Review tests and other improvements to the questions framework

Name: Shiqi Wu

Email: wushiqi1998@gmail.com

GitHub Account: sophiewu6

University of California, Irvine, Computer Science major

Project Details

Why are you interested in working with Oppia?

I have been using online tutorials and courses since middle school. They have helped me a lot when my school doesn't offer certain classes, when I couldn't understand some materials in class, or when I want to take classes from foreign countries. However, I still found this new education channel needs some improvements. When I was searching for open source communities, Oppia's goal "provide high quality education to those who lack access to it" quickly got my attention because of my interest in online education. I feel like I can make contributions to Oppia with my programming skills and experience with online education tools.

As I got involved into Oppia, I found everyone in the community is very friendly and willing to help newcomers. In the past, I only had chances to work with students from the same school. This is my first chance to work with people from all over the world. It impressed me that people from different parts of the world, who didn't know each other, are working together for the same goal. I would like to be part of this community to develop something useful and make friends from all over the world.

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

Reviewing is a very important part of studying. When I attempt to memorize vocabulary, I have to review old words every day so that I won't forget about them. I also found that if we have multiple quizzes throughout the semester, we won't struggle during the final week as much because we have reviewed the class materials before the quizzes.

When I first looked into Oppia's website, I felt like something was missing because there was no review tests between lessons. Learners would forget about old materials if reviews don't happen in time. When I saw this project idea in Google Summer of Code idealist, I picked this project right away because I think the learning experience can be significantly improved after adding review tests.

Another purpose of tests is to help learners get to identify their weaknesses so they can improve upon that. By implementing skill mastery level, learners can get a clear view of which parts they need to put more efforts on. It helps learners to allocate time to improve the weakness more efficiently.

Prior Experience

I have experience contributing to a large shared repository, Cloudberry, which is from a research team at University of California, Irvine. It intends to develop a general-purpose middleware system to support visualization on large amounts of data, as well as a TwitterMap prototype which supports interactive analytics and visualization on more than one billion tweets with new data continuously being ingested.

As for skills that are required in Oppia, I am proficient in both Python and AngularJS. I have three year's experience coding in Python. I have developed lots of projects, including a gaming AI and a search engine, with Python. AngularJS is the language for the frontend part of my research team. I have implemented a few large features for the team by using AngularJS.

Other than that, I have participated in a few Hackathons and Medical AI Jam, where I have developed Android apps and web apps in teams.

Links to My PR for Oppia

I will keep contributing to Oppia after submitting this proposal. Please refer to this link to get an updated version: My Pull Requests

#6566: Convert the current per-exploration translator role to a voice-artist role

#6557: Write e2e tests for editing exploration properties (coreEditorAndPlayerFeatures.js)

#6540: Make test coverage of core.storage.exploration.gae_models 100%

#6472: Remove globals ALLOWED_INTERACTIONS_CATEGORIES from creator_dashboard, exploration_editor, skill_editor, and topic_editor

Overview

This project aims to add review tests and improve the current question framework. After this project, review tests will be shown after going through a few lessons. The tests results will be used to update skill mastery level, so that learners can review and practice according to the mastery level and suggestions. Tests results will also be sent to creators once a few months to help improve the questions and lessons.

Display Review Tests

A review test player will show up after learner going through every three chapters in a story. The test player will make use of the existing question player. There will be three questions for each skill linked to the three chapters. Questions should be randomly selected from the question bank linked to each skill.

Learners can try the same questions infinite times, but every time they get the wrong answer, scores will be deducted (details see Mastery Level). Learners can't proceed to the next question until they get the right answer. Learners have the option to see hints and solutions like practice sessions when getting stuck. However, seeing solutions can result in deducting scores and mastery levels.

To avoid having too many questions in a review test, less questions will be selected for every skill when there are too many skills. When number_of_skills >= 7, we select two questions for each. When number_of_skills >= 10, we select one question for each. There will be an upper limit of 15 questions in the review test. If there are more than 15 skills linked to the three chapters, only select questions linked skills with relatively low mastery level.

Review Tests Results

Learners can attempt questions multiple times until they get the right answer. However, every time they get the answer wrong on the first time, the score of the review test will be deducted by 100 / number_of_questions. The score is set to 100 in default. For example, if a review test has 10 questions, answering 8 questions correctly on the first time means the score is 80.

The results can be used to decide whether the learner can proceed to the next chapter. The requirement for passing the test is answering at least 1/2 questions of each skill correctly at the first time. This percentage will be the same even if there are only one or two questions. That means we allow one wrong answer for every skill when we have two or three questions, but learners must get every question correctly at the first time when there is only one question.

If the learner passed every skill being tested, he/she can proceed to the next chapter. If the learner failed one or more of the skills, he/she has to review the concept cards and take the review test, which only contains the failed skills. Only after passing the review test, he/she can finally proceed studying.

