

Google Summer of Code 2018 Proposal for Oppia

New Interactions Project (Number with Units & Drag-and-Drop Sorting Interaction)

Personal Information

- **Name:** Vibhor Agarwal
- **University:** The LNM Institute of Information Technology, Jaipur, Rajasthan.
- **Field of Study:** B.Tech. (Computer Science and Engineering)
- **Email:** agarwalvibhor84@gmail.com
- **Github:** vibhor98 ([link](#))

Table of contents:

Why am I interested in working with Oppia?
What interests you about this project? Why is it worth doing?
Prior Experience
Project Plan and Implementation strategy
Drag and Drop Sorting Interaction
Randomization of the tiles (elements) in the interaction
Positioning of draggable tiles and markers
Technical Implementation
Backend Storage
Creating Interaction
Adding Tests
Adding Rules
Technical Implementation of the Rules
Explanations and Rationales
Number with Units Interaction
Technical Implementation
Backend Storage
Creating Interaction
Adding Tests
Adding Rules
Answer Verification
Technical implementation of the Rules
Currency Units
Workflow & Milestones
Summer Plans
Future Plans

Project Details

Project name: [New Interactions project](#) (Numbers with Unit & Drag-and-Drop Sorting Interactions).

Why am I interested in working with Oppia?

Oppia is a great learning platform which is unique in its story telling style of teaching the students no matter how difficult the concept is with no language constraints. Around seven months ago, I got in touch with the Oppia community and got to know about their marvelous work. As I love teaching, I found this platform as an apt to acquire good skills not only in backend and frontend development but also in implementing the lessons. Doing Hinglish audio translations for RCT in Delhi was a great fun as I had never done anything like this before. It gave me confidence and clarity to my speech. Since then I have been an active member of the community solving minor to major bugs, implementing new features like adding private information tooltip in the interactions, adding docstrings for the full documentation of the Oppia's backend and adding several docstrings linting checks. Slowly, I became very passionate about contributing to Oppia and therefore help the mankind in bringing this type of quality education on everyone's doorstep. Apart from this, this project involves both backend and frontend development. This is a good chance to acquire skills in both of these domains and manage such a large project with the skills for solving real world problems.

What interests you about this project? Why is it worth doing?

New Interactions is one of the significant learner facing projects for GSoC 2018. It aims at adding new drag and drop sorting interaction and can give enriching understanding of the units to the learners. It can surely bring a vast change in the teaching style of the creators and for the learners as they can significantly learn in the better way. Moreover, it requires both backend and frontend development. I've good chance to brush up my skills as I have good prior experience of working with Oppia in both backend and frontend development.

Prior Experience

I have been an active contributor to Oppia from the past 7 months and have made several contributions- both technical and non-technical.

Right from fixing minor to major bugs, adding docstrings to the Oppia's backend, adding private information tooltip to warn the learners about their answer being stored for analysis and doing Hinglish audio translations for the lessons.

[Here](#) is the complete list of all my technical contributions made to Oppia so far. Here, I'll describe some of my most significant contributions and projects that will substantially help me in implementing 'New Interactions' project.

- *Private Information Tooltip inside Interactions (#4650)*
This project adds a tooltip in almost every interactions to warn the learners to be careful while submitting their private information as their submitted answers are stored and can be seen by the creator of the exploration. It has greatly helped me in understanding how the interactions actually work in Oppia and gave me good familiarity with the codebase of the interactions. This will definitely give me benefit while doing this project. I will try to get it merged as soon as possible as most of the work has already been done.
- *Fully documenting the Oppia's backend*
This project intends to add docstrings to all the Oppia's backend files and thus help the contributors in the future. After completing this project, we can enable the missing docstrings pylint checks to ensure proper documentation in the future also. It gave me strong familiarity with the Oppia's backend.
- Some of my considerable PRs made so far:
<https://github.com/oppia/oppia/pull/4650>
<https://github.com/oppia/oppia/pull/4832>
<https://github.com/oppia/oppia/pull/4572>
<https://github.com/oppia/oppia/pull/4828>
<https://github.com/oppia/oppia/pull/4408>
<https://github.com/oppia/oppia/pull/4161>
<https://github.com/oppia/oppia/pull/4280>

Here are some of my self projects that resembles my strong familiarity with both backend and frontend development.

- *Slack bot- 'valet'* [[Github link](#)]
This bot empowers both the public chat rooms and private conversations. Using the Slack's rest Messaging API, this bot greets the users, gives them translation feature both text and speech using Google's gTTS Python library breaking the language constraints, gives Google search feature in the chat room and displays informative tweets related to the subject. This was the project I did under Computer Society of India. Even we can have this bot if Oppia plans to use Slack in the future.

Being a creator and a user myself, I have got good idea about what goes into making a cohesive and friendly user experience for both learners and creators.

Further, I'm proficient in Python and AngularJS. I have been working with Python in some way or the other from the past 3 years and with front end particularly AngularJS from the past 2 years. Here is the list of all my projects on Github.

Python - [Projects](#)

Web - [Projects](#)

Others - [Projects](#)

Project Plan and Implementation strategy

This project aims at implementing two new interactions- **Number With Units** and **Drag And Drop Sorting interaction** for improving the learner's experience.

The interactions play a very important role in Oppia. They help the learners in learning new concepts by solving certain questions asked within the exploration. These interactions provide many ways of input by the learner starting from simple Text Input, Numeric Input to more sophisticated Graph Input, Map Input, etc.

I have given due consideration on the functional and UI aspects of the proposed interactions in order to make the process as much intuitive and efficient as possible. The design and approach is based on the following -

- For the learner view of the interactions, I have taken 6 peer to peer reviews and feedbacks. Out of 6 reviewers, 2 were small kids while 4 were my colleagues. I have got interesting feedbacks that I have included at the relevant places.
- For the creator view of the interactions, I have got lots of feedbacks from the Oppia users that are very well familiar how the things work here. They were Oppia mentors and contributors that helped me in making the designs more concrete and intuitive.
- My own observation and experience with Oppia helped me to come up with these new interactions. They are consistent with the existing overall design and user experience of the interactions in Oppia.

Now let us throw light on the design implementation of each of the two interactions in the learner view as well as in the creator view.

Drag and Drop Sorting interaction

Current workaround:

Creators use number or fraction input field and ask students to write just the largest or the smallest value from the list of items. While sorting in the case of images is not possible yet. This is only partially serving our purpose as students are unable to sort the complete list of elements.

The **Drag and Drop Sorting interaction** will provide an interface for the learners to sort any given list of items by dragging and dropping the tiles easily. Our aim is to provide an easy to use and flexible drag and drop interaction for both the learners and the creators of the explorations. The detailed documentation can be found [here](#). However, this proposal covers all the major aspects of the technical implementation described in the link.

Solution:

To solve this problem we will create a new Drag and Drop sorting interaction which will provide the students with easy drag and drop feature to sort the tiles. The following are the details-

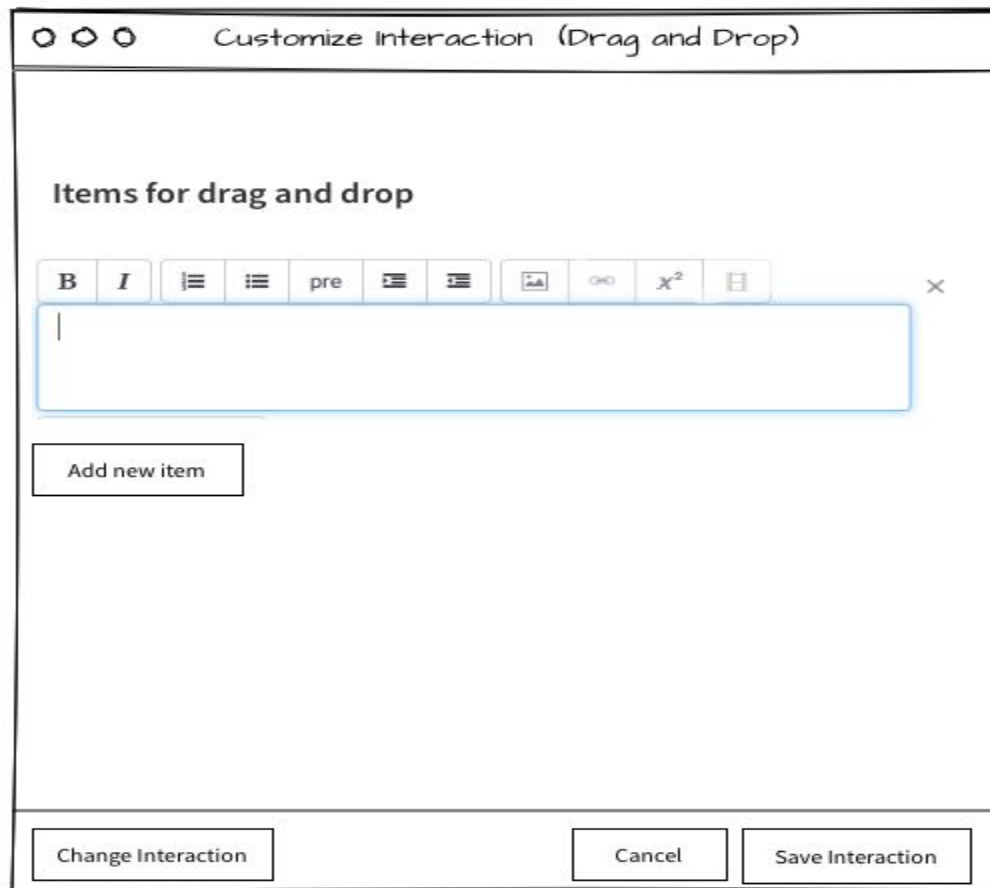
- The tiles (element card) can include objects like images, simple text, numeric input, fractional input or LaTeX input. Moreover, there is no restriction on the objects type and hence, they can be of any type including text, image, fraction, number etc. However, based on the feedbacks, we can add this restriction as the optional feature as described [here](#).
- Once the creator inputs any text or any other element, it will become a draggable tile that can be dropped at the desired location.
- At the time of adding response, the creator will assign the expected sequence number to each tile to set the correct answer sequence.
- For answer evaluation, 4 new rule specs "is equal to this ordering", "is equal to this ordering with at most one element in the wrong place", "has element X at position Y in the list" and "has element X coming before element Y" will be added.
- When the user plays an exploration that has drag and drop sorting interaction, the answer stored in backend will be transferred to frontend. This will contain a list of dicts of 'content', unique 'id' and 'position numbers' for each element (tile) in the right order.
- When the user submits an answer it will be also in the list format containing containing dicts of content, id and position numbers of the element from left to right in order.
- The answer can be verified by comparing the answer list submitted by the learner with the correct answer list. According to the chosen rule specs, the relevant feedback can be given after verifying the learner's answer.
- Moreover, if we further want that two or more tiles may take the same position in the sequence then the creator can simply assign them same position numbers in the sequence.

