

Google Summer of Code 2019

Asking students why they picked a particular answer **(Oppia)**

Personal Details:

Name : Anubhav Sinha

Email : anubhavsinha98@gmail.com

GitHub Handle: [anubhavsinha98](https://github.com/anubhavsinha98)

College: Jaypee Institute of Information Technology, Sec 62, Noida

Program: B.Tech in Computer Science(4 year course)

I am currently pursuing my 2nd year of B.Tech in Computer Science at Jaypee Institute of Information Technology. IIIT Noida is one of the renowned institutes for computer science in the Delhi NCR. I am having a cumulative GPA of 8.9 out of 10 i.e securing as follow

1st Semester : 8.1 SGPA, 2nd Semester: 9.5 SGPA, 3rd Semester : 9.3 SGPA.

I am a member of IEEE Student Branch IIIT Noida and Open Source Developers Club of my college. I have been working on Python since 2 years and JavaScript for 1 year. I am a full stack developer. I started my web development journey during the mid of 2018. My other skills are Angular JS, C++, PHP, Node.js, GUI Development using Python, CSS & Photoshop.

Project Details:

Project Name: [Asking students why they picked a particular answer](#)

Why I am interested in working with Oppia:

Oppia's mission is to help anyone learn anything they want in an effective and enjoyable way. And I want to contribute towards the mission so that via Oppia students can learn the various subjects and its topics. Oppia also helps educators around the world who want to express their knowledge and make some interesting explorations so that they can help the students. And explorations are much interesting to learn a topic/subject, as explorations include videos, images, graphics, interactive questions which helps in creating interest among the students. I also prefer this kind of learning, as it helps in boosting critical and natural thinking.

And the team at Oppia is very helpful and friendly. The support from the Oppia team is truly amazing, this helps the first time contributor to merge a successful PR which proves as a motivational boost for the contributor. I would conclude that the whole team of Oppia is really helpful and cooperative.

Also I like the way that every new contributor is assigned to a mentor, which helps the contributor to directly ask for help to the mentor.

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

Questions are the best parameter to judge a person that he knows about that topic. Whenever a person makes an online course, he adds various tests at every step in the course to ensure that the learner has learned the previous topics. But sometimes it happens that the learner is not able to attempt that question correctly in 2-3 attempts which must not be expected by the creator of the course, as he has added the questions according to the course he has created. But the actual insights can only be provided by the learner who is solving the questions related to the course. So in this case if a learner is able to provide a response to the creator about his approach that what is leading to the submission(i.e how the learner landed on this answer) it would be really helpful to the creator to reach out those responses filled by the learner, so that he can edit the course accordingly which will help in increasing the learner experience and in future he will be able to make more good courses which will improve the feedback or review from the learner. And through this response, the learner will also be able to analyze what is the reason for their submission.

The best part of this project is that the creator will get to know how the learners are applying their approach, same as what happens in actual class when students discuss their approach for attempting a question from their teachers such that it helps the creator to make the course/exploration more user-friendly.

So I think this project will help the creators of the Oppia to enhance their explorations, enriching the learner's experience, which will overall improve the Oppia's feedback.

The learner will also be able to analyze more about their happenings of the wrong submission.

Prior Experience:

I have worked on the following projects:

- **Get It Registered** - A python project which includes an interface built using Tkinter module, which is currently being used in my college by various hubs in order to do registration of students for any event conducted in college.

- **Library Management** - Web development project which includes the management of the Library, like the students can view the books are currently available in Library and can hold that book if they want to issue that book, but for a limited period only. Backend was built in PHP, Database used was MySql and frontend with HTML, CSS and JavaScript.
- **Learn Blockchain** - A website which help teenagers to learn about the Blockchain technology. Various UI interactions, games using javascript are build to help the teenagers learn blockchain technology easily.
- **Best Route** - Data Structures project on finding the best route from destination airport to the arrival one. Built in C++, Binary Heaps, Priority Queue and Graphs were the data structures used.
- **Bill Split** - Project built on C. Helpful for splitting the bill for a group of people. User account is also maintained, such that all the transactions can be viewed done by the user. UI was also built using C.
- **Task Killer** - Basic task killer which kills the tasks by clicking it, built in python.

I am an active member of Open Source Developers Club, IIIT Noida, and IEEE Student Branch IIIT Noida. I have also taken lectures on Python and C++, held by my college under the workshops conducted by IEEE Student Branch IIIT Noida.

I have also attended various hackathons in the Delhi NCR, like HACK CBS, DSC Hackathon, Hack With Tony, etc. and gained experience on how to make real-world projects & I am also active in various conferences around Delhi NCR like I have attended PyDelhi meetup 2018, LinuxChix Meetup 2018, various tech talks conducted in our college.

I have also completed a course on Python “Automate the Boring Stuff with Python: Practical Programming for Total Beginners” - By Al Sweigart.

I also do competitive programming sometimes & play capture the flag events.

I am an active contributor at Oppia, I am a part of Dev Workflow Team lead by Apurv Bajaj, Overall Learner Experience lead by Akshay Anand, Bug Fixing Team, Oppia Onboarding Team, and also an Oppia Team Member. I have been contributed to Oppia since November 2018 so I am very much familiar with the codebase both frontend and backend.

Link to important PRs:

I have been contributing to Oppia since November 2018, and till now I have my 18 PRs merged, and 4 are open.

1. One off job, to populate message count in GeneralFeedbackThreadModel [#5999](#)
2. Added story viewer backend handler [#6237](#)
3. Added subtopic page data handler [#6327](#)
4. Updating Feedback threads in real time [#6183](#)
5. Added a presubmit check for JS files and refactoring the files required [#6125](#)

I have fixed many bugs related to frontend and backend, and for a full list of PR click [here](#).

Project Plan:

Overview:

This project aims to introduce a new feature of asking the students why they landed on a particular answer, such that students can explain what lead to this situation. This will help students to encourage reflection on their part, as well as this will also help in providing the anonymized information to the creators about student misconceptions, so that creator can improve the Oppia's feedback for future students. Through this creators can enhance their explorations basically the questions quality such that creators can vary the difficulty of the questions at different stages of the explorations through the statistical information received & this will improve the Learner's Experience too.

Why do we need this feature?

This feature will help in making the explorations more robust, such that creators can learn from the student misconceptions and get an in-depth knowledge of their published explorations, which will help in editing the explorations and creating more learner-friendly explorations in future which will lead to improved feedback of Oppia in future. From the student's side, this will benefit them to recognize their thought process that leads to the submission of the answer. This will create a class like environment for Oppia, as in classes the teacher gets to know the approach a student is applying while solving any question, in the same way this response that will be the approach described by the learner will help the creator to understand what actually the students are applying and thus the creator can improve the exploration. Thus resulting in a good feedback for Oppia.

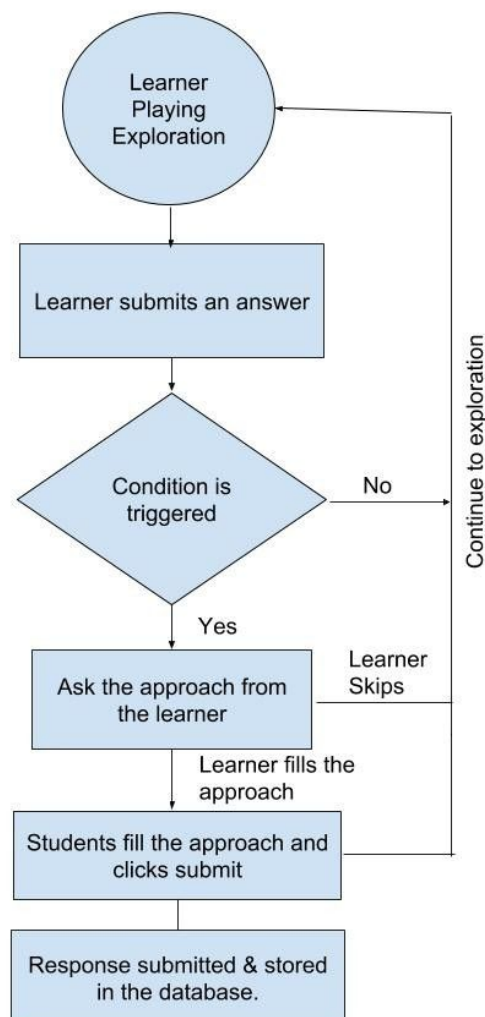
Explorations which will get this feature:

The explorations which have their correctness feedback enabled will get this feature, as with such explorations we can count the correct & wrong submissions made by the learner, whereas for the exploration in which correctness feedback is disabled the wrong submission can't be judged as there is no parameter mention such that the outcomes can be judged whether it is correct or not. While in the case of correctness feedback enabled and if the creator has marked an outcome as correct then that outcome can be judged as a correct response.

