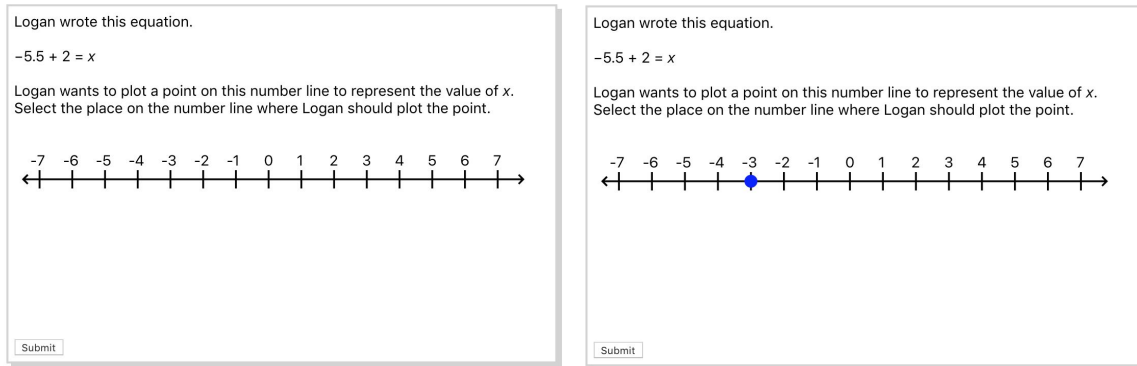


Figure 41. The final rendition of NumberLine Problem Number One with a possible student response.

CoordPlane Problem Number One

To create the CoordPlane Problem Number One, I began with a static outline similar to the number line. At the top of the designated workspace is the question, while the interactive

problem follows along with a submit button. To create the coordinate plane, I reused the loop from the NumberLine code. Starting from the center, each x and y line is generated moving outwards from the main axes. Designed as slightly thicker lines with arrowheads, the x-axis and the y-axis are rendered separately from the other lines. Located in the component class CoordPlane, the functions that create the lines that comprise the basic coordinate plane are shown in Figure 42.

```

139 makeArrowLines = () => {
140   let arrowEndX = this.props.xLength - 5;
141   let arrowStartX = this.props.xStart + 5;
142   let arrowEndPointsX = arrowEndX+" "+(this.props.yPlace - 5)+" "
143     +this.props.xLength+" "+this.props.yPlace+" "
144     +arrowEndX+" "+(this.props.yPlace + 5);
145   let arrowStartPointsX = arrowStartX+" "+(this.props.yPlace - 5)+" "
146     +this.props.xStart+" "+this.props.yPlace+" "
147     +arrowStartX+" "+(this.props.yPlace + 5);
148
149   let arrowEndY = this.props.yLength - 5;
150   let arrowStartY = this.props.yStart + 5;
151   let arrowEndPointsY = (this.props.xPlace - 5)+" "+arrowEndY+" "
152     +this.props.xPlace+" "+this.props.yLength+" "
153     +(this.props.xPlace + 5)+" "+arrowEndY;
154   let arrowStartPointsY = (this.props.xPlace - 5)+" "+arrowStartY+" "
155     +this.props.xPlace+" "+this.props.yStart+" "
156     +(this.props.xPlace + 5)+" "+arrowStartY;
157
158   return (
159     <svg height={this.props.windowHeight} width={this.props.windowWidth}>
160       <line x1={this.props.xStart} y1={this.props.yPlace} style={{stroke: this.props.color}}/>
161       <polyline points={arrowStartPointsX} style={{stroke: this.props.color}}/>
162       <polyline points={arrowEndPointsX} style={{stroke: this.props.color}}/>
163
164       <line x1={this.props.xPlace} y1={this.Y_START}
165         x2={this.props.xPlace} y2={this.props.yLength} style={{stroke: this.props.color}}/>
166       <polyline points={arrowStartPointsY} style={{stroke: this.props.color}}/>
167       <polyline points={arrowEndPointsY} style={{stroke: this.props.color}}/>
168     </svg>
169   );
170 };
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237

```

Figure 42. Three functions responsible for creating the x-axis, y-axis, and coordinated.