The student can place the tiles with the same position numbers one over the other but not completely hiding the previous tile as in the card game Solitaire. Ex- The fractions $\frac{4}{5}$ and $\frac{8}{10}$ will occupy the same position in the sequence and hence can be put one over the other. The mockups are discussed [here](#).



Let's start with the creator's journey first. Efforts have been taken to make the process as intuitive as possible by taking reviewers feedbacks and user studies. They are included at the relevant areas. The creator first starts by choosing **Drag and Drop interaction** from the 'Add

interaction' section while creating the exploration on the Creator's dashboard. He will then encounter the customize interaction page where he can set all the desired tiles for sorting. The page will look like-



Here the creator is free to add any number of items by clicking on the 'Add new item' button. Each item allows an image upload, simple text, numeric text, fraction input or a LaTeX input in the text area. So he is free to use any of the possible ways to set the tiles. However, the size of the tile will be fixed based on the screen size as the bigger or smaller images may vary its size making it look odd. Note that the options for video uploads and link will be customized in order to disable them. In the wireframe, they are made transparent so that the creator can not click on them.

The creator is also provided with the convenience to change the interaction in the middle as the case in our current interactions as well. Or he can cancel it if he thinks so. The third button 'Save Interaction' saves this interaction and redirects to 'Add Response' section. This section empowers the creators to set the correct answers of the tiles included.

Add Response

If the learner's answer...

is equal to this ordering...

Select 2/3

Select 7/2

Select 3/4

Select 4/2

Oppia tells the learner...
Nothing

And afterwards, directs the learner to...
(try again)

Cancel Save Response Save and Add Another

Add Response

If the learner's answer...

is equal to this ordering...

Select 2/3

Select

1

2

3

4

Oppia tells the learner...
Nothing

And afterwards, directs the learner to...
(try again)

Cancel Save Response Save and Add Another

The 'Add Response' tab supports multiple responses/ordering of the form- "is equal to this ordering", "is equal to this ordering with at most two elements in the wrong place", "has element X at position Y in the list" and "has element X coming before element Y".

Further for checking the rules- "is equal to this ordering" and "is equal to this ordering with at most two elements in the wrong place", the creator sets the sequence number for each tile by selecting the sequence number from the drop down list (from 1 to N) where N is the total number of tiles as seen in the mockup. Also, the creator sets the Oppia's response to the answer and finally sets the path where the card redirects to. At this point the creator successfully sets all the parameters for the interaction. The response is saved by clicking on 'Save response' and you can see its preview in the preview section. Note that the preview will look like the [interaction](#) as in the learner view described in the learner's journey.

For checking the rule- "has element X at position Y in the list" as in the wireframe,

Add Response

If the learner's answer...

has element X at position Y in the list...

Select element

2/3

7/2

Set position

Select

Oppia tells the learner...

Nothing

And afterwards, directs the learner to...

(try again)

Cancel Save Response Save and Add Another

the creator selects the desired element (say 2/3) from the 'Select Element' section. Then it's correct position is set from the drop down in the correct answer sequence. This rule only checks the selected element (say Image 2/3) whether it is at the correct position or not. However, the creator can add multiple responses to add multiple constraints on the accepted answer. He can select multiple elements on which this rule applies by clicking 'Save and Add Another' button. However, the answer is verified and marked as correct if anyone of the rules passes. They are in the OR condition.

For checking the rule- "has element X coming before element Y" as in the wireframe

Add Response

If the learner's answer...

has element X coming before element Y...

Select element

2/3

7/2

coming before element

2/3

7/2

Oppia tells the learner...

Nothing

And afterwards, directs the learner to...

(try again)

Cancel Save Response Save and Add Another

the creator first selects the desired element (say 2/3) from the 'Select element' section. Then, he selects the other element (say 7/2) that has to come after '2/3' in the accepted answer. This rule only checks whether the given element lies before the other in the submitted answer list. However, we can add multiple responses to add multiple constraints on the accepted answer here also. He can select multiple elements on which this rule applies by clicking 'Save and Add Another' button. However, the answer is verified and marked as correct if anyone of the rules passes.

Randomization of the tiles (elements) in Drag and Drop Interaction

After the implementation the tiles will be shown to the learners in the incorrect order everytime as set by the exploration creator. But we can extend it to randomize the order of the tiles every time so that the learners do not get the same order every time. However, this random sequence will be compared with the correct answer sequence as set by the creator. There are $1/n$ chances that the random sequence will be same as the correct answer where n is the number of

tiles. Definitely we would avoid to show the correct sequence to the learner at the beginning only.

So through this intuitive process, the creator can set this interaction along with all the rule specs for the answer responses successfully.

Now let's come to the learner's journey. The learner finds the drag and drop interaction like:

Look at the three fractions here. Can you arrange them in the ascending order?

Drag and drop tiles here →

$\frac{2}{3}$	$\frac{7}{2}$	$\frac{3}{4}$
$\frac{4}{5}$	$\frac{5}{4}$	$\frac{6}{3}$
$\frac{4}{6}$	$\frac{3}{2}$	$\frac{4}{3}$

Submit

Remember: Order the tiles altogether from left to right.

The drag and drop interaction is displayed in the supplemental mode here. This example requires the learner to arrange the fractions in the increasing order. Every blue tile containing fractions is draggable and can be dropped at the suitable location.

For this drag and drop interaction, both absolute and relative positioning of the tiles can take place for sorting. But based on the feedbacks from the reviewers and user studies, relative positioning emerged victorious over the other. [Here](#) [2] are the reasons.

So the current implementation allows the **relative positioning** of the tiles for sorting. It means every tile is draggable and when it is dragged onto another all the preceding or succeeding tiles shift either left or right accordingly. Here in the mockups at most 9 tiles can be displayed at once on the screen. However, if we have tiles less than 9 then only that number of tiles will be displayed.

Look at the three fractions here. Can you arrange them in the ascending order?

Drag and drop tiles here →

$2/3$	$7/2$	$3/4$	↑
$4/5$	$5/4$	$6/3$	
$4/6$	$3/2$	$4/3$	

Submit

Remember: Order the tiles altogether from left to right.

The blue dotted rectangle denotes the current position of the tile that is held before dropping it to the appropriate location. Additionally, it has solved the problem when the tiles are really large in number.

$2/3$	$4/3$	$3/4$	↑
$4/5$	$5/4$	$6/3$	
$4/6$	$3/2$		

Submit

Remember: Order the tiles altogether from left to right.

$2/3$	$7/2$	$3/4$	↑
$4/5$	$5/4$	$6/3$	
$4/6$	$4/3$		

Submit

Remember: Order the tiles altogether from left to right.

If the number of tiles exceeds nine, the learner can use these two options to navigate through the tiles smoothly.

- He can use up and down navigation arrow buttons for scrolling through the tiles manually as seen in the extreme right in the mockups.
- Whenever a tile is held, it is not possible to hold the tile and click on the arrows simultaneously. As a remedy, whenever the dragged tile comes to the extreme upward or downward side of the interaction, there is automatic scrolling up or down based on the position of the tile as shown in the mockups. The up and down red arrow buttons act as the markers in this case.

One of the reviewer reported in his study that the learner can get confused with the ordering-whether to start ordering from left or from right.

- The first solution I could dig out is that we can add informative tooltip. On hovering, it should notify the user that the ordering starts from left to right actually by sorting tiles altogether and not row-wise.
- The second approach is to directly show the text message that was previously shown after hovering over the help icon.

But the second solution emerged victorious over the first. Here are the two reasons why.

- First, the learner may not hover over the help icon making him completely unaware of the useful note.
- Secondly, hovering does not work on the mobile devices. It is important to show this message on the very first go without requiring the learner to do anything.