Technical Outline:

Learner's Aspect:

Learner Control Flow for submitting the approach of an answer submission.



Condition which will trigger this flow:

Approach:

The approach which will trigger only when the creator has enabled ask learners for response option, so after then if a learner submits an answer which is not marked as correct, then the count of the wrong submission for the learner will get increase and this count will be compared to the threshold value of that state and if it exceeds then the response form will get open to ask the approach learner has applied. For this, a threshold value (type: float) will be linked to every state of the exploration, prior to that the correctness feedback is enabled for exploration and this threshold value will be trained such that it always attain its optimum value.

About wrong submit threshold:

This variable is being used for every state if the creator has enabled correctness feedback option for that exploration, and it will be initialized to a value whenever an interaction is created in an exploration.

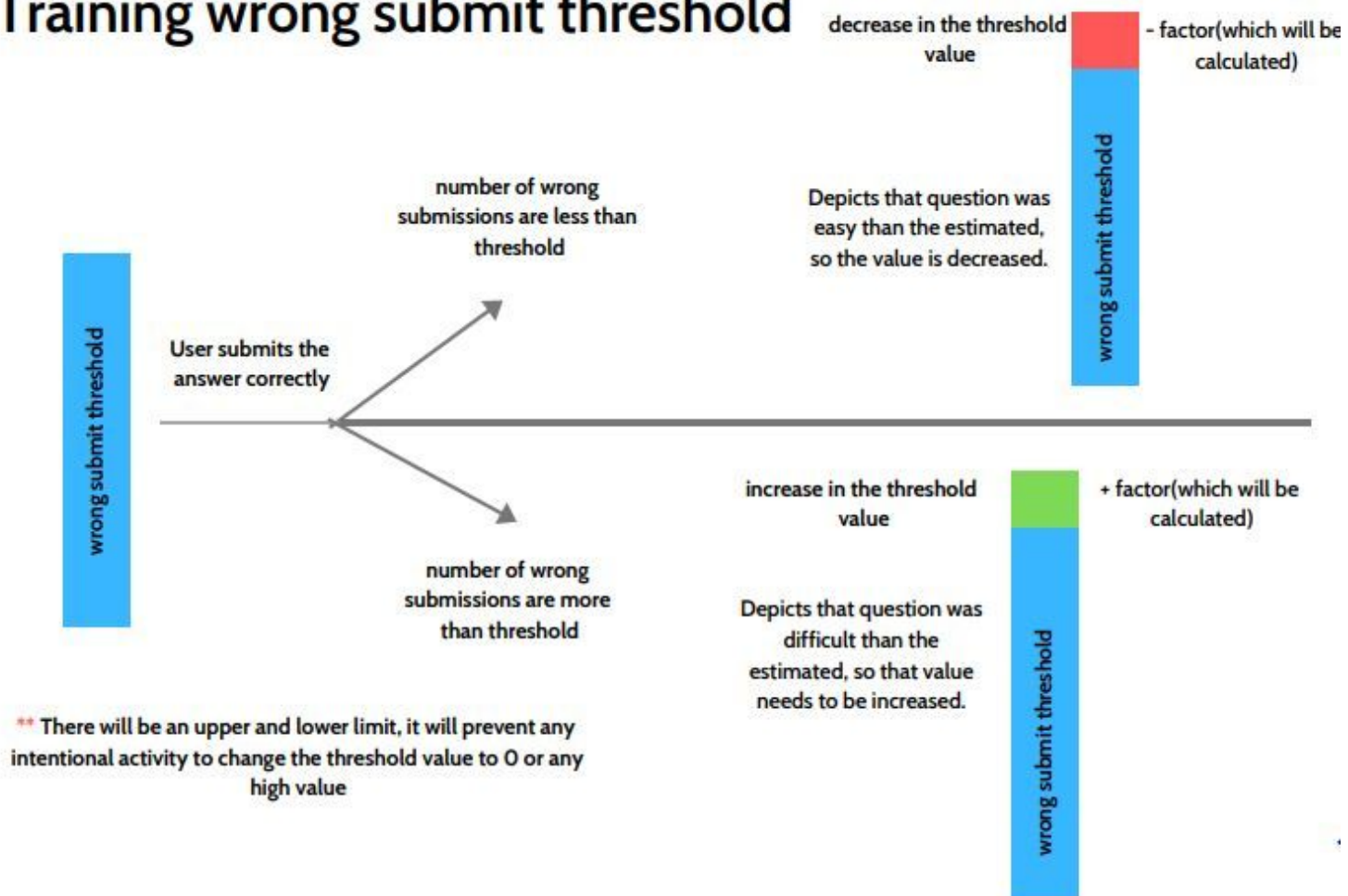
Let's suppose we create interaction in exploration then wrong submit threshold will get linked to that state, therefore every state will have its own wrong submit threshold value. And whenever the learner submits the question present in the state wrongly more than the wrong submit threshold for a particular session, he will be asked for the response to explain the approach. This threshold value will get changed by a factor, such that factor will decrease its value if any learner corrects the answer in less than the threshold value & correspondingly increase if the learner gives the correct answer in more than the threshold value, the wrong submit threshold value will get an increase. This will help in maintaining the optimum value of the threshold i.e if the question will be easy then factor will decrease the threshold indicating that it is an easy question (as many learners will submit the question correctly in attempts less than threshold) and if the factor increases the threshold i.e. the question is tough(as learners are failing to solve that problem in the given threshold attempts so its value needs to be increased).

So in this way, a wrong submit threshold variable will be added to the state, such that if a learner submits a question in more than threshold attempt for a question he will be asked for a response. And there will be a fixed range for wrong submit threshold (eg. [3, 10]) such that its value stays between the range only. This will prevent the situation like suppose 5 learners submit the question correctly in the 1st attempt only & the threshold value for that question will get reduced to 1 (when we have not defined any range). So if any learner comes who solves the question, not in 1st attempt will be asked for the question, which can lead to bad UX. Hence there

will be a fixed range for wrong submit threshold for better UX. Wrong submission count will be maintained in the frontend to check that at which time we have to ask for a response.

The above approach is an implementation of training value by providing the real-time data, such that in this case, the value which is being trained is threshold value and the real-time data which is being fed is the number of wrong submissions.

Training wrong submit threshold

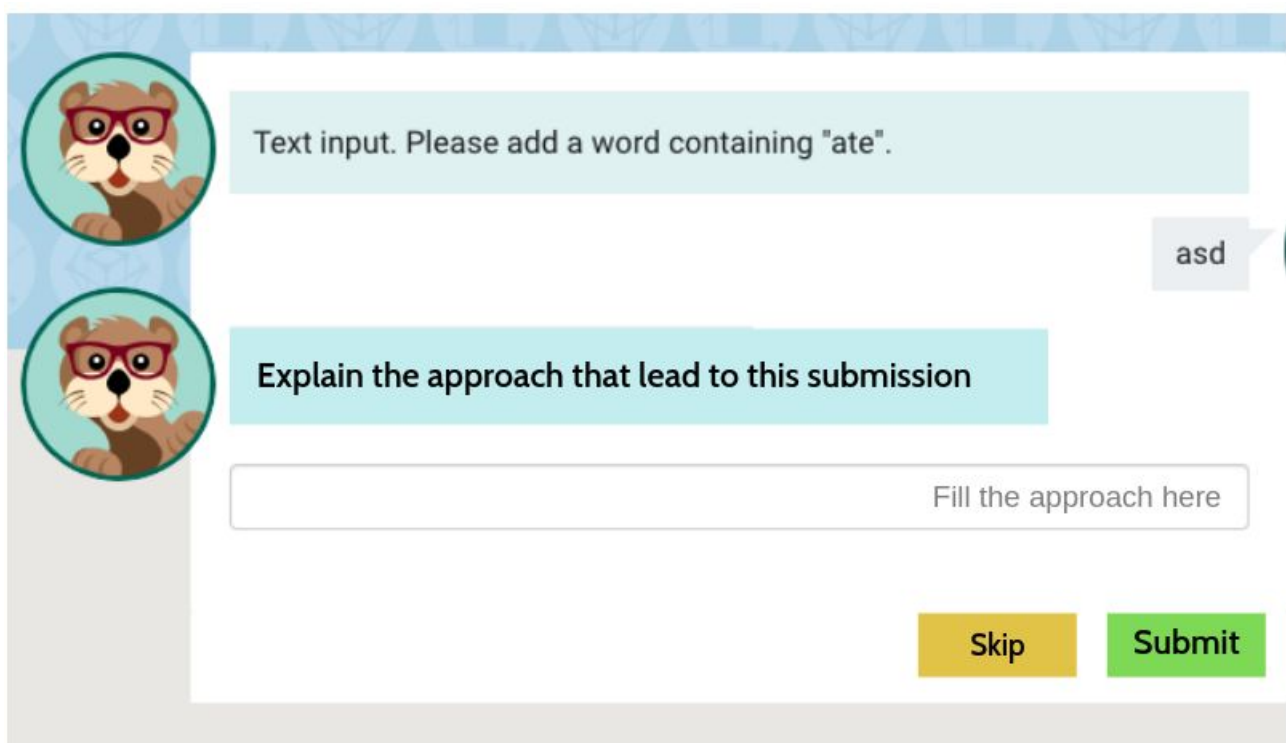


Overview of learners view:

When the learner will submit any answer which is not marked as correct by the creator, and prior to that there must be any answer group which is marked as correct then that submission will be counted as a wrong submission and through various mathematical calculations (as mentioned above) if the condition is triggered a response form will get opened in the exploration player such that the learner will be asked to fill their approach such that they landed on this answer (the learner will be unaware that whether he has marked the correct or wrong answer, it will help to get genuine response from the learner). Response Form will include the textbox, such that the learner can write about its approach that leads to the wrong submission. And then submit the form which will save the response and a message for 'Thank You for the response' will be opened which will thank the learner and the learner can then continue back to the exploration. There will be an option on the response form 'Skip', it will help the learner to bypass the response form if they don't want to fill it or if they have just guessed this answer.

(Please Note that these are just the very basic preliminary designs which will be improved upon with the help of community feedback)

Form where learner will submit his approach:



Text input. Please add a word containing "ate".

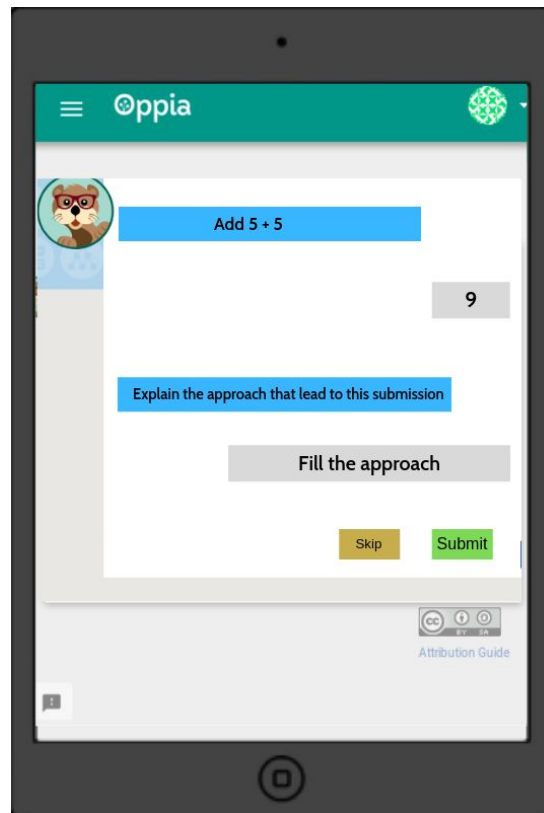
asd

Explain the approach that lead to this submission

Fill the approach here

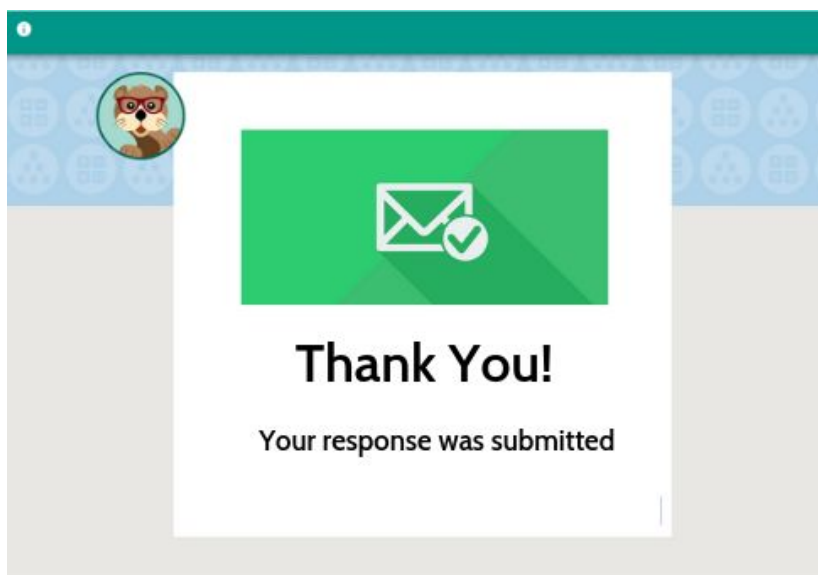
Skip Submit

The skip option will bypass the response form such that the learner can continue to the exploration, if he does not want to fill the form or he has just guessed the answer.



View of the learner in the mobile

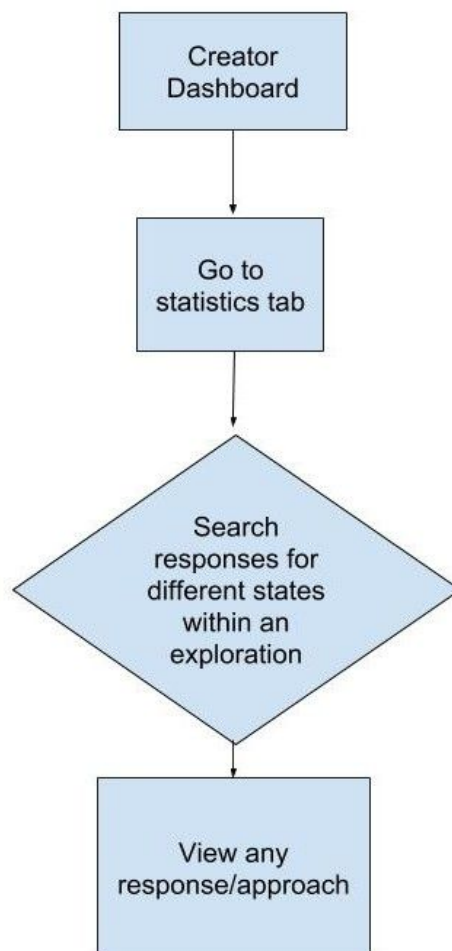
UI for thanking learner for submitting the response:



Creator Aspect :

The creator can view the responses received for any exploration by going to its statistics tab in the Creator Dashboard nav bar. In the statistics tab, by default view will be the exploration stats view and a switch option will be provided which will help the creator to change the view to the response statistics. In the response stats, there will be a table provided such that the creator can view the responses in the table state wise i.e. by selecting the state in the exploration. The table which will contain responses with explanation will have attributes such as no./id of the response, few starting words of the explanation, the number of words, time on which the response was received/created and note message i.e the note message the creator has added to the response. The creator will be able to delete any response if he feels is trivial. He will be able to delete all the responses and delete a particular response by opening and deleting it.

Control Flow for Creator:



The creator can view the responses by going to the statistics tab in the nav bar. There will be a table consisting of responses such that the creator can view the responses of the exploration for different states. Showing responses state wise will help the creator to know exactly where the student has provided the response while playing the exploration.

The option from where the creator can enable to ask responses from learners when played the exploration:

The option can be enabled from the Settings tab on the Creator Dashboard, this feature will be available in the Advanced Features options. The creator can also disable just by making it off. This will help the creator to manage this feature, such that if in future he doesn't want to get responses to the approaches made by the students.

Advanced Features

Parameters



Parameters are values that change as the learner moves between cards ([more info](#)).

Automatic Text-to-speech



Automatic text-to-speech generates audio from your content for learners to listen to. It is recommended that you disable this feature if creating an exploration whose content consists of multiple languages.

Correctness Feedback



Correctness feedback allows the user to categorise answer groups as correct or incorrect.