To simplify the creation of each line, I also created the XLine and YLine classes. By separating the code necessary to design each line from the loops in CoordPlane, I can simply create a new line with a given location without repeating the same information. The content of these classes are shown in Figure 43.

```
4 class XLine extends React.Component {
5
6   render() {
7     return(
8       <svg height={this.props.windowHeight} width={this.props.windowWidth}>
9         <line className="coordline"
10            x1={this.props.xStart} y1={this.props.y}
11            x2={this.props.xEnd} y2={this.props.y}
12            style={{stroke: this.props.color}}/>
13         <text x={this.props.xEnd/2 + 3} y={this.props.y - 3}>{this.props.text}</text>
14       </svg>
15     );
16   }
17 }
18
19 export default XLine;
```

```
4 class YLine extends React.Component {
5
6   render() {
7     return(
8       <svg height={this.props.windowHeight} width={this.props.windowWidth}>
9         <line className="coordline"
10            x1={this.props.x} y1={this.props.yStart}
11            x2={this.props.x} y2={this.props.yEnd}
12            style={{stroke: this.props.color}}/>
13         {(this.props.text !== 0) ?
14           <text x={this.props.x + 2} y={this.props.yEnd/2 - 3}>{this.props.text}</text> :
15           null
16         }
17       </svg>
18     );
19   }
20 }
21
22 export default YLine;
```

Figure 43. The classes used to create the x-lines and y-lines from the XLine.js file and YLine.js file, respectively.

Once the static setup was complete, I included the interactivity of the problem. Realizing I could not design the problem the same way as one dimensional NumberLine Problem Number One, I added small squares at each x-y coordinate. These square components, defined in the HoverSquare class, are rendered as visible entities in Figure 44. With HoverSquares placed behind the lines of the coordinate plane and colored white, I was able to include interactivity similar to the NumberLine Problem. When the mouse pointer hovers over a square, a Dot

component appears, defined by the Dot class replicated from NumberLine Problem Number One. If the user moves the pointer away, the Dot will disappear; but if the user clicks on the Dot, it will remain there.

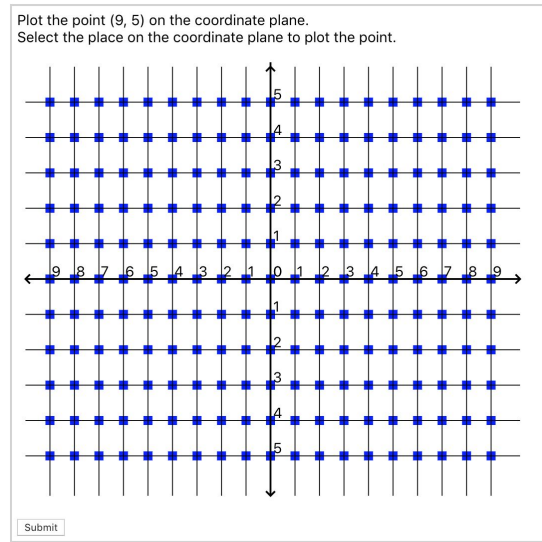


Figure 44. A rendition of the coordinate plane with the HoverSquares made visible.

With this new idea for interaction, I researched a way to prevent the lines of the coordinate plane from interrupting the mouse pointer. During this search, I discovered the CSS property “pointer-events.” By adding to the stylesheets “pointer-events: none” for each component I did not want to disrupt the mouse pointer, I could control how the user interacts with the problem. Figure 45 shows this property being utilized by various selectors which prevent the appropriate components from interrupting the mouse pointer.

```
24  .coordline {
25      stroke-width: 1;
26      pointer-events: none;
27  }
28
29  .dot {
30      pointer-events: none;
31  }
```

Figure 45. The application of the CSS Property “pointer-events.”

After discovering this CSS property, I realized I could apply it to previous problems rather than attempt to create a convoluted form of logic that would result in a similar outcome. By combining the static structure of the coordinate plane, the interactive HoverSquares, and the application of this CSS property, I was able to create the CoordPlane Problem Number One as shown in Figure 46.