Positioning of the draggable tiles and markers:

Look at the three fractions here. Can you arrange them in the ascending order?

Drag and drop tiles here →

$\frac{4}{3}$	$\frac{7}{2}$	$\frac{3}{4}$
$\frac{4}{5}$	$\frac{5}{4}$	$\frac{6}{3}$
$\frac{4}{6}$	$\frac{3}{2}$	

Submit

Remember: Order the tiles altogether from left to right.

The tiles indicated with the blue color are draggable. Here, $4/3$ is dragged and held over the first tile. The red circular marker indicates that this tile will be dropped over the first tile when released. This red marker will appear when the tile, that is held currently, is put at the center of the intended position. This situation will occur when two or more tiles can take up the same position and the learner can put them over each other. Finally after dropping the tile, it will look like-

The image shows a 3x3 grid of fraction tiles. The tiles are arranged as follows:

$4/3$	$7/2$	$3/4$
$4/5$	$5/4$	$6/3$
$4/6$	$3/2$	

Below the grid is a 'Submit' button and a reminder box that says: "Remember: Order the tiles altogether from left to right." On the right side of the grid, there are two arrow buttons: an upward-pointing arrow above the downward-pointing arrow.

The dashed blue vacant area for the dragged interaction is removed as no one acquires it and thus, it is vacant now. Here $4/3$ is put over $2/3$ but $2/3$ is completely hidden. The best solution I came up with after many user reviews is- After clicking on the hidden tile it will reappear over $4/3$ with the highlighted boundaries. See the wireframes below. While clicking elsewhere the tile will get back to its original position i.e. somewhat behind the $4/3$ tile.

The image shows a 3x3 grid of fraction tiles. The tiles contain the following fractions:

$\frac{2}{3}$	$\frac{7}{2}$	$\frac{3}{4}$
$\frac{4}{5}$	$\frac{5}{4}$	$\frac{6}{3}$
$\frac{4}{6}$	$\frac{3}{2}$	

Below the grid is a 'Submit' button and a reminder box that says: "Remember: Order the tiles altogether from left to right." There are also up and down arrow buttons on the right side of the grid.

The second situation occurs when the learner wants to pick one tile and drop at the specific position. The existing tile has to be shifted right by one position and consequently, all the tiles following it. This will be shown by the red right arrow marker that shows that all the tiles after it will be shifted by one place to vacant the position for the tile. This marker will appear when the center of the tile is positioned to the left of the center of intended position. The wireframe is shown below.

2/3 4/3 3/4

4/5 5/4 6/3

4/6 3/2

Submit

Remember: Order the tiles altogether from left to right.

Similarly for the situations when the learner wants to put the tile over the intended position and left shift all the tiles preceding it, the red left arrow marker will be displayed. It will appear when the held tile is positioned to the right of the center of the intended position. All the preceding tiles will shift left by one position when the held tile is dropped. Note that the first tile will go to the last in this case to occupy the vacant position.

2/3 4/3 3/4

4/5 5/4 6/3

4/6 3/2

Submit

Remember: Order the tiles altogether from left to right.

Ultimately after dropping all the tiles one by one at the intended positions, the learner can submit his/her answer now.

For the mobile screen widths, the interaction in the learner's view will look like:

Look at the three fractions here. Can you arrange them in the ascending order?
Drag and drop tiles below

$2/3$	$7/2$	$3/4$	▲
$4/5$	$4/6$	$6/3$	▼

Submit Order the tiles altogether from left to right.

Look at the three fractions here. Can you arrange them in the ascending order?
Drag and drop tiles below

$6/3$	$7/2$	$3/4$	▲
$4/5$	$4/6$		▼

Submit Order the tiles altogether from left to right.

This is the case when one tile is put over the other.

Look at the three fractions here. Can you arrange them in the ascending order?
Drag and drop tiles below

$6/3$	$7/2$	$3/4$	▲
$4/5$	$4/6$		▼

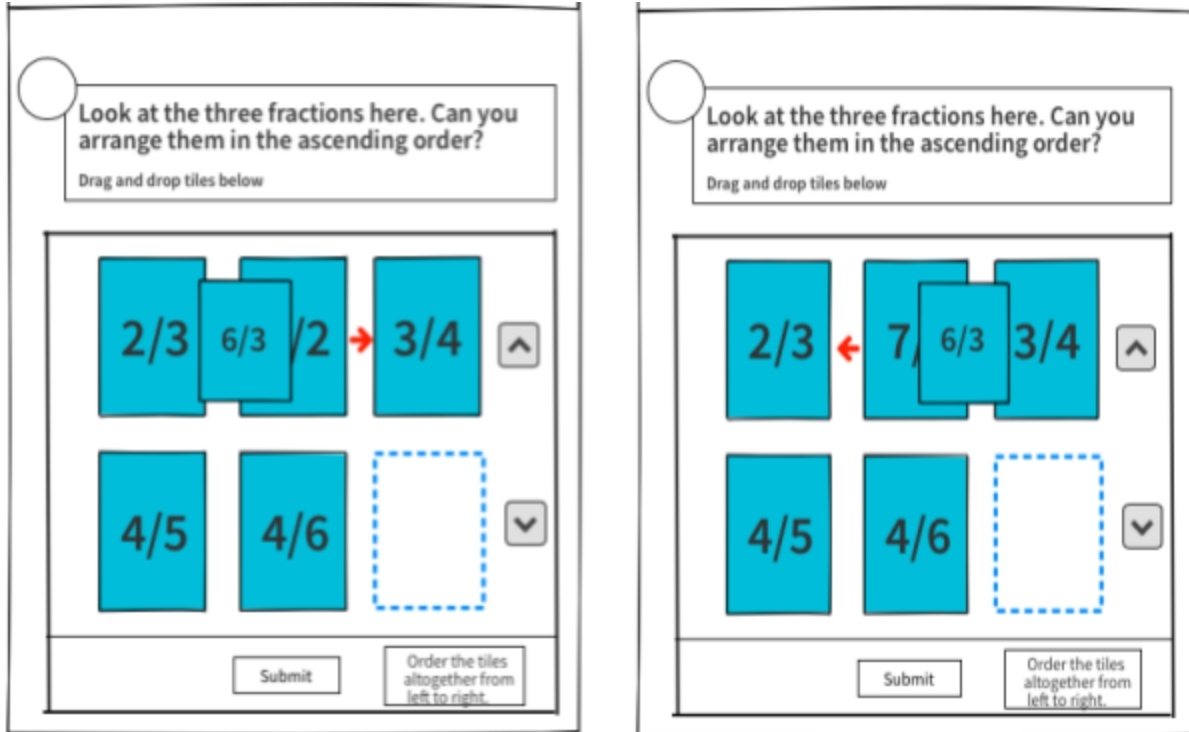
Submit Order the tiles altogether from left to right.

Look at the three fractions here. Can you arrange them in the ascending order?
Drag and drop tiles below

$2/3$	$7/2$	$3/4$	▲
$4/5$	$4/6$		▼

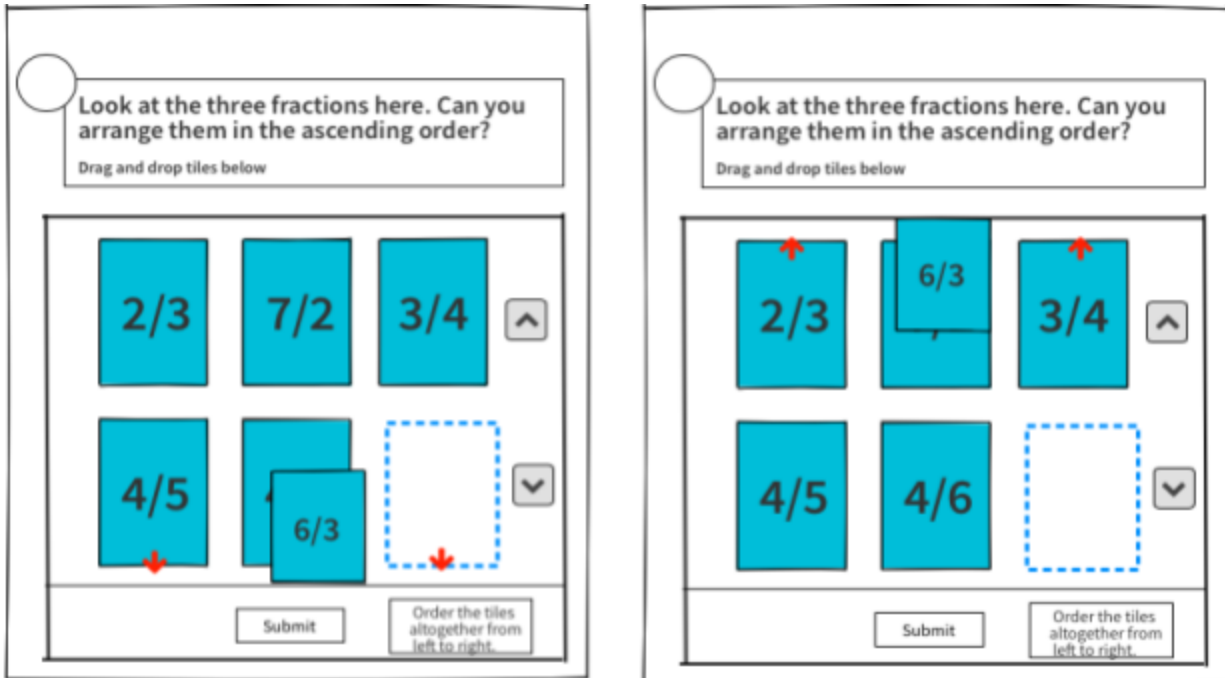
Submit Order the tiles altogether from left to right.