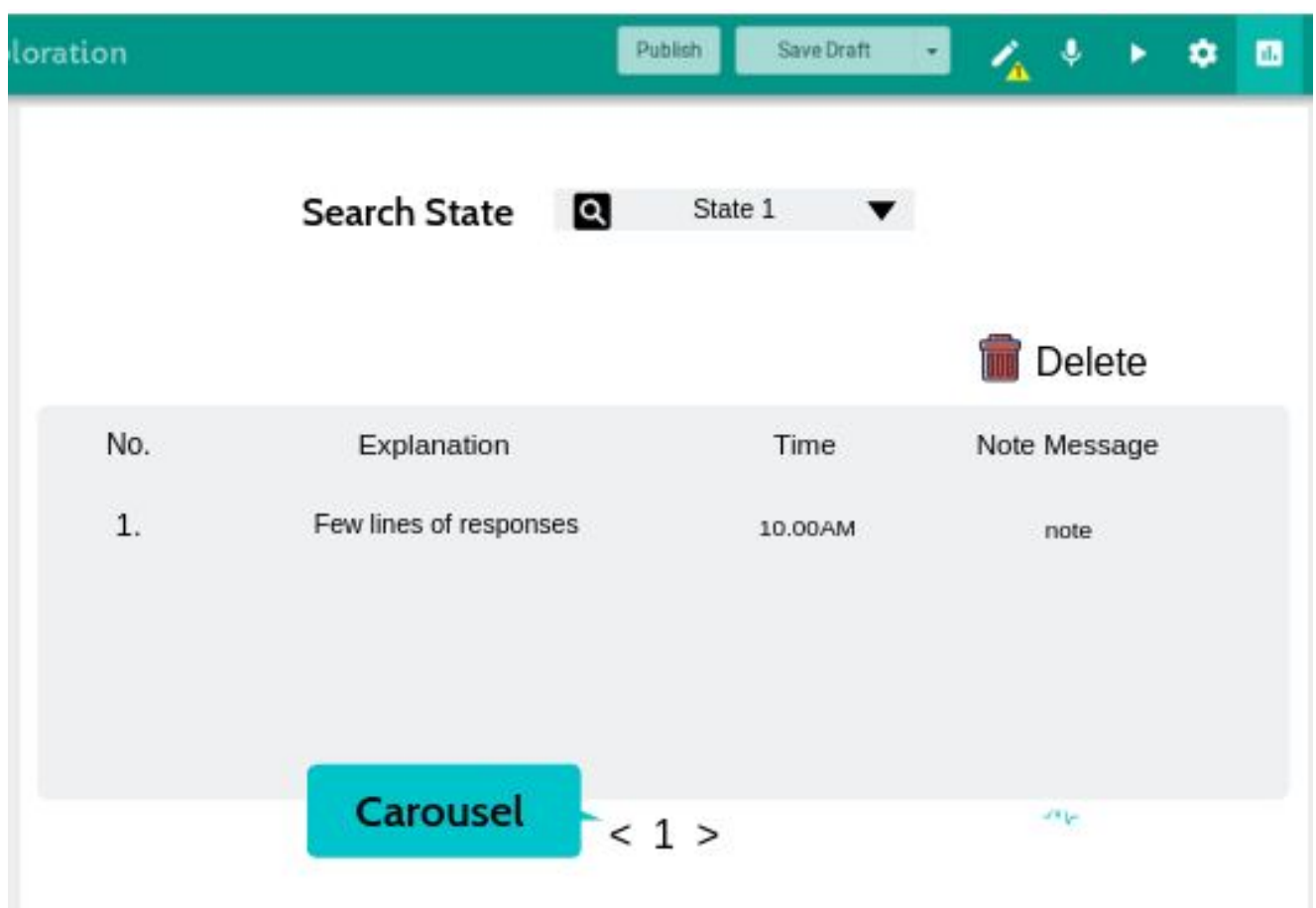
Ask learners for response

It allows to ask for response from learners whenever they submit any answer other than the marked as correct. Can only be enabled if the correctness feedback is enabled.



The UI of statistics of the response:

The UI will be displayed in the Statistics Tab, through a switch button. There will be a table which will show the responses statewise in any exploration such that the creator can select the state for which he wants to see the exploration. A feature of adding a note message will be there such that the creator can add the note for him, Delete button will be provided such that creator can delete the all the responses. Creator can also delete a particular response by opening and clicking on delete key in the modal window.



On a single page only 10 responses will be shown and the next one can be shown by moving to the next page(i.e the carousel feature). On clicking on each response a modal window will be opened which will contain the complete response.

UI for viewing any response:

Approach taken by the learner

Approach

I think XYZ is the best method to solve this question, so I applied this method and therefore answered this one.

Add a note

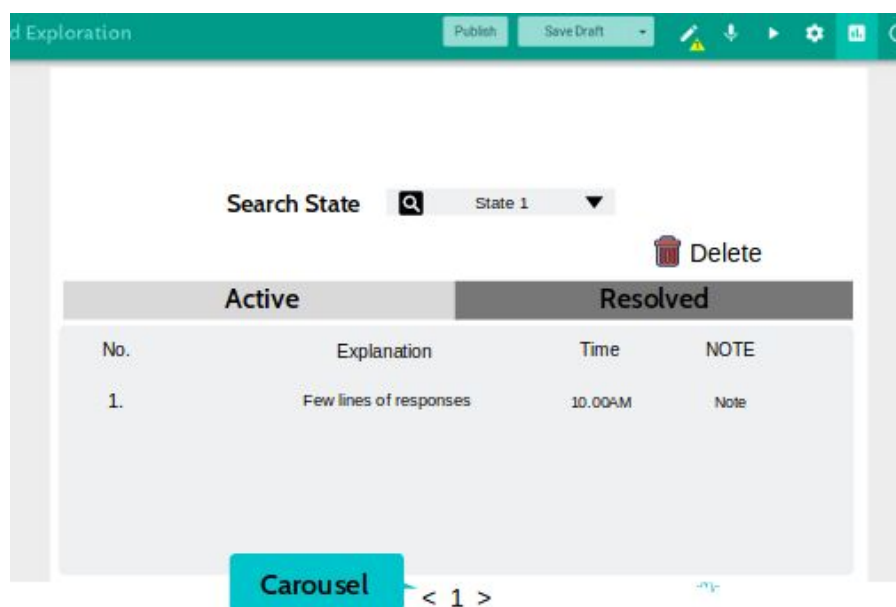
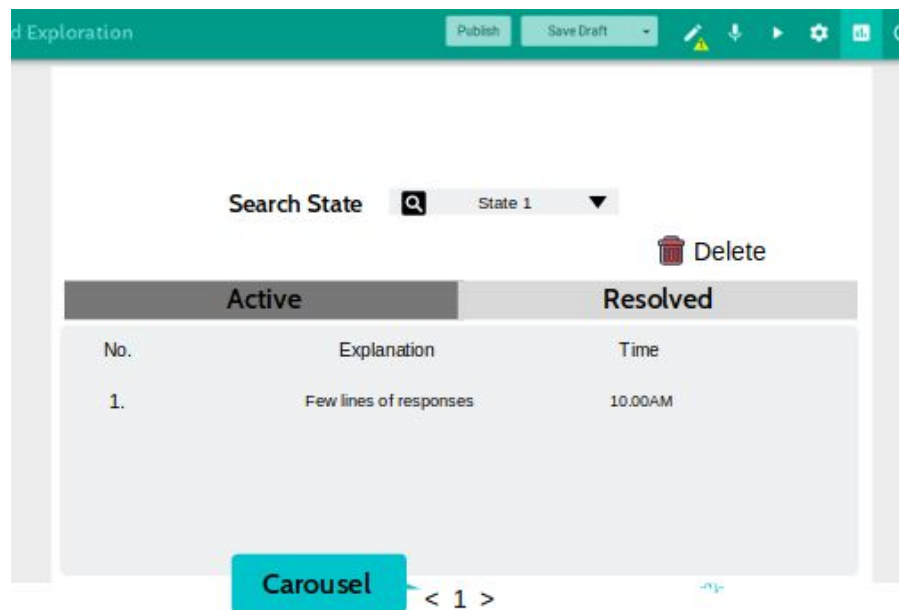
Cancel