Mastery Level
- Mastery levels are bound to each skill and learner, and they are set to 0.0 when nothing is done.
- Mastery level can be increased or decreased by demonstrating the corresponding skill, mostly by answering questions.
    - If a question is solved correctly the first time, increase the mastery level by 10% + 10% * question_difficulty (a float between 0 and 1). Difficulty 0.1: 11%, difficulty 0.2: 12%, etc.

- - If a question is solved correctly the second time, increase the mastery level by 10% * question_difficulty (a float between 0 and 1). Difficulty 0.1: 1%, difficulty 0.2: 2%, etc.
    - If a question is solved correctly after the second time, decrease the mastery level by 5% * (times_of_trying - 2). The third time: -5%, the fourth time: -10%. It won't be deducted any more than 10%.
    - Since the mastery level is between 0.0 and 1.0, it won't increase after reaching 100%, or decrease after reaching 0%.
  - Viewing hints or solutions can decrease the mastery level.
    - Mastery level will be decreased by 2% every time viewing a hint.
    - Mastery level will be decreased by 10% if viewing the solution.
  - Learners will be able to view their mastery levels in two ways.
    - Score modals that show up every time after finishing tests.
    - A page that has a list of skills and their corresponding mastery levels inside the topic viewer.

Send Feedback to Creators

The review tests results will be automatically sent back to the creator. In this way, creators can get the idea how the learners are doing in the review test, and how well they can handle the skill after going through the stories. Feedbacks will be designed to be sent every month (can be changed by the creators), so the creators won't be overwhelmed by too many feedbacks.

Users will have the option to set how often they want to receive feedbacks (ten days, one month, three months, six months, etc).

Format of feedbacks:

Writing a Fraction (skill name):

- Average mastery level: B
- Mode: A
- Questions list:
- Question 1: Total attempts: 5; Correction rate: 60%.
- ......

From these feedbacks, creators can overview which skill they did not explain well enough that learners can't understand thoroughly. Creators can improve their teaching materials accordingly.

Give Suggestions Using Mastery Level

Review Tests and Practice Sessions

Different suggestions will be given under different situations:

1. When mastery level is above 80% and it is increased: "You have mastered this skill very well! You can work on other skills or learn new skills."
2. When mastery level is under 80% and it is increased: "You have made progress! You can increase your mastery level by starting practice sessions."
3. When mastery level is above 80% and it is decreased: "Seems like you didn't do very well this time. Please keep practicing."
4. When mastery level is under 80% and it is decreased: "Please practice more on this skill by starting practice sessions."
5. When mastery level is at 100%: "Congratulations! You have mastered this skill!"

Pretests

1. When every prerequisite skill's mastery level is above 80%: "Congratulations! You have mastered every skill that is required for this new lesson."
2. When one or more of prerequisite skills' mastery levels are under 80%: "Please practice more on skill1 and skill2 (specify the names). You need to take the pretest before starting this lesson."

Also, pretests will be automatically skipped when every prerequisite skill's mastery level is above 80% because too many tests will be overwhelming for learners. Otherwise, learners have to take the pretests containing questions from prerequisite skills with mastery level lower than 80%.

## Select Questions based on Difficulty Level

Questions were randomly selected before implementing mastery level feature. We can improve it by selecting questions using both question difficulty and learner's mastery level. Questions with the most similar difficulty to the mastery level will be selected. For example, when there are five questions with difficulty 0.1, 0.3, 0.5, 0.7, 0.9, and a learner's mastery level is 0.65, then questions with 0.5, 0.7, 0.9 difficulty will be selected.
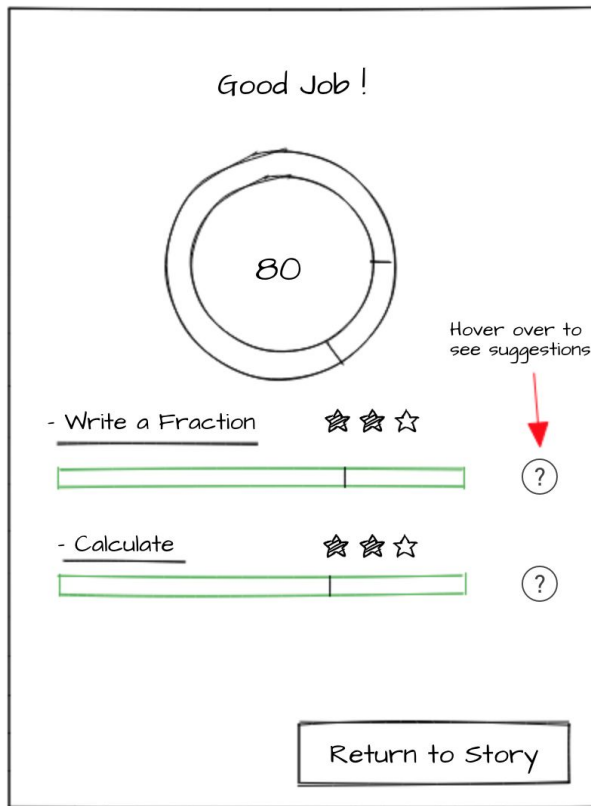
User Workflow