Note that the interaction uses similar drag and drop feature here also as described for the larger screens except that the currently dragged tile will shrink in its size. It is done so that the left, center and right positioning of the tile can be done easily without intersecting with the neighboring tiles. Here are the wireframes for different positions of the tile currently being dragged.



While developing the user facing projects, its equally important to test the product and take the feedback from the learners for whom we are devoting our time and efforts. As our target learners will be using mobile devices mostly that too android so, I will be testing the interaction on several android mobile devices of the varying screen size.

The mockups below show the markers for automatic scrolling up or down when the dragged tile is either at the extreme top or at the extreme bottom respectively.



As evident from the user reviews and feedbacks, the learners will definitely find this interaction intuitive to use and will give them better approach of learning by sorting things with this drag and drop sorting interaction. We have come to the end of the presentation of the design layout for Drag and Drop interaction. Let's discuss its technical implementation in detail.

Technical Implementation

Backend storage:

- The schema for this information can be implemented using lists. This list will store the content, unique element 'id' and a non-unique 'position number' for each and every tile (containing image, text input, etc.) in the **dict** format.
 Ex- `[{ 'content': html.SCHEMA, 'id': 1, 'position_no': 2 }, {...}, ...]`.
 Note that here we have used list of dicts instead of list of lists. The reason why list of dicts is preferred is discussed [here](#) [1].
- At the time when the creator sets the correct response, its corresponding list containing dictionaries for every tile will be created in the frontend and transferred to the backend.
- In case when more than 1 tile has to acquire the same position, they will be assigned a unique id but their position number in the correct sequence must be equal (non-unique). This will keep track of the ties (two or more tiles occupying the same position) and distinguishes them with the unique id as well.

To add new backend data storage, we will add **SetOfDragAndDropElements** and **DragAndDropElement** class to `extensions/objects/models/objects.py` -

```

class DragAndDropElement(BaseObject):
    default_value = []

    SCHEMA = {
        'type': 'dict',

        'properties': [{
            'name': 'content',
            'schema': Html.SCHEMA,
        }, {
            'name': 'id',
            'schema': PositiveInt.SCHEMA
        }, {
            'name': 'position_no',
            'schema': PositiveInt.SCHEMA
        }],

        'validators': [{
            'id': 'is_uniquified'
        }]
    }

```

Why we have separated DragAndDropElement class from the set of elements schema?

- The rule specs 3rd and 4th as described [here](#) need single element X and Y and not the set of elements. So, this schema satisfies the requirements of all the rule specs.

```

class SetOfDragAndDropElements(BaseObject):
    default_value = []

    SCHEMA = {
        'type': 'list',
        'items' = DragAndDropElement.SCHEMA;
    }

```

Thus, the answer in the JSON format will contain each item with its 'content', unique 'id' and non-unique 'position number'.

We need to create supplemental Drag and Drop editor for this interaction. The following files will be added to **extensions/objects/templates**:

- **drag_and_drop_editor.html**: It contains template for Drag and Drop interaction. It determines how the interaction actually looks.

- **DragAndDropEditor.js:** It contains directives for controlling the overall functionality of this interaction and complements drag_and_drop_editor.html. It controls the behaviour for handling id and position_no of each tile and validates the input within the editor before the submission.

New files to be added:

The following new backend and frontend files will be added under **oppia/extensions/interactions/DragAndDropSort**:

Creating Interaction

- **DragAndDropSort.py** : This is a Python configuration file that provides various attributes for the interaction. These attributes are documented in *extensions/interactions/base.py*. Some of the attributes are:

```
display_mode = base.DISPLAY_MODE_SUPPLEMENTAL
answer_type = 'SetOfDragAndDropElements'
can_have_solution = True
show_generic_submit_button = True

_customization_arg_specs = [{
    'name': 'choices',
    'description': 'Items for drag and drop',
    'schema': {
        'type': 'list',
        'validators': [{
            'id': 'has_length_at_least',
            'min_value': 2
        }],
        'items': {
            'type': 'html',
            'ui_config': {
                'hide_complex_extensions': True,
                'placeholder': 'Enter at least two options for the learner' +
                    'to drag and drop',
            },
        },
        'ui_config': {
            'add_element_text': 'Add new item',
        }
    },
    'default_value': [],
}]
```

- **./directives/DragAndDropSort.js** : It contains two directives that define the behaviour of the interaction- **oppiaInteractiveDragAndDropSort**, **oppiaResponseDragAndDropSort** and a factory service called **dragAndDropSortRulesService**.

- The **oppiaInteractiveDragAndDropSort** directive defines the view that is presented to the learner to interact with.
 - The dragging and dropping of the tiles can be handled smoothly by using Angular [ui-tree](#) API. It fulfills all the requirements required for the drag and drop functionality. The documentation for external reference can be found [here](#).

We can treat every tile (element) as a separate node of the tree here and add *ui-tree-node* to each element. In our implementation each node (tile) does not contain sub-nodes so we refrain from adding *ui-tree-handle* so that the entire node can be dragged and dropped.

But for the center position of the tile when one tile has to be put over the other as discussed in the [mockup](#), the dragged tile will become the sub-node of the existing tile.

To highlight the dragged tile, *angular-ui-tree-drag* is added to the required element and a new empty node is added into the tree to act as the placeholder by adding *angular-ui-tree-placeholder* class to the new node. Structure of the *angular-ui-tree* will be-

```

ui-tree           → Root scope of the tree
  ui-tree-nodes   → Container of nodes
    ui-tree-node  → One of the node(tile) of a tree
      ui-tree-handle → Handle
    ui-tree-node  → Another node(tile)
      ui-tree-handle → Handle

```

- It receives the user's answer sequence in an associative array *\$scope.userSequence* containing 'id' of each tile, their 'content' and 'position_no'.
 - It checks for the validity of an answer using *\$scope.isAnswerValid*. It prevents submission of empty or undefined array (happens when none of the tile is dragged and dropped).
 - Finally, it submits an answer to the server if it passes the validation using *\$scope.submitAnswer*.
- The **oppiaResponseDragAndDropSort** directive defines the view that is shown to the learner as a response after the interaction has taken place. It basically receives the answer in the JSON format from *\$attrs.answer*, converts it into object using *HtmlEscaperService.escapedJsonToObj()* and stores it in the *\$scope.answer* variable.

- The **dragAndDropSortRulesService** factory service is described in the [Adding Rules](#) section. It contains the functions that evaluate rules described in the **rule_templates.json** (ex- IsEqualToOrdering, HasElementXAtPositionY etc.) and returns a Boolean value (True if the rule is satisfied, False otherwise).
- **DragAndDropSort.html** : It contains two templates used to display the interaction's HTML, each of which is bounded to the directives in DragAndDropSort.js and DragAndDropSortValidationService.js .
- **./directives/drag_and_drop_sort_interaction_directive.html** : It contains the template corresponding to the directive **oppiaInteractiveDragAndDropSort** defined in 'DragAndDropSort.js' for the interaction before the answer submission.
- **./directives/drag_and_drop_sort_response_directive.html** : It contains the template corresponding to the directive **oppiaResponseDragAndDropSort** defined in 'DragAndDropSort.js' for the response to the answer submitted by the learner.

Once these files are created, the interaction can be activated by adding interaction's name- **DragAndDropSort** to the ALLOWED_INTERACTION_CATEGORIES variable in '**feconf.py**' under the 'General' category. Then, the interaction should be available in the interaction repository, and can be used in the explorations.

Adding Tests

Oppia uses end-to-end testing framework for testing. These tests mimic the actual users by interacting with the web page normally by clicking and typing. Then the expected behaviour of the interaction is checked. The interaction specific tests can be enabled by adding the interaction name to **extensions/interactions/protractor.js** .

The drag and drop operations using protractor are implemented using *browser.actions()*

```
browser.actions().dragAndDrop(elem, target).perform();
```

where elem=Tile to be dragged and target=Tile where the elem is to be dropped. Instead of specifying the target element, we can specify offset in pixels also.

```
browser.actions().mouseMove(element).mouseMove({x: 50, y: 0}).
  doubleClick().perform();
```

The above example will double click to the right of an element. Link to the external reference is [here](#). The interaction specific tests are added in **./DragAndDropSort/protractor.js** by implementing the following functions:

- **expectInteractionDetailsToMatch**: This function verifies that the interaction is displayed correctly in the player.
- **submitAnswer**: This function simulates the user submitting an answer to the interaction.

- **answerObjectType**: It is the type of the object returned by the interaction. This should match the **answer_type** defined in the `DragAndDropSort.py`. In this case it is the `'SetOfDragAndDropElements'` object type.
- **testSuite**: It is an array of dictionaries each of which describes the scenario in which the interaction is used and specifies how it should behave. The test moves to the player checking whether the interaction is behaving as expected and submits a series of right and wrong answers, and verifies that they are handled as expected. It includes the following dict keys within a list:

```
testSuite = [{
    interactionArguments: [],
    ruleArguments: [],
    expectedInteractionDetails: [],
    wrongAnswers: [],
    correctAnswers: []
}];
```

Adding Rules

The rules defined for an interaction takes the learner's response, normalizes it and evaluates it against the predicate. Each response is verified against the set of rules and handled accordingly.