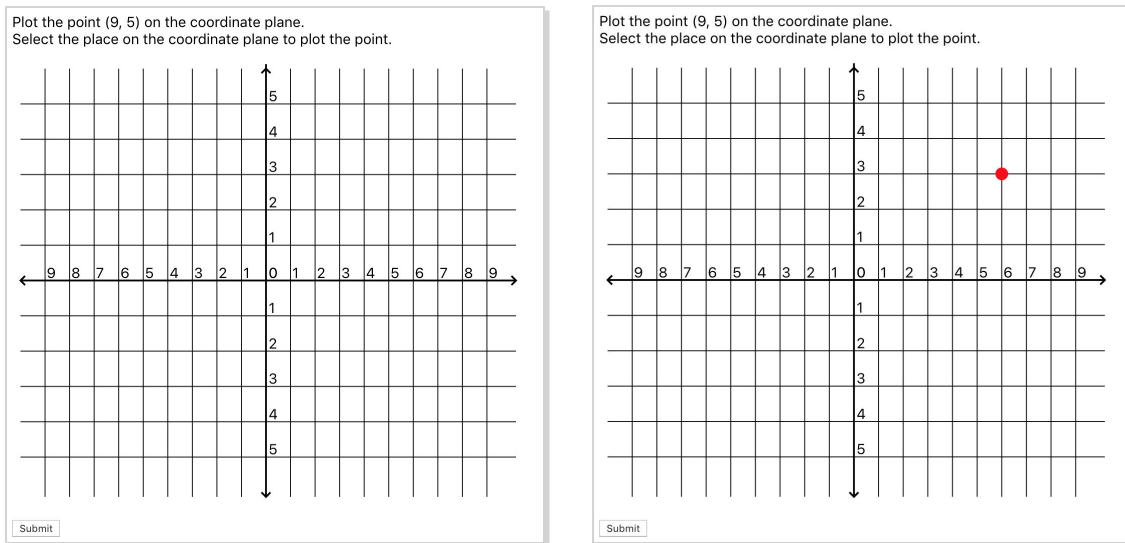


Figure 46. The final rendition of CoordPlane Number One with a possible student response.

CoordPlane Problem Number Two

I chose to do CoordPlane Problem Number One first because it presented itself as a first iteration of this problem. The coordinate plane and hidden squares remain the same, but the second CoordPlane Problem required the ability to add a second Dot and the generation of a line after the second Dot was created.

In CoordPlane Problem Number One, the number of Dots on the coordinate plane remains at one. By creating a list of Dots and limiting its number of items to one, I was able to maintain a specific number of Dot components. As I moved onto the next problem, I simply

increased the limit of this list to two. The code that reviews the size of this list to determine whether a line should be added is shown in Figure 47.

```
31 //If the dot has been clicked =>
32 clickDot = (x, y) => {
33   let g = [];
34   let index = -1;
35
36   for(let i = 0; i < this.state.dots.length-1; i++){...}
42   if(g[index] && g[index].visible){...} else {
49     g.push({x: x...});
56     if(g.length > 2){
57       g.splice(g.length-2, 1);
58       this.makeArrowLine(this.state.dots[0].x,
59         this.state.dots[0].y,
60         this.state.dots[this.state.dots.length-1].x,
61         this.state.dots[this.state.dots.length-1].y);
62       this.setState( state: {dots : g});
63     }
64     else if(g.length > 1) {
65       this.makeArrowLine(this.state.dots[0].x,
66         this.state.dots[0].y,
67         this.state.dots[this.state.dots.length-1].x,
68         this.state.dots[this.state.dots.length-1].y);
69     }
70     this.setState( state: {dots : g});
71   }
72 };
```

Figure 47. An excerpt from the clickDot function that reviews the size of the list of Dots.

By measuring the Dot list, I am able to gauge when the second Dot is added. Once the list exceeds one element, the makeArrowLine function is called and a dynamic line is generated that extends through the two Dots. This function uses basic algebra to find the slope of the line and the y-intercept between the two given points as shown in Figure 48. The arrowheads for the arrowed line are generated using trigonometry to maintain a consistent angle between the lines of the arrowhead and the line itself shown in Figure 49.

```

161 lineStartX = this.props.xStart;
162 lineStartY = m*lineStartX+b;
163 if(lineStartY >= this.props.yLength) {
164   lineStartY = this.props.yLength;
165   lineStartX= (lineStartY - b)/m;
166   multiplier = 1;
167 }
168 else if(lineStartY <= this.props.yStart) {
169   lineStartY = this.props.yStart;
170   lineStartX= (lineStartY - b)/m;
171   multiplier = -1;
172 }
173
174 lineEndX = this.props.xLength;
175 lineEndY = m*lineEndX+b;
176 if(lineEndY <= this.props.yStart) {
177   lineEndY = this.props.yStart;
178   lineEndX = (lineEndY - b)/m;
179   multiplier = 1;
180 }
181 else if(lineEndY >= this.props.yLength) {
182   lineEndY = this.props.yLength;
183   lineEndX = (lineEndY - b)/m;
184   multiplier = -1;
185 }
186

```

Figure 48. The algebra necessary to create the appropriate dynamic line on the coordinate plane.

```

187 let angle = Math.atan(-m);
188
189 let subAngle1 = (Math.PI/4) - angle;
190 let subAngle2 = (Math.PI/4) + angle;
191
192 let subLine = 10;
193 subLineXL = subLine*Math.sin(subAngle2);
194 subLineYL = subLine*Math.cos(subAngle2);
195 subLineXR = subLine*Math.sin(subAngle1);
196 subLineYR = subLine*Math.cos(subAngle1);

```

Figure 49. The trigonometry necessary to create the appropriate arrowheads on the dynamic line.