Delete

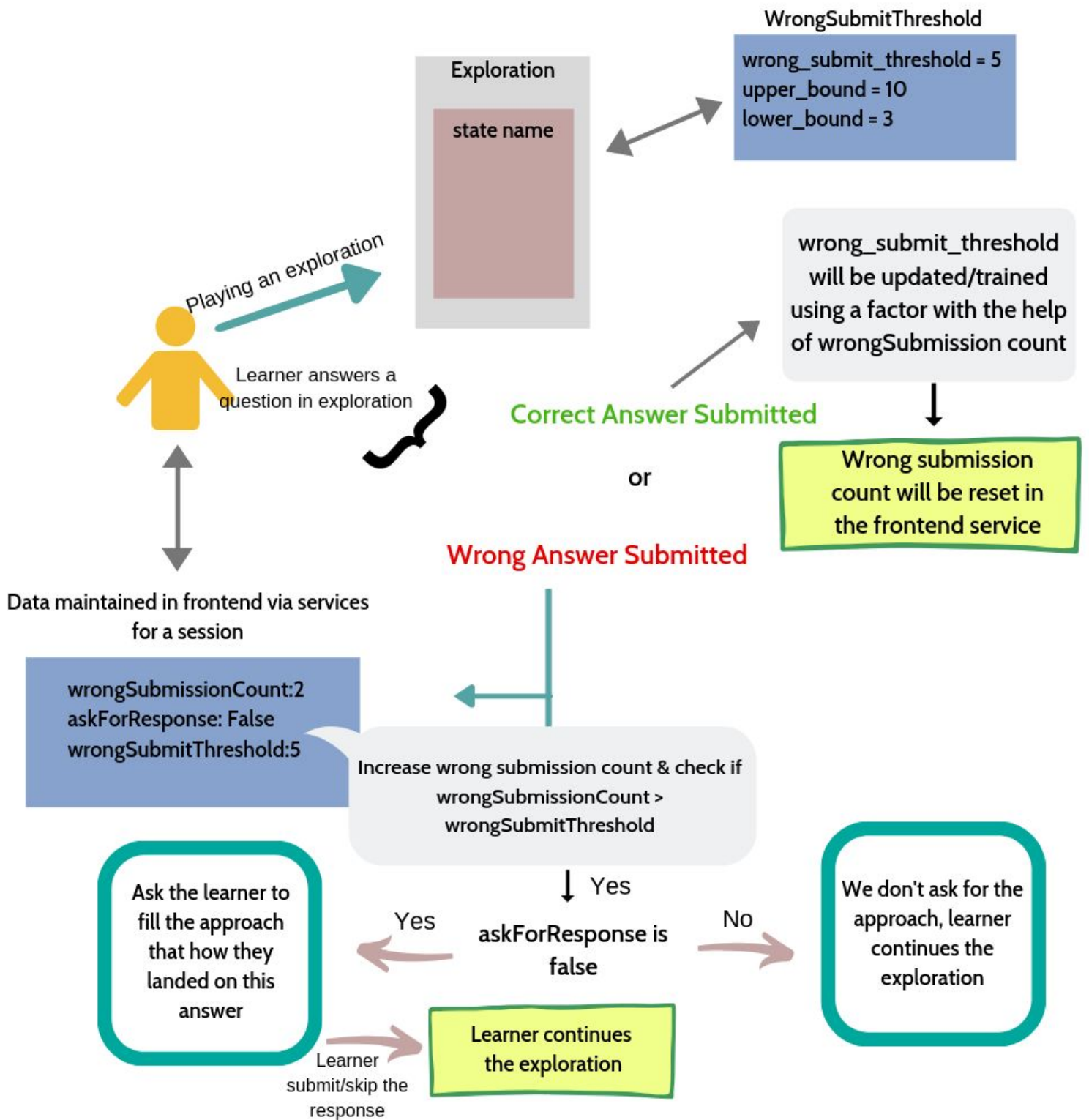
Save
Note

The approach will be viewed in the textbox, and there will be add a note option such that the creator can add a note to the response and save it. This will the creator if he opens the response in future, then he can see the note added to it.

The UI will contain two tabs one for active responses which will contain the unresolved responses and the other will be resolved tab, which will contain the resolved responses. Such that different directives will get loaded when different tabs are switched.



Diagrammatic overview of the project:



Milestones:

Milestone 1:

Implement the backend data storage models, domain objects, services and controllers needed for this feature.

1. Add a bool variable **ask_learners_for_response** to ExplorationModel in storage/exploration/gae_models.py which would ensure that if the creator wants to ask for the response from the learners when they play the exploration.

```
ask_learners_for_response = ndb.BooleanProperty(default=False,  
indexed=True)
```

- Edit the class Exploration in **exp_domain.py**, add **ask_learners_for_response** in the init method, edit the corresponding class methods. Also add a class method to **update_ask_learners_for_response(self, ask_learners_for_response)**.
- Add a checkbox in settings_tab.html with ng-change directive, such that **ng-change="saveAskLearnersForResponse()"**, this checkbox will remain disabled till the correctness feedback is enabled & correspondingly edit SettingsTab.js & add **\$scope.saveAskLearnersForResponse** function. And a service will be created **ExporationAskLearnersForResponseService.js**. And a proper note will be addressed below this checkbox, about how to enable it.
- Modify functions in exp_services.py, like **_save_exploration**, **_create_exploration** and many more functions.
- Test files **exp_domain_test.py** & **exp_services_test.py** will be modified accordingly to test the modified functions.
- One-off Job script will be written to populate **ask_learners_for_response** in the ExplorationModel which are already created in the database.
One-off Job: **PopulateAskLearnersForResponseOneOffJob**

Such that after the job is run in the production we can ensure that every ExplorationModel has a variable ask_learners_for_response which is initially set to False.

2. Now the task is to implement the **wrong_submit_threshold** value(which can be initialized to a suitable float value) & its corresponding range.

A class will be created in the state_domain.py, which will be responsible for storing the threshold value. And the range for the threshold will be saved in constant.js, **THRESHOLD_LOWER_BOUND** & **THRESHOLD_UPPER_BOUND**.

It will help to change the range easily for all the threshold models rather than migrating the models and then changing it.

```
class WrongSubmitThreshold(object):
    """Object for wrong submit threshold value & its corresponding range"""
    def __init__(self, exp_id, state_id):
        self.exp_id = exp_id
        self.state_name = state_name
        self.wrong_submit_threshold = 5.00; #Or any most suitable float value

    def update_wrong_submit_threshold(self, number_of_wrong_submission):
        """Update/Train the wrong_submit_threshold so as to ask for responses
        at an appropriate situation."""
        if number_of_wrong_submission != self.wrong_submit_threshold:
            exploration = exp_services.get_exploration_by_id(exp_id)
            number_of_states = len(exploration.states_dict)
            if number_of_states != 0:
                #Or any other factor which is more suitable.
                factor = (1/(number_of_states)) * (number_of_wrong_submission - self.wrong_submit_threshold)
                if (constants.THRESHOLD_LOWER_BOUND < self.wrong_submit_threshold+factor < constants.THRESHOLD_UPPER_BOUND:
                    self.wrong_submit_threshold+=factor
```

- After adding the class the corresponding tests will be added in state_domain_test.py
- update_wrong_submit_threshold function will be responsible to update wrong_submit_threshold value with the help of factor which will be calculated before updating.
- create_default_threshold will create default object of WrongSubmitThreshold for a state in the exploration.

- `get_wrong_submit_threshold` function will be created to get the threshold value of the state in an exploration, in the `state_services.py` & its corresponding test in `state_services_test.py`.
 - `delete_wrong_submit_threshold` function will be created to delete the threshold model if a state in an exploration is deleted.
 - `reset_wrong_submit_threshold`: This function will reset the value of the threshold.
3. After the implementation of the above parts, now the task will be to link **WrongSubmitThreshold object with the State of the exploration whose correctness feedback is enabled**, so whenever a state of an exploration is created, `WrongSubmitThreshold` with its predefined values i.e. range and the wrong submit threshold, `exp_id` and `state_name` will also be created. For all the explorations which don't have their correctness feedback enabled will have their `wrong_submit_threshold` to be `None`, as it will help to extend this functionality for those explorations if their correctness feedback is enabled after sometime or correctness measure is made more robust in future.

The init function of the state in the `state_domain` will be changed as follow:

```

    content_ids_to_audio_translations)
    self.written_translations = written_translations
    self.wrong_submit_threshold = WrongSubmitThreshold.create_default_threshold()

def validate(self, exp_param_specs_dict, allow_null_interaction):
    """Validates various properties of the State.
```

In this approach whenever the threshold will change, it will record as state has been edited/modified hence it will log commit history of that state hence resulting in the long commit history.

Another approach: To prevent the commit history not get always recorded, the threshold model should not be added as a class variable in `State`. Rather it should be treated as an independent class, which will be created if a new state is created. **So this approach will be good to go.**

In this approach a new model will be created in `explorations/gae_models.py` for implementing this approach, such that the model will be responsible to save the threshold models of the states in an exploration. And this model has been made generalized, such that it can be later use for the questions or if we implement anything new like questions in the future.