Rule Specs and Answer verification:

When the learner drops all the tiles in the desired area and submits his answer, the frontend list will be created containing the unique 'id' and non-unique 'position_no' for each tile. This list will be compared with the original answer as set by the creator.

Technical Implementation of the Rules:

- We start by adding rules definitions for this interaction as a JSON dictionary to **`extensions/interactions/rule_templates.json`**.


```

"DragAndDropSort": {
  "IsEqualToOrdering": {
    "description": "is equal to {{x|SetOfDragAndDropElements}} ordering"
  },

  "IsEqualToOrderingWithAtmostTwoElementsInWrongPlace": {
    "description": "is equal to {{x|SetOfDragAndDropElements}} ordering" +
      "with at most two elements in the wrong place"
  },

  "HasElementXAtPositionY": {
    "description": "has element {{x|DragAndDropElement}} at position" +
      "{{y|NonnegativeInt}}"
  },

  "HasElementXBeforeElementY": {
    "description": "has element {{x|DragAndDropElement}} before element" +
      "{{y|DragAndDropElement}}"
  }
}

```

- Add factory service **dragAndDropSortRulesService** to `./directives/DragAndDropSort.js` . It contains the functions that evaluate rules described in the `rule_templates.json` (ex- `IsEqualToOrdering`, `HasElementXAtPositionY` etc.) and returns a Boolean value (True if the rule is satisfied, False otherwise). The implementation of each function is:
 - **IsEqualToOrdering**: This function receives 'answer' and 'input' lists as input arguments. The submitted answer list is compared with the input list (expected answer) matching unique 'id', 'content' and its corresponding non-unique 'position_no'. It then returns 'true' if there is an exact match, false otherwise.
 - **IsEqualToOrderingWithAtmostTwoElementsInWrongPlace**: This function also receives 'answer' and 'input' lists as the arguments. The submitted answer list is matched with correct input answer list comparing corresponding 'id' and 'position_no'. The variable 'no_of_unmatches' keeps record of how many 'id's have different 'position_no' than expected. If no_of_unmatches are less than or equal to two, it returns true. Otherwise false.
 - **What is the rationale for choosing at most two elements here? Why not one?**
It is quite obvious that if one element is at the incorrect position in the list, then at least 1 other element will also be at incorrect place i.e. they both will be swapped while at most all other (n-1) elements can be at the incorrect place.

- **HasElementXAtPositionY:** This function receives 'answer' and 'input' as the input arguments. 'input.x' contains 'id' of element X and 'input.y' contains expected 'position_no' of X. The 'position_no' in the submitted answer corresponding to the input.x 'id' is compared with input.y . If they happen to be same, this function returns true. Otherwise, false.
 - **HasElementXBeforeElementY:** This function also receives 'answer' and 'input' as arguments. The 'input.x' contains 'id' of element X while 'input.y' contains 'id' of selected element Y. The 'position_no' for both the ids- input.x and input.y are compared. If position_no for input.x is less than that of input.y, it returns true. Otherwise, false.
- Add **./directives/DragAndDropSortRulesServiceSpec.js** for testing the rule specs whose functions are defined in **dragAndDropSortRulesService** factory service.
 - Add **./directives/DragAndDropSortValidationService.js**: This is a validation service for this interaction. It checks the submitted answer for the given rule specs and validators as described in the Rule Specs section and give appropriate warnings as and when necessary.
 - Add **./directives/DragAndDropSortValidationServiceSpec.js** for testing the validator service for this interaction. It sets certain correct and wrong inputs and checks whether the interaction is behaving as expected giving appropriate warning messages.

Explanations:

[1] Why list of dicts is preferred over list of lists in the object model of Drag and Drop sorting interaction?

We generally use dicts over lists when the indexes have special meaning besides just the positional placement. Here the case is same. We have meaningful indexes- 'content', 'id' and 'position_no' for each element (tile) in the interaction. So it makes more sense to give meaningful names to the indexes in the dict rather than dealing with the integer indexes so that the code can be more meaningful and understandable.

[2] Why relative positioning of the tiles is preferred over absolute positioning?

The relative positioning technique for sorting the tiles is preferred over absolute positioning due to the following reasons-

- It solves the problem when the learner sorts the tiles and later on realizes that it is incorrect and hence, he has to reorder them. With this implementation, he does not need to shift all the tiles for reordering.

- At max. 9 tiles are shown on the screen. It will solve the problem faced by the learners while sorting as all the tiles will be shown to them. Problems would arise when only few of the tiles are shown to the learners in absolute positioning.
- There will be no scrolling required here except when the number of the tiles exceeds 9.

Now let's come to the second interaction that this project aims at.

Number with Units Interaction

Current workaround:

Creators use a number input field and ask students to write just the number. This is suboptimal because it doesn't give students practice in actually working with units.

The **Number With Units interaction** will give more sense of units to the learners. It will allow the learner to answer as the numeric or fraction type input along with the units. The detailed documentation can be found [here](#). However, all the important technical aspects have been covered here as well.

Solution:

To solve this problem we will create a new Number with Units interaction which is the extended version of Numerical Input and Fraction Input interaction with units as well.

- Units will be added as a customization for extended numerical/fractional input. This will be done by adding 2 new rule specs 'answer with unit is equal to' and 'answer with unit is equivalent to'. These rule specs will take an extra input string from creator which is unit of that answer. Then this unit will be converted to dict representation in frontend and sent to the backend storage (see backend storage [below](#)).
- When the user plays an exploration that has numeric answer with units, the answer stored in backend will be transferred to frontend. This will contain 'type', 'value' and 'unit' fields of the respective answer (even if the answer is just a number without any unit associated with it. These 3 fields will be sent to frontend always with unit set to null).
- When user submits an answer, it will be of following form.

<beginning_unit> numeric_value <ending_unit>

Thus the answer will be divided into respective fields and will be verified.

- We should also provide an option for the fixed units. This will be required when creator is teaching unit conversions himself. See 'answer with units is equal to' rule in the [Answer Verification](#) section.

Let's start with the creator's journey first. The creator first starts by choosing **Number with Units interaction** from the 'Add Interaction' section while creating the exploration on the Creator's dashboard. He will then encounter the Add Response page where he can set the expected answer with rule specs either of type- 'is equal to' or 'is equivalent to'. The page will look like-

The creator chooses the desired rule spec out of the two and enters the desired answer in the Number with Units editor. The creator is allowed to write numeric value as well as the unit. However, he may choose to just write the numeric value only in accordance with the question asked.

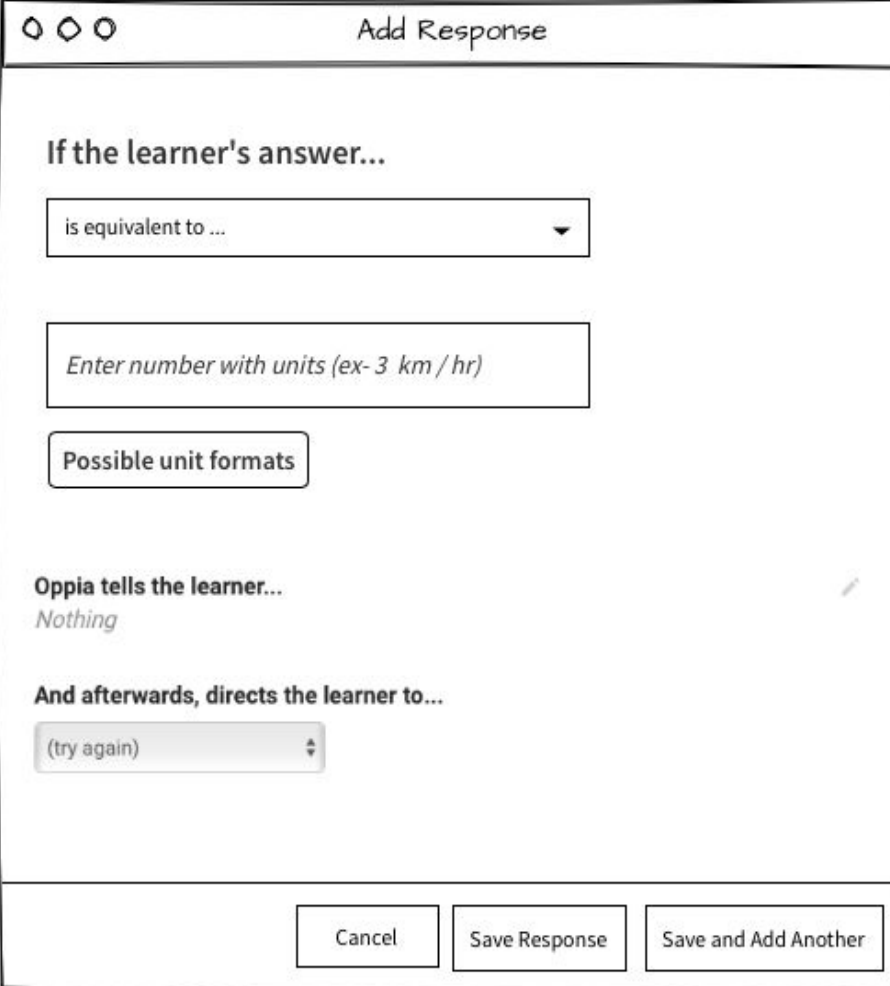
What is the rationale behind entering both the value and units in the same editor?