Following the addition of the arrowed line, the components come together in the App file to be rendered. The final result of this problem is shown in Figure 50.

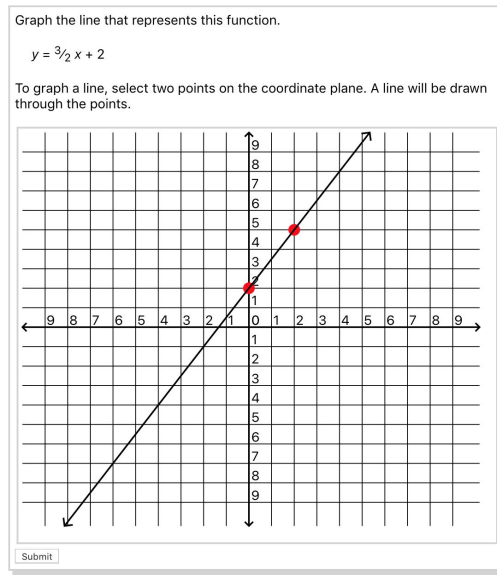
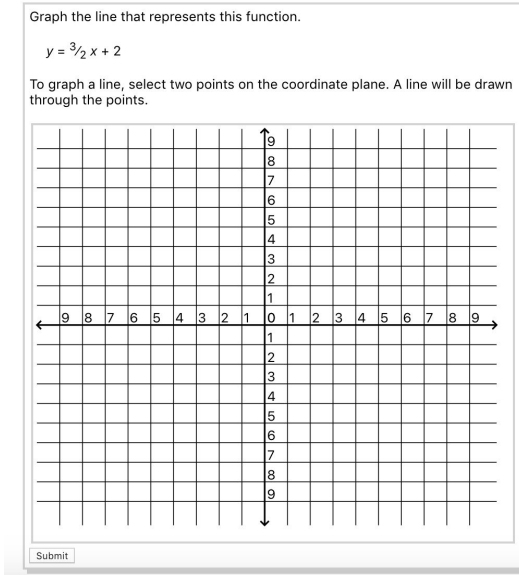


Figure 50. The final rendition of CoordPlane Problem Number Two with a possible student response.

5. Discussion

During this project, I aimed to develop new interactive problems for MathSpring, so students could become more familiar with the advancing technologies in education and high-stakes testing. While the classification of the problems was fairly simple, the development process provided an opportunity to learn new technical skills and methods of approaching a large project. Over these months of work, I evaluated languages new and old to me, maintained my code through Git, and organized my tasks with spreadsheets. The beginning of my work with designing new problems required a choice of programming language, so I researched the javascript libraries and frameworks that would best apply to my project. This research improved my knowledge as a student and a software developer.

As I built new problems, small issues would arise. I noticed during the development of the “CheckAll” problems that the order of the selections could prove a challenge when checking the work. After attempting to find a solution for a few hours, I realized it would not be worth my time to focus on the perfection of each detail. It was in this moment I began recording the small bugs I encountered so I could return to them later. Generally, I only chose to ignore an issue if it did not deeply affect the functionality or interpretation of the problem; otherwise I noted errors or ideas for each problem in my original spreadsheet. Due to my continuous learning of ReactJS, the code of each problem became more intuitive and better planned than the last. While I struggled to create the first problem programmed with React, the last few became more organized and included ideas I had acquired along the way. As I learned to use React, I became more eager to not only finish the problems I listed during the classification and planning phase, but to return to the past problems and clean up the code. Deciding to create these problems in

iterations rather than focusing on each one until perfection, I was able to complete a larger portion of the problems.

Throughout this project I applied my knowledge and research to create new problems for MathSpring.org. The use of spreadsheets allowed for better organization and time management, while my technical research founded new skills in software development. After learning a great deal throughout this project, I plan to continue working on the development of these problems for MathSpring.org. I look forward to completing new problems to aid in the future of MathSpring and its users as well as returning to past problems for revision.

References

- Durán, R. P. (2003). *NAEP validity studies: Implications of electronic technology for the NAEP assessment : Working paper series: NCES;2003 ASI 4826-23.87;no. 2003-16;NCES working paper no. 2003-16.* (). Retrieved from <https://statistical.proquest.com/statisticalinsight/result/pqpresultpage.previewtitle?docType=PQSI&titleUri=/content/2003/4826-23.87.xml>
- Jacobson, L. (2017). Nation's report card transitions to digital format. Retrieved from <https://www.educationdive.com/news/nations-report-card-transitions-to-digital-format/508593/>
- MCAS resource center. (2018). Retrieved from mcas.pearsonsupport.com