ThresholdModel:

- Entity_id: This will contain the id of the entity. For the state of an exploration the id will be 'exploration_id.state_name' and for the question the id will be the question id only.
- Entity_type: This will be of string property and it will be assigned to the constants that will be declared in the feconf.py file. Such that if the model is being used for the state then that constant for state will be assigned to the entity_type and for similarly for the question. Also this will help to generalise the model for the objects which will hold this model in future, just a constant will be added for that object in the feconf.py. Like currently these constants can be declared in the feconf.py file
 - THRESHOLD_ENTITY_STATE : 'exploration_state'
 - THRESHOLD_ENTITY_QUESTION : 'question'.
- Threshold_value: The threshold value.

```
class ThresholdModel(base_models.BaseModel):
    """Model for storing the threshold value of the state or
    question.
    """

    # The id of the entity.
    entity_id = ndb.StringProperty(required=True, default=None)
    # The type of the entity, that whether it is state, question, or
    # any other object if it is added in future.
    entity_type = ndb.StringProperty(required=True)
    # The threshold value.
    threshold_value = ndb.IntegerProperty(default=5)
```

A new class in state_domain.py (**WrongSubmitThreshold**) will be created, same as mentioned above in the code and services which will be implemented will be able to get in sync with the exploration such that if the state gets renamed or deleted or the whole exploration is deleted. This class will be linked to the exploration with exploration id and states with the state names, so that it will be easy to fetch the details of the threshold.

- change_threshold_model_state_name(exp_id, old_state_name, new_state_name): changes the name of the state in the model instance if the state name changes in the exploration

- `update_threshold(exp_id, state_name, wrong_submissions_count)`:
This function will update the threshold value of the state.
 - `delete_state_from_threshold(exp_id, state_name)`: remove the state name from state thresholds if the state is deleted.
 - `delete_threshold_model(exp_id)`: deletes the instance model if the exploration is deleted.
 - `get_threshold_value_by_state(exp_id, state_name)`: fetch the threshold value of a particular state of an exploration
 - `get_threshold_values_by_states(exp_id)`: fetch all the threshold values of the states in an exploration and return in the dict format with key as the state name and value as threshold value.
 - Now the migration of the state will be done according to the procedure as mentioned in the Oppia's wiki page : **Writing State Migrations** & then a validation job will be made to validate the exploration.
4. New folder will be made in storage for approach as storage/approach and then `gae_models.py` and its corresponding test. A new model will be created which will be responsible to storing the responses received.

The instance of the model will be created every time a new response is received. Every response will be stored in different instances.

StateApproachModel

And the model will be as:

```

class StateApproachModel(base_models.BaseModel):
    """Model for storing the approach/response user enters
    when he is asked for response after the wrong submission of a question
    in an exploration whose correctness feedback is enabled"""

    # The id of the exploration.
    exp_id = ndb.StringProperty(required=True, indexed=True)
    # The name of the state.
    state_name = ndb.StringProperty(required=True, indexed=True)
    # The id of the response.
    response_id = ndb.StringProperty(required=True, indexed=True)
    # The content of the approach
    approach = ndb.StringProperty(required=True, indexed=False)
    # The time at which the response was created.
    created_on = ndb.DateTimeProperty(indexed=True, required=True)
    # The note message entered by the creator.
    note_message = ndb.StringProperty(indexed=False, required=False)

    @classmethod
    def create(cls, exp_id, state_name):
        """Creates a new StateApproachModel entry.

        Args:
            exp_id: str. ID of the exploration.
            state_name: str. Name of the state .

        Returns:
            StateApproachModel. Instance of the new
            StateApproachModel entry.
        """
        response_id = cls._generate_id()
        return cls(id=instance_id)

```

5. Corresponding services/functions for the this model will be created in the gae_models.py using the model queries.
 - create_response(exp_id, state_name, response_content, time_received): Create the response with the given response content
 - get_responses_by_exploration_id(exp_id) : It will fetch all the responses linked to the exploration.
 - get_response_by_state(exp_id, state_name):It will fetch all the responses for a given state of an exploration.
 - delete_response_by_exploration_id(exp_id): It will delete all the responses of an exploration.

- `delete_response_by_state(exp_id, state_name)`: It will delete all the responses received for a state in an exploration.
- `save_note_message(exp_id, state_name, response_id, note_message)`: It will save the note message(that will be entered by the creator) for the response.
- All the above model query functions will be called in `response_services.py` and its corresponding tests will be added in `response_services_test.py`
- These functions will get in sync with **`_save_exploration`** function in `exp_services.py`, such if any change in change in exploration is there changes in this model also occurs. Like if the state is deleted then all the response will also get deleted linked to that state, and if the exploration is deleted then all the responses of that exploration needs to be deleted.

Timeline for Milestone 1:

Community Bonding Period 7 May - 26 May	Finish ongoing PRs, communicate with my mentor to make any required changes to the implementation plan, and complete responsibilities for May release.
---	--

My end semester exams will get over around 22 May, so I can start the work before 27th May.

PR Details	Date for first draft	Date to be merged by
Add functionalities for creator to enable ask learners for response option & one off job <code>PopulateAskLearnersForResponseOneOffJob</code>	29-30 May 2019	2-3 June 2019
Implementing the <code>WrongSubmitThreshold</code> object and its corresponding services in the backend	5-6 June 2019	9-10 June 2019
Linking the threshold model with the states and doing changes required in the backend <code>ExplorationMigrationValidationJob</code>	12-13 June 2019	15-16 June 2019

Add the StateApproachModel which will responsible for storing the responses	17-18 June 2019	20-21 June 2019
Implementing the domain and services for the StateApproachModel	21-22 June 2019	23-24 June 2019

Jobs to be run in production:

Name of the job	Request submitted for running on test server	Request submitted for running on production server	Release Targeted
PopulateAskLearnersForResponseOneOffJob	4 June 2019	8 June 2019	July
ExplorationMigrationValidationJob	17 June 2019	22 June 2019	July

In case any error is reported on the test server, then there might be a delay of 3 days in the date of the submission of the request to run the job on the production server.

Milestone 2:

Add the controllers for handling the submit response event and data handler for handling the data of responses that needs to be showed to the creator. Also augment the learner view UI to enable learners to fill out responses.

1. A file **response.py** will be created in core/controllers & corresponding response_test.py
 - New constants will be added in feconf.py
 - **RESPONSE_DATA_HANDLER**: 'response_data_handler'
 - **NEW_RESPONSE_URL**: 'responsehandler/create_new_response'
 - **THRESHOLD_DATA_HANDLER**: '/thresholddatahandler'
 - **UPDATE_THRESHOLD_HANDLER**: '/updatethresholdhandler'

- New routes will be created in main.py for both the viewing the responses and storing it in database.

```
get_redirect_route(
    r'%s/<exp_id>/<state_name>' % feconf.RESPONSE_DATA_HANDLER,
    response.ResponseDataHandler)
get_redirect_route(
    r'%s' % feconf.NEW_RESPONSE_URL,
    response.NewResponseHandler),
get_redirect_route(
    r'%s/<exp_id>/<state_name>' % feconf.THRESHOLD_DATA_HANDLER,
    response.ThresholdDataHandler)
get_redirect_route(
    r'%s/<exp_id>/<state_name>' % feconf.UPDATE_THRESHOLD_HANDLER,
    response.UpdateThresholdHandler)
```

- In response.py two classes will be created.
 - Class **NewResponseHandler**: It will create the new response through post request, exp_id, state_name, response will be fetched using payload, which will use the services of approach_services.py to store response made by the learner.
 - Class **UpdateThresholdHandler**: It will receive the post request with the exploration id, state name and the number of wrong submissions. Such that updation can be done in the threshold value of the state.
 - Class **ThresholdDataHandler**: It will manage the get request and send the wrong submit threshold value to the frontend. It will fetch the threshold value from the parameters exploration id and state name.
 - Class **ResponseDataHandler**: It will manage the responses to be displayed to creator for a particular state of an exploration. get(self, exp_id, state_name) will get the responses from the service get_responses_for_state. And will update the values and render it in json format.


```

class NewResponseHandler(base.BaseHandler):
    """Handles operations relating to creating response."""

    @acl_decorators.can_create_response
    def post(self, exploration_id, state_name):
        response_message = self.payload.get('response')
        if not response_message:
            raise self.InvalidInputException(
                'Response cannot be empty.')

        time_received = self.payload.get('time_received')

        if not time_received:
            raise self.InvalidInputException(
                'Enter a valid time')

        result = response_services.create_response(
            exploration_id, state_name, response_message, time_received)
        self.render_json({
            'result': result,
        })

```

```

class UpdateThresholdHandler(base.BaseHandler):
    """Handles operation for updating the threshold."""

    @acl_decorators.can_update_threshold
    def post(self, exploration_id, state_name):
        wrong_submissions_count = self.payload.get(
            'wrong_submissions_count')

        if not wrong_submissions_count:
            raise self.InvalidInputException(
                'Wrong Submission Count cannot be empty')

        wrong_submission_count = int(wrong_submission_count)
        if not isinstance(wrong_submission_count, int):
            raise self.InvalidInputException(
                'Wrong Submission Count should be integer')

        result = state_services.update_threshold(exploration_id,
            state_name, wrong_submission_count)
        self.render_json({
            'result': result,
        })

```

2. Now a HTML directive file will be created for the response form i.e. **response_form_directive.html** and its corresponding **ResponseFormDirective.js** will also be created. The response form directive will be added in the **tutor_card_directive.html** such that it will be viewable only if the value of `$scope.askForResponse` is true (**ng-if = “askForResponse && !responseSubmitted”**), by default the value of `$scope.askForResponse` will be false. When the learner will submit the answer, if the answer is correct the learner will move to the next state, but if the answer is not correct and the condition is triggered then before going to the next state a response form will get open below the answered question, such that learner will be asked to fill is approach and submit it. There will be an option of skip such that the learner can bypass it and move to the next corresponding state. The function `$scope.submitAnswer` in **ConversationSkinDirective.js** will be modified such that it loads the next state after the learner has submitted the response. Before going to the next state, and asking for response from the learner will help in getting the genuine response from the learner. In this way the response form will be shown to the learner.

response_form_directive.html(form in which the student will explain their approach)

```
<div class="modal-header">
  <h3>Explain your approach that lead to this submission</h3>
</div>

<div class="modal-body">
  <textbox ng-model="responseData"></textbox>
</div>

<div class="modal-footer">
  <button class="btn btn-default" ng-click="skipResponse()">Skip</button>
  <button class="btn btn-success" ng-click="submitResponse" ng-disabled="!responseData">
    Submit Response
  </button>
</div>
```

thanking_directive.html(This directive can be used for many things : Like when learner submits a feedback, submits a suggestion, rate the exploration etc.). An image of thank you will be linked using the image tag and this will be displayed if the learner has submitted the response. And correspondingly **ThankingDirective.js** which will be responsible for loading the page through **UrlInterpolationService.getDirectiveTemplateUrl(//location of the html file)**.

3. Checking the number of wrong submission via frontend service: **WrongSubmitService**. This service will be responsible for detecting the wrong submission, which will be detected in the frontend only. When the learner is playing an exploration and answering a answer, and if the outcome of that answer group is marked as correct it means that the answer is correct and then we have to perform the update in the threshold value of that state, or if he marks the answer group whose outcome is not marked as correct then, the outcomes of the all the answer group will be checked, if any of the outcome is labelled as correct it means that the learner has done a wrong submission, so the count of wrong submission will be increased and checked if that value is greater than the wrong submit threshold value of the state then ask for response. So when the learner will submit the response the variable **responseSubmitted** will be marked as true, such that no further response is asked from that learner. And if any of the outcome of answer groups is not marked as correct it means that there is no particular correct answer for that state so no wrong submission needs to be count in that state. Before counting the submissions, the detail of the exploration will be fetched from backend that whether the creator has enabled the option of ask learners for response. If the creator has enabled it, then only the following tasks will be done to trigger the response form to the learner.
- In the service there will be a function which will send get request to the **ThresholdDataHandler** with the params exploration id and state name. Such that we can get threshold value for a particular state and exploration.

```
$http.get('/thresholddatahandler' + explorationId + '/', + stateName, {
  exp_id: explorationId,
  state_name: stateName
}).then(function(response){
  wrongSubmitThreshold = response.data.wrong_submit_threshold;
}, function(){
  AlertService.addWarning('There was an error while fetching the threshold value').
});
```

- WrongSubmitService will be imported in ConservationSkinDirective.js. There is already **\$scope.answerIsCorrect** variable which tests that whether the answer marked has outcome labelled as correct. So if its get corrected, there will be request which will be send to backend which will update the threshold value of that state, the parameters which exploration

id, state name and number of wrong submissions will be passed to the controller.

- The wrong submission will get detected by the service, **resetWrongSubmissionCount** will reset the count if the learner has moved to another state i.e. old state name does not matches with the current state name. Increase wrong submit count will be there which will increase the **wrongSubmissionCount** and **getWrongSubmissionCount** function will be created which will return the wrong submission count.
 - There will be a `$scope.responseSubmitted` will be added in `ConservationSkinDirective.js`, which will ensure that the response is not asked again from the learner in that session again.
 - A function in the service will be added which will detect the wrong submission, **checkWrongSubmission(answer, answerGroups)**, it will be checked if the answer outcome was not marked as correct, hence all the answerGroup will be checked that whether **answerGroup.outcome** is labelled as correct or not.
 - When the learner will mark an answer correctly the wrongSubmissionCount will be send to update the threshold value of that state, the wrong submission count will be now set to 0.
4. The response form will be loaded when the **askForResponse** is marked true and **responseSubmitted** is marked false then the response form directive will be loaded, if the learner fills the approach and submit it **askForResponse** will be marked false and **responseSubmitted** will be marked true. Or if the learner clicks on 'skip' option then also the same as above will happen.
 5. When the learner will submit the response, the responseSubmitted will be marked as true for that session, and data will be sent through post request to the new response handler, such that the new response is stored in the database. After that a thanking directive will open, and will fade out in 1-2 seconds and then the learner can continue to play the exploration.
 6. Testing of the frontend services(adding the test files), built during Milestone 2 and overall testing of the learner side will be done.

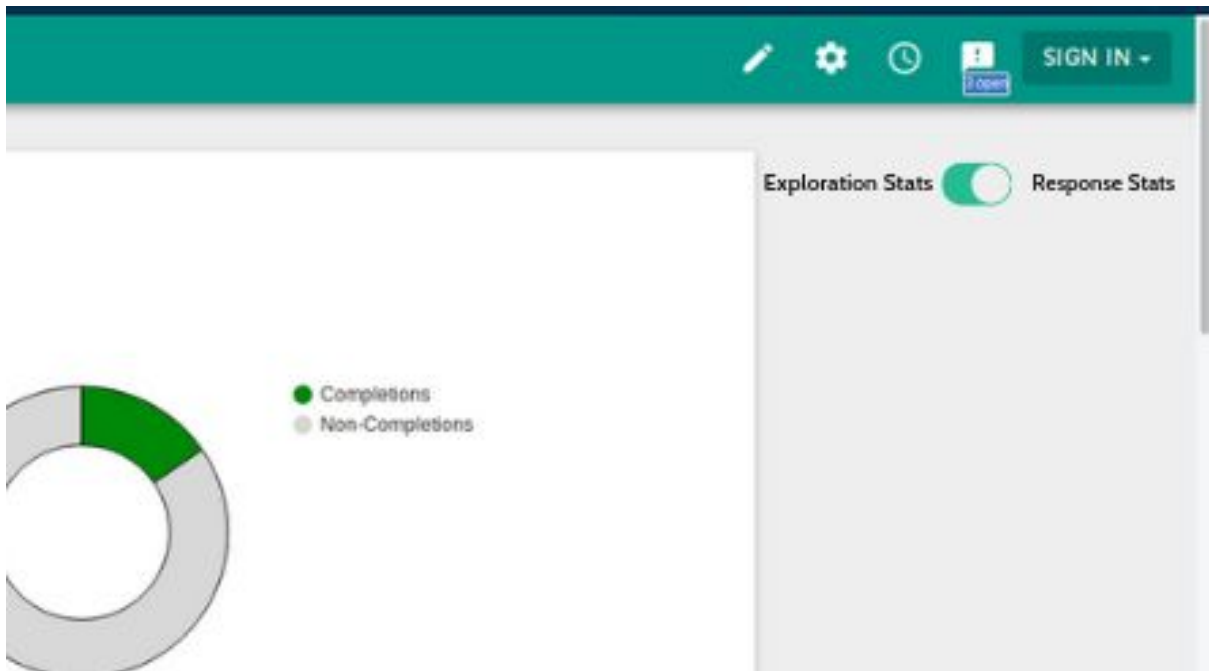
Timeline for Milestone 2:

PR Details	Date for first draft	Date to be merged by
Implementing the backend controllers for handling the task related to responses	1-2 July 2019	4-5 July 2019
Making the HTML files - response form, thanking form	7-8 July 2019	10-11 July 2019
Detection of wrong submission via implementing suitable frontend services & its corresponding tests	13-15 July 2019	16-17 July 2019
Linking all the aspects to the exploration player and testing whether the learner is able to fill the response form	18-19 July 2019	21-22 July 2019

Milestone 3:

Augment the editor view UI to surface these responses to creators, and complete the end-to-end feature so that the complete life-cycle functions correctly. Add e2e tests to verify the overall life-cycle.

1. The statistics page of the exploration will be edited, so as to show the responses received by the creator. A **switch option** will be added in Statistics page, such that creator can switch the button to show the learner submission approach stats and exploration stats.



Statistics_tab.html will get modified to introduce a switch. Such that for different directives will be loaded on different switch. **StatisticsTab.js** will be edited such that when **\$scope.explorationStats** is true then explorations stats directive is loaded and if **\$scope.responseStats** is true then response table html file is loaded.

2. A new directive **response_table_directive.html** will be made which will be responsible for showing the responses received in an exploration for a particular state. The creator will be able to select the states for which he wants to see the responses and then click on submit to show the responses of that state. The response stats will only be shown if the correctness feedback is enabled for an exploration otherwise it will show a message to view the response please enable the correctness feedback and allow ask learner for response to receive responses again. The responses will be displayed orderly by received time, the latest ones will come first.

On the page there will be limit, such that only 10 responses will be shown and there will be a **carousel feature** such that creator can click on the arrow button to move to the next page and view the other 10 responses and similarly next, this will help to increase creator UX.

The overview of the HTML file:

```

<table class="table">
  <tr ng-if="getResponse().length > 0">
    <th>No</th>
    <th>Approach</th>
    <th>Time</th>
    <th>Note Message</th>
  </tr>
  <tr ng-repeat="response in getResponses()|orderBy:'-recieved_time'" ng-click="onClickRow(response.responseId)">
    <td>
      <span ng-class="getId(response.id)"><[getHumanReadableId(response.id)]></span>
    </td>
    <td>
      <span><[response.approach | truncate:40]></span>
    </td>
    <td>
      <span><[getLocaleAbbreviatedDatetimeString(response.timeReceived)]></span>
    </td>
    <td>
      <span ng-show="response.noteMessage"><[response.noteMessage | truncate:40]></span>
      <span ng-show="!response.noteMessage">No note</span>
    </td>
  </tr>
</table>

```

3. **ResponseTableDirective.js** will be created, which will be responsible to display the data in the html directive. The responses will be fetched via **ResponseDataService.js**, which will be responsible for get request to fetch all the responses related to a state of an exploration.

```

var _fetchResponses = function(exp_id, state_name) {
  $http.get('/responsedatahandler/' + exp_id + '/' + state_name).then(function(response) {
    var allResponses = response.data.responses;
  });
};

```

DateTimeFormatService will be used for to display the time the response was received.

\$scope.getLocaleAbbreviatedDatetimeString=
(DateTimeFormatService.getLocaleAbbreviatedDatetimeString

4. Controllers needs to be implemented for deletion of response & marking the response as important. Controllers will be made such as **ResponseDeleteHandler** : It will handle post request with parameters exp_id, state_name and the response ids list which needs to be deleted will be passed as parameter which can be fetched through payload.

ModifyResponseHandler: It will help in modifying the response i.e. adding a note message to the response(It will be a general controller used for modifying the response, as per now it's only functionality is to add note message to the response, but if the features of the responses are increased in future then we can perform the modification with this handler only). A constant `ADD_MESSAGE_TO_RESPONSE` will be send as parameter along with response id, state name, exp id and the note_message, hence in the controller if-condition will be checked for the constant.

Feconf.py will contain a constant : **DELETE_RESPONSE_HANDLER** :
'/responsedeletehandler'

MODIFY_RESPONSE_HANDLER: '/modifyresponsehandler'

And the new routes for both the handlers will be added in the main.py

5. A new modal for showing the response will be created such that it will include three buttons that will be cancel(to close the modal), delete(to delete the response), save note(providing note to the response). A small text box will be provided for providing any note to the response.

editor_view_response_modal_directive.html

```
<div class="modal-header">
  <h3>Approach taken by the learner</h3>
</div>

<div class="modal-body">
  <section>
    <div>
      <strong>Approach:</strong>
      <textbox ng-model="responseContent">
      </textbox>
    </div>
    <div>
      <span>Add a note</span>
      <textbox ng-model="response.noteMessage"></textbox>
    </div>
  </section>
</div>

<div class="modal-footer">
  <button ng-click="closeWindow()">Cancel</button>
  <button ng-click="deleteResponse()">Delete</button>
  <button ng-click="addNote()" ng-disable="newNoteMessage != oldNoteMessage">
    Save Note
  </button>
</div>
```


6. After clicking on the response (**ng-click = “onClickRow(response.id)”**) a modal will be opened such as the creator can view the response on which he has clicked. **ResponseServices.js** will be created which will handle the deletion part and other modification parts (like adding the note message) related to the responses.

For the deletion of responses, the explorationId, stateName, and the list of responselds will be sent to the controller such that responses gets deleted after that.

deleteResponse(expld, stateName, responseld) : which deletes a particular response.

deleteMultipleResponses(expld, stateName, responselds): will delete multiple responses.

addNoteMessage(expld, stateName, responseld, newNoteMessage): will add the note message to the response.

7. Testing of the frontend services(adding the test files), built during Milestone 3.
8. Creating e2e tests & overall testing the full cycle of submitting the response, till viewing and deleting it.

Timeline for Milestone 3:

PR Details	Date for first draft	Date to be merged by
Modification for statistics page & making UI for the creator view.	25-26 July 2019	29-30 July 2019
Making the necessary frontend services for creator side eg. for displaying the responses data, deleting and viewing any response in particular	3-4 August 2019	6-7 August 2019
Making test files for the frontend services added during Milestone 3	10-11 August 2019	13-14 August 2019
Adding e2e tests & testing the overall cycle of the functionality. If any error seems to come then fixing it.	18-19 August 2019	21-22 August 2019

Why I am using number of states in the formula for updating threshold:

In the formula, the number of states is inversely proportional to the factor, which means the exploration which contains more number of states will face less deviation in the threshold value and which has less number of states will face more deviation in the threshold.

As the explorations which contain less number of states will be completed by most of the learners, and the value of the factor will be big(due to less number of states) and the threshold will deviate much, but since most of the learners are completing that exploration it will get its optimum value.

In the case of the exploration with more number of states, the value of the factor will be less so the value of threshold will deviate less, as the learners who will complete the exploration (which contain more number of states) will be less. So the value of the thresholds in this exploration will deviate less and hence optimum value can be maintained.

Also, I have just made this formula, if any changes in this formula need to be done I am fine with it.

Documentation:

A Google Doc will be maintained throughout all the Milestones, such that it will include the documentation which needs to be added at each milestone. So at last the Google Doc will contain all the documentation that needs to be published. It will be submitted by 22 August 2019.

Alternatives Considered:

Note: These alternatives were also thought while making the proposal.

1. For viewing the response form, we can hide the current state i.e. conversation skin directive, whenever there is a need of submitting the response form. Such that ng-hide directive will be used to hide the conversation skin if the askForResponse is true, whereas the response form will use the ng-if directive, such that if askForResponse is true and responseSubmitted is false, then the response form will be shown.
2. The other thing we can provide the response form is by using modal window.

Such that a modal window will get opened up, when the learner needs to submit the response/approach.

3. The view section of the statistics page can have the tab mode, for switching between ExplorationStats and ResponseStats.

Future Extensions of project (will be taken after completion of the GSoC period):

1. Extend the asking for approach model for the Questions in Oppia.
As we are now working on Questions, so asking for an approach can be extended to those Questions.
2. We can provide submission details to the learner, regarding the total correct submissions, total wrong submissions, the percentage of correct submissions, etc. Such that analysis of the learner submission details can be shown to the learner so that they can get the in-depth detail of their submissions. For this, the submission details (wrong or right) needs to be stored in the backend using gae_model.
3. Extend the access to set the threshold value by the creator and fix the range accordingly. The creator will be able to set the threshold and range according to the state difficulty level.

Time Zone where I will be able to commit to the project :

Indian Standard Time(IST) which is ahead of UTC by 5 hours and 30 minutes.

Time which I will be able to commit to the project :

I will be able to devote 8-10 hrs a day, and about 5-6 hrs during the weekend. i.e. approximately 55-60 hrs in a week. In August my classes will be started hence time spend during weekdays will be lesser i.e 5-6 hrs while during weekend 8-10 hrs.

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

I have no commitments during the summer. I have also no classes or lectures till the end of July, my classes will start in August so my number of hours might get less during the weekdays, but I will cover it during the weekend.

However two days might be needed for traveling one around 24 May and other around 20 July, which I will cover by extending my work hours.

Communication:

My contact information:

Email: anubhavsinha98@gmail.com

GitHub Handle: [anubhavsinha98](#)

I am also active on Gitter & Hangouts, by the way I am okay with any communication channel.

How often, and through which channel(s), do you plan on communicating with your mentor?

I will be comfortable with any mode, be it Gitter or Hangouts and I am willing to choose any mode used by the mentors.

I will also plan to maintain a daily record of my progress.

I will be in continuous touch with the mentors, we can have biweekly(twice a week or as proposed by the mentor) meetings to discuss the workflow. And I am okay with any platform used for meetings.