- It gives creator the freedom to just set numeric value as the answer without units if he chooses so in accordance with the question asked.
- Secondly, the units can be at the beginning (ex- \$, Rs. etc.) or at the end (ex- kg, m etc.). So it will be better to have just one editor allowing all these functionalities. If we use separate editors for value and units, then we will have to write the beginning units at the end which is not the expected behaviour.

In case the creator wants to look at the unit formats, he can get help by clicking on the '*Possible unit formats*' button. The description of the unit formats is provided [here](#) in the learners view section. Moreover, the creator can set the Oppia's response to the answer and finally sets the path where the card redirects to. At this point the creator successfully sets all the parameters for

the interaction. The response is saved by clicking on 'Save response' button and the creator can see its preview in the preview section. Note that the preview will look like the [interaction](#) as in the learner view described in the learner's journey.

For the rule spec- 'answer is equivalent to...', the 'Add Response' section will look like-



The screenshot shows a dialog box titled "Add Response" with three window control buttons in the top-left corner. The dialog is divided into several sections:

- If the learner's answer...**
 - A dropdown menu with the text "is equivalent to ...".
 - A text input field containing the placeholder text "Enter number with units (ex- 3 km / hr)".
 - A button labeled "Possible unit formats".
- Oppia tells the learner...**
 - The text "Nothing" is displayed, with a small edit icon to its right.
- And afterwards, directs the learner to...**
 - A dropdown menu with the text "(try again)".

At the bottom of the dialog, there are three buttons: "Cancel", "Save Response", and "Save and Add Another".

Here also the creator can set the expected answer value with the units and submits the answer along with the Oppia's response. The answer can be validated for the equivalency of both the value and the units as described [here](#).

Now let's come to the learner's journey. The learner finds the Number with Units interaction like:

The diagram shows a rectangular interface for a 'Number with Units' interaction. On the left side, there is a small circle. To its right is a larger rectangular box containing the following elements from top to bottom: a box labeled 'Question', an input field with the placeholder text 'Enter number with units (ex-3 km / hr)', a button labeled 'Possible unit formats', and a button labeled 'Submit'.

Number with Units interaction is displayed in an inline mode here. The learner can enter his answer according to the placeholder shown in the interaction editor. The placeholder depicts the correct format for entering the units. For further help, the learner can click on the 'Possible unit formats' button to get the help about all the possible unit formats. He will see the unit formats like the one in the existing Logic Proof interaction-

Rules and possible unit formats

Rule Description	Incorrect Format	Correct Format
Separate value and the unit with one space.	3km	3 km
In case the value is a fraction, separate numerator and denominator by forward slash '/' symbol.	3 by 5 km	3/5 km
For multiplying two or more units, put one space or asterisk (*) between them.	5 J-s 5 J . s	5 J * s Or 5 J s
Separate numerator and denominator of the units using '/' symbol. Keep one space on either sides.	3 km/hr 3 km per hr	3 km / hr
For adding dimensions to the units, use caret (^) symbol to add both positive or negative powers.	5 kg / m ⁽⁻²⁾ 5 kg / m ⁽²⁾ 5 kg / m ^{**2}	5 kg / m ⁻² 5 kg / m ² 5 kg / m ^{^2}
In case of multiple units in the denominator, enclose them within the parenthesis.	5 kg / m * s 5 kg/m*s	5 kg / (m * s) Or 5 kg / m / s
For writing beginning units (like currencies), separate unit and value with one space.	\$5 5 \$ 5\$	\$ 5

Moreover, the presubmit answer validation will also happen here so that the learners do not get stuck with the format in case they are entering the answer in the wrong format. The editor glows red each time the learner enters any answer in an invalid format. Ex-

Answer Validation :

- **3km** will make the editor turn red as units and numeric value are not separable here.
Correct: **3 km**
- **9.8 m / s ^ 2** will also be invalid as the answer contains two numeric values.
Correct: **9.8 m / s²**.
- **km** will be invalid as unit with no value is not allowed. However, reverse is allowed if the creator chooses to keep the value only as the answer with no units.

Note that this is the validation of the answer format only. The actual correctness of the answer is verified only when the learner submits his answer. After entering the answer in the valid format the submit button will become functional and the learner is free to submit his answer.

Now let us discuss its technical implementation in detail.

Technical Implementation

This section describes technical implementation of the solution mentioned above. Note that the approach at some sections is inspired from Prasanna's original docs. However, the technical implementation with the appropriate code snippets added here makes the plan more concrete.

Backend storage:

- This type of information can not be stored with current 'Real' data type. Both units and fractions will contain non-numeric part (text characters) which are incompatible with current Real data type.
- To solve this we will add new backend data storage of the following schema:
 - 'type': 'real' or 'fraction'
 - 'value': value will be different based on the 'type' of answer. This field will contain two more keys.
 - 'real': If type is 'real' then this will store answer value else None.
 - 'fraction': If the type is 'fraction' then this will store whole number, numerator and denominator of the answer else None. This is the existing Fraction schema as defined in 'objects.py'.
 - 'unit': dict representation of unit for e.g. kg m s⁻² will be stored as:

```
{'kg': 1, 'm': 1, 's': -2}
```

Units can also be stored as Unicode string instead of dict. But we have preferred dict representation over unicode string. [Here](#) is the reason.

To add new backend data storage, we will add **NumberWithUnits** class to extensions/objects/models/objects.py -

```

class NumberWithUnits(BaseObject):
    default_value = {
        'fraction': Fraction.default_value,
        'real': 0.0,
        'unit': {}
    }

    SCHEMA = {
        'type': 'dict',

        'properties': [{
            'name': 'type',
            'schema': {
                'type': 'str'
            }
        }, {
            'name': 'fraction',
            'schema': Fraction.SCHEMA
        }, {
            'name': 'real',
            'schema': {
                'type': 'float'
            }
        }, {
            'name': 'unit',
            'schema': {
                'type': 'dict'
            }
        }
    ]
}

```

Thus, the answer in the JSON format will contain 'type', 'fraction', 'real' and 'unit' parts.

We need to create inline Number with units editor for this interaction. The following files will be added to **extensions/objects/templates**:

- **number_with_units_editor.html**: It contains templates for number with units interaction. It determines how the interaction actually looks like.
- **NumberWithUnitsEditor.js**: It contains directive for controlling the overall functionality of this interaction.

New files to be added:

The following new backend and frontend files will be added under **oppia/extensions/interactions/NumberWithUnits**:

Creating Interaction

- **NumberWithUnits.py** : This is a Python configuration file that sets various attributes for the interaction. These attributes are documented in *extensions/interactions/base.py*. Some of the attributes are:

```
display_mode = base.DISPLAY_MODE_INLINE
answer_type = 'NumberWithUnits'
can_have_solution = True
show_generic_submit_button = True
```

- This interaction will have display mode inline i.e. it will appear within the same card below the question like the existing NumericInput interaction.
 - The answer_type will be the object 'NumberWithUnits' defined in *objects.py*.
 - It will definitely need a generic submit button and can have a solution.
- **./directives/NumberWithUnits.js** : It contains two directives that define the behaviour of the interaction- **oppiaInteractiveNumberWithUnits**, **oppiaResponseNumberWithUnits** and a factory service called **numberWithUnitsRulesService**. The former directive defines the view that is presented to the learner to interact with.
 - ❑ It keeps a continuous watch whenever the answer is written within the interaction editor using *\$scope.\$watch()*.
 - ❑ Checks for the validity of an answer using *\$scope.isAnswerValid*. *\$scope.answer* contains the learner's answer in the dict format. It prevents submission of undefined and null 'float' or 'fraction' values in an OR condition since the answer can be of the 'float' or 'fraction' type but not both simultaneously.
 - ❑ Finally, submits an answer to the server if it is valid using *\$scope.submitAnswer*.

While the latter ('response') is the view that is shown to the learner after the interaction has taken place. It basically receives the answer within *\$scope.answer* variable. The factory service is described in the [Creating Rules](#) section.

- **NumberWithUnits.html** : It contains two templates used to display the interaction's HTML, each of which is bounded to the directives in *NumberWithUnits.js* and *NumberWithUnitsValidationService.js* .

Once these files are created, the interaction can be activated by adding interaction's name- **NumberWithUnits** to the `ALLOWED_INTERACTION_CATEGORIES` variable in '**feconf.py**' under 'Math' category. Then, the interaction should be available in the interaction repository, and can be used in the explorations.

Adding Tests

Oppia uses end-to-end testing framework for testing. These tests mimic the actual users by interacting with the web page normally by clicking and typing. Then the expected behaviour of the interaction is checked. The interaction specific tests can be enabled by adding the interaction to **extensions/interactions/protractor.js** .

The interaction specific tests are added in **./NumberWithUnits/protractor.js** by implementing the following functions:

- **expectInteractionDetailsToMatch**: This function verifies that the interaction is displayed correctly in the player.
- **submitAnswer**: This function simulates the user submitting an answer to the interaction.
- **answerObjectType**: It is the type of the object returned by the interaction. This should match the **answer_type** defined in the `NumberWithUnits.py` . In this case it is '*NumberWithUnits*' object type.
- **testSuite**: It is an array of dictionaries each of which describes the scenario in which the interaction is used and specifies how it should behave. The test moves to the player checking whether the interaction is behaving as expected and submits a series of right and wrong answers, and verifies that they are correctly handled as expected.

Adding Rules

The rules defined for an interaction takes the learner's response, normalizes it and evaluates it against the predicate. Each response is verified against the set of rules and handled accordingly.

Rule specs:

- The following new rule specs should be added for this new feature:
 - **Answer with unit is equivalent to**: This will check whether the submitted answer is equivalent to the expected answer. This includes conversion of one unit into other if required.
 - **Answer with unit is equal to**: This will check that submitted answer and expected answer both have equal unit and equal value. This does not allow conversion between units.

Answer verification:

- Submitted answer will be divided into 3 fields: 1. `beginning_unit`, 2. `value` and 3. `ending_unit`.
- Answer verification will be performed in 2 levels.
 1. This level will check that the unit that comes at the beginning of the answer is a valid unit. For e.g. '\$' or 'Rs.' can come at beginning of the answer but 'K', 'N', 'kg' can not come at beginning.
 2. Verification for '**answer with unit is equal to**' rule:
 - Perform step 1 as shown above.

- This level will combine unit present at the beginning of answer and at the end of answer in a single unit (this is done for both expected answer's unit and submitted answer's unit). Then dict representation of both the units will be compared to make sure that they have same units and same power for each unit.
- Next value of expected answer and submitted answer will be checked for equality. Value can be of 'Fraction' or 'Real' type as described in the Backend storage section.

3. Verification for 'answer with unit is equivalent to' rule:

- Perform step 1 as shown above.
- Then both (expected and submitted) units will be converted to string representation from their equivalent dict representation.
- Then create unit quantity for expected (answer, unit) and unit quantity for submitted (answer, unit). This will be done using one of the 3rd party libraries- math.js, check here for more [info](#). Check for both quantity's equality (using equals method, quantity contains both number and its unit so *equals()* check for the total equality, not just unital or numerical).

Ex-

```
var a = math.unit('2 cm');
var b = math.unit('20 mm');
b.equals(a); // returns true
```

- Answer will be correct if both the submitted and expected answers are equivalent (use math.js to check equivalency using *equals()* method), otherwise incorrect.

Technical Implementation of the Rules:

- We start by adding rule definitions for this interaction as a JSON dictionary to `extensions/interactions/rule_templates.json`.

```
"NumberWithUnits": {
  "IsEqualTo": {
    "description": "is equal to {{x|NumberWithUnits}}"
  },
  "IsEquivalentTo": {
    "description": "is equivalent to {{x|NumberWithUnits}}"
  }
}
```

- Add factory service **numberWithUnitsRulesService** to `./directives/NumberWithUnits.js` . It contains functions that evaluate rules described in the `rule_templates.json` and returns a Boolean value (True if the rule is satisfied, False otherwise).
- Add `./directives/NumberWithUnitsRulesServiceSpec.js` for testing the rule specs whose functions are defined in **numberWithUnitsRulesService** factory service.

Convert unicode string unit to dict representation:

- This is the python [code](#) that implements the conversion of a unicode string of the units entered to the dict representation.

Currency units:

To represent the currency related units (which are not part of SI units), `math.js` does not support them directly. We will have to create custom units to represent the currency (fortunately `math.js` supports custom units).

We will create a base unit i.e. for \$ (USD) → `math.createUnit('USD', {aliases: ['$']})`. Since this is a currency unit it cannot be expressed in terms of SI units. Here the US Dollar becomes the base unit. The 'aliases' means that the learner can use both 'USD' and '\$' to specify USD. Similarly we can add other custom base units as per the need.

Then create custom units with reference to the base units i.e. for **cents** → `math.createUnit('cents', '0.01 USD')`. Create custom units for which conversion rate from the base unit is constant (here **1 USD = 100 cents** is a constant conversion rate). This should be done for all units which have constant exchange rates. Clearly, it can not be applied to the currencies whose exchange rate varies like 'USD' and 'Rupees'.

Explanations:

[1] Why dict representation of the units is preferred over unicode string?

The reason is the implementation of the answer verification rules. As suggested, `math.equals()` provides easy approach to check our answer with units but only for the equivalency. It can not be strict to check only for equality. So we have to check the correct answer in dict format checking units and their powers for exactly equal answer. Clearly we can not do this with unicode string as there can be different approaches for writing the units with acceptable format. Ex- **kg / m / s** and **kg / (m * s)** both are correct. But on comparing unicode strings they will be incorrect and hence correct answer will not be accepted.

Complete Workflow (Milestones):

This section includes the complete workflow through each of the suggested milestones. The detailed implementation plan has already been provided in the above sections. Please note that NumberWithUnits is a new interaction altogether inspired partially from NumericInput and partially from FractionInput along with its new features for units.

During the Community Bonding period (April 24 - May 13), I will finalize all the technical documentation sheets and strategy required to be followed with the mentors consent and reviews. Also, I will make a task sheet so that I as well as the mentors can keep track of the project progress and will start the project implementation at my earliest to avoid any delays in the future.

Milestone 1: (May 14 - June 10)

- Summary

In this milestone, we will implement a preliminary version of 'NumberWithUnits' interaction from scratch. This includes-

- Creation of new backend object model as described above.
- Addition of new Python config file NumberWithUnits.py.
- Addition of the directives for both before and after the learner's response.
- Addition of two new rule specs- 'answer with unit is equal to' and 'answer with unit is equivalent to'.
- Implement functions to convert string to dict representation of the unit.
- Implement functions to support for the SI units (as supported by math.js library) as well as the support for the conversion between the units. See point 3 of the answer verification section.
- Implement functions that will check that the expected and submitted unit are equal (not equivalent, see step 2 of answer verification). Basically implement verifier function for the rule spec 'answer with unit is equal to.'
- Implement functions that will check that the expected and submitted answers are equivalent. Basically implement verifier functions for rule spec 'answer with unit is equivalent to.'
- Addition of the factory service for the validation and verification of the rule specs.

The subdivision of the tasks is as follows-

- **Milestone 1.1 (May 14 - May 22)**

- Code:**

- Add new object model for 'Number With Units' in /extensions/objects/models/objects.py
- Add new HTML template 'number_with_units_editor.html' in /extensions/objects/templates for the editor.

- ❑ Add directive 'NumberWithUnitsEditor.js' corresponding to the HTML template in /extensions/objects/templates.
 - ❑ Add 'NumberWithUnits.py' Python configuration file in /extensions/interactions/NumberWithUnits.
 - ❑ Add 'oppiaInteractiveNumberWithUnits' and 'oppiaResponseNumberWithUnits' directives in /extensions/interactions/NumberWithUnits/NumberWithUnits.js
 - ❑ Add a Rule Spec for NumberWithUnits interaction- 'answer with unit is equal to'.

- ❑ **Tests:**
 - ❑ Add end to end testing functionality (defined in protractor.js).
 - ❑ Add './NumberWithUnits/protractor.js' to incorporate unit tests for NumberWithUnits interaction. The expected behaviour of the interaction is checked here by clicking, typing and mimicking the user. It covers all the basic tests for the proper functioning of the directives as well.

- **Milestone 1.2 (May 23 - May 28)**
 - ❑ **Code:**
 - ❑ Add new factory service 'numberWithUnitsRulesService' for the validation of the two rule specs in /extensions/interactions/NumberWithUnits/NumberWithUnits.js
 - ❑ Add evaluation functions for rule specs as described in [Answer Verification](#) section.
 - ❑ Implement function to convert unit from string to dict representation.
 - ❑ Implement conversion function to convert input (answer) string to the backend data storage object. Add the previous function to convert unit from string to dict type here.
 - ❑ **Tests:**
 - ❑ Add tests to check for the conversion of the input units from string to dict type.
 - ❑ Add './directives/NumberWithUnitsRulesServiceSpec.js' that contains tests for the rule specs defined within 'numberWithUnitsRulesService' factory service.

- **Milestone 1.3 (May 29 - June 10)**
 - ❑ **Code:**
 - ❑ Implement function to convert unit from dict to string representation.
 - ❑ Add third party library 'math.js' in Oppia.
 - ❑ Implement functions to support for the SI units (as supported by math.js library).
 - ❑ Add function to support the conversion between the units. See [point 3](#) of the answer verification section.

- ❑ Add new Rule Spec for NumberWithUnits interaction- 'answer with unit is equivalent to'.

- ❑ **Tests:**

- ❑ Add './directives/NumberWithUnitsRulesServiceSpec.js' that contains tests for each and every rule specs defined within 'numberWithUnitsRulesService' factory service.
- ❑ Add tests to check for the conversion of the input unit from dict to the string type.

During the review period, I'll finalize all my workflow for the next milestone (with the mentor's consent) and install dependencies (if any). If the time permits, I'll start implementation also so as to complete everything on time.

Milestone 2: (June 12 - July 9)

- Summary

In this milestone, we will implement a preliminary version of 'DragAndDropSort' interaction from scratch. This includes-

- Creation of new backend object model as described [above](#).
- Implement supplemental editor for Drag and Drop Sorting interaction. It includes creation of HTML template and its corresponding directive having functions for dragging and dropping the tiles.
- Addition of new Python config file DragAndDropSort.py.
- Addition of the directives for both before and after the learner's response.
- Addition of four new rule specs- 'answer is equal to this ordering', 'answer is equal to this ordering with at most two elements at the wrong place', 'answer has element X at position Y in the list', and 'answer has element X coming before element Y in the list'.
- Implement functions that will check that the expected and submitted answer sequences are equal. Basically implement verifier functions for the rule spec 'answer is equal to to this ordering' and 'answer is equal to this ordering with at most two elements at the wrong place'.
- Implement functions that will check for the constraints added to the selected element X in the expected and submitted answer lists. Basically implement verifier functions for the rule specs- 'answer has element X at position Y in the list' and 'answer has element X coming before element Y in the list'.
- Addition of the factory service for the validation and verification of the rule specs described above.

- **Milestone 2.1 (June 12 - June 22)**

- ☐ **Code:**

- ☐ Add new object model for 'Drag and Drop Sorting interaction' in /extensions/objects/models/objects.py
 - ☐ Add new HTML template 'drag_and_drop_editor.html' in /extensions/objects/templates for the editor.
 - ☐ Add directive 'DragAndDropEditor.js' corresponding to the HTML template in /extensions/objects/templates.
 - ☐ Add 'DragAndDropSort.py' Python configuration file in /extensions/interactions/DragAndDropSort.
 - ☐ Add 'oppiaInteractiveDragAndDropSort' and 'oppiaResponseDragAndDropSort' directives in /extensions/interactions/DragAndDropSort/DragAndDropSort.js

- ☐ **Tests:**

- ☐ Add end to end testing functionality (defined in protractor.js).
 - ☐ Add './**DragAndDropSort/protractor.js**' to incorporate unit tests for Drag and Drop interaction. The expected behaviour of the interaction is checked here by clicking, putting the tile on the drop location and mimicking the user. It covers all the basic tests for the proper functioning of the directives as well.

- **Milestone 2.2 (June 23 - June 29)**

- ☐ **Code:**

- ☐ Add new Rule Specs for DragAndDropSort interaction- 'answer with unit is equal to'.
 - ☐ Add second Rule Specs - 'answer is equal to this ordering with at most two elements at the wrong place' for the interaction.
 - ☐ Add new factory service 'DragAndDropSortRulesService' for the validation of the rule specs in /extensions/interactions/DragAndDropSort/DragAndDropSort.js

- ☐ **Tests:**

- ☐ Add unit tests in ./directives/**DragAndDropSortRulesServiceSpec.js** for testing validator service defined in 'DragAndDropSortRulesService'.

- **Milestone 2.3 (June 30 - July 8)**

- ☐ **Code:**

- ☐ Add new Rule Specs - 'answer has element X at position Y in the list' for the interaction.
 - ☐ Add new Rule Specs - 'answer has element X coming before element Y in the list' for the interaction'.

- ❑ Add these Rule Specs described above to 'DragAndDropSortRulesService' in /extensions/interactions/ DragAndDropSort/ DragAndDropSort.js.
- ❑ Add validation service for the interaction in **./directives/ DragAndDropSortValidationService.js** . It checks the submitted answer for the given rule specs and validators as described in the Rule Specs section and give appropriate warnings as and when necessary.
- ❑ **Tests:**
 - ❑ Add unit tests in **./directives/ DragAndDropSortRulesServiceSpec.js** for testing 2 new Rule Specs that will be added in this milestone.
 - ❑ Test that the submitted answer is correctly verified for the rule spec- 'answer has element X at position Y in the list'.
 - ❑ Test that the submitted answer is correctly verified for the rule spec- 'answer has element X coming before element Y in the list'.
 - ❑ Add **./directives/ DragAndDropSortValidationServiceSpec.js** for testing the validator service for this interaction. It sets certain correct and wrong inputs and checks whether the interaction is behaving as expected giving appropriate warning messages.

Milestone 3: (July 13 - August 5)

Milestone 3.1: (July 13 - July 23)

- Summary:

In this milestone, the advanced features like support for the custom (non-SI) units will be added such as currencies. The math.js library plays a very important role in this implementation. It provides method for adding our own custom units- example for \$ (dollar) → *math.createUnit('dollar')* .

 - Add required custom units (base unit + custom units). Ex- US Dollar(\$) is the base unit while Cent is a custom unit.
 - The custom unit can be verified against the base unit using *math.eval()* function in math.js library. So, no external conversion function is needed.
 - Add functions for answer verification in case the value is of 'Fraction' type. The fraction is converted into the decimal notation first and then can be validated using math.js library for the value as well as for the unit.
- ❑ **Code:**
 - ❑ Add required custom currency units (as shown in [currency units](#) section).

- ❑ Add function that converts fraction type input to floating type and ultimately calls answer validation function as suggested in milestone 1.

- ❑ **Tests:**

- ❑ Add tests for the verification of the (Value, Custom Unit) against the expected correct answer.
- ❑ Add tests for the conversion of fraction into float type value and ensure that it is validated against the expected answer with units.

Milestone 3.2: (July 24 - July 30)

- **Summary:**

In this milestone, more advanced features for Drag and Drop sorting interaction like two or more tiles occupying the same position in the sequence will be implemented. Ex- if you want to place [1/2, 1/3, 2/4] in the ascending order: the tiles [1/2, 2/4] will occupy the same positions actually. This will be set by the creator by assigning same 'position_no' to all of these tiles during the response addition though their 'id' will remain unique for distinction. These tiles can be put one over the other (but not completely) as described in the Synopsis of the design mockups section. It will require:

- Addition of the functionality so that the tiles can be put over each other.
- Addition of answer verification functions in case the tiles have same 'position_no' but unique 'id'.

- ❑ **Code:**

- ❑ Add function so that the tiles can be put one over the other in the interaction editor 'DragAndDropEditor.js'.
- ❑ Add corresponding changes to the 'drag_and_drop_editor.html'.
- ❑ Add validation service in **./directives/DragAndDropSortValidationService.js** corresponding to the answer validation functionality.

- ❑ **Tests:**

- ❑ Add service specs in **DragAndDropSortValidationServiceSpec.js** for the validation service implemented in this milestone.

During July 31 - August 5, I will finalize all my work and try to get everything merged within time so as to end towards the successful GSoC project.

Summer Plans

Which timezone(s) will you primarily be in during the summer?

- During the GSoC period, I will be working from India (GMT + 5:30).

How much time will you be able to commit to this project?

- In the months of May, June and July, I will be able to work 8-9 hours per day (weekdays and weekends both). During this period, there will be vacations in my University and so I will be flexible enough to manage things if they don't get completed on time.
- In the last week of July and August, I will be able to work for 6-7 hours per day as my next semester will begin. However, I will be able to devote 8-9 hours on weekends. So, I will try to complete most of my work before this period as mentioned in the milestones.

What jobs, summer classes, and other obligations might you need to work around?

- During the entire GSoC period, I've no commitments. I will neither have any classes nor jobs till the last week of July. After that my next semester will begin but I will manage to work for 6-7 hours per day. However, I will be flexible enough on the weekends.

What is your contact information, and preferred method of communication?

- Email: agarwalvibhor84@gmail.com
- Github handle (gitter): [vibhor98](#)

My preferred mode of communication will be gitter and Email. I will be available through Hangouts also.

How often, and through which channels, do you plan on communicating with your mentor?

- I will be in constant touch with the mentors via Gitter and Email. There can be biweekly or weekly discussions (based on mentors) on the workflow and issues (if any). Meetings can be held on Gitter. Moreover, to have a healthy continuous interaction, I will be giving updates in every two days or will update tasks sheet on the daily basis (based on the mentors) so that the mentors can keep track of the progress of the project.

Future Plans

There are a lot of different ways to make these interactions even more amazing and helpful for the learners. Due to the time constraints I have not included them in the milestones. Below are some of the ideas that I'll try to complete if everything works fine within a duration of 3 months. Else we can work after the GSoC period also to make them even better.

- **Add rule to restrict the object type for each tile in Drag and Drop interaction:**

Currently each tile can have image, text input, numeric input, fractions or LaTeX in a single question of the exploration. However, we can restrict it to one type only. During the GSoC period, we will be doing continuous testing so that the end product is usable and beneficial for our target learners. As per the need, the creator can choose one of the above mentioned types for the tiles. The other options will be disabled so that the creator can choose that type for input only. Based on the feedbacks, I can implement it within the GSoC period as well.

Ex- Say creator chooses image input. Then all other input types will be disabled by customizing our existing Rich Text Editor. This will ensure that the object will be of one type only.

- **Add more generic Rule Specs for Drag and Drop answer verification:**

To enhance the above mentioned rules, we can add more generic rules like-

- 'Answer is equal to this ordering with at most n elements at the wrong place'
- 'Answer is equal to this ordering with at least n elements at the wrong place'
- 'Answer is equal to this ordering with exactly n elements at the wrong place'

We can thus extend the rules imposed on 2 elements to n elements. Initially the creator will choose this n value and accordingly the n elements will be checked for the answer verification. This will help in giving better responses to the learner after the answer is submitted.

Thank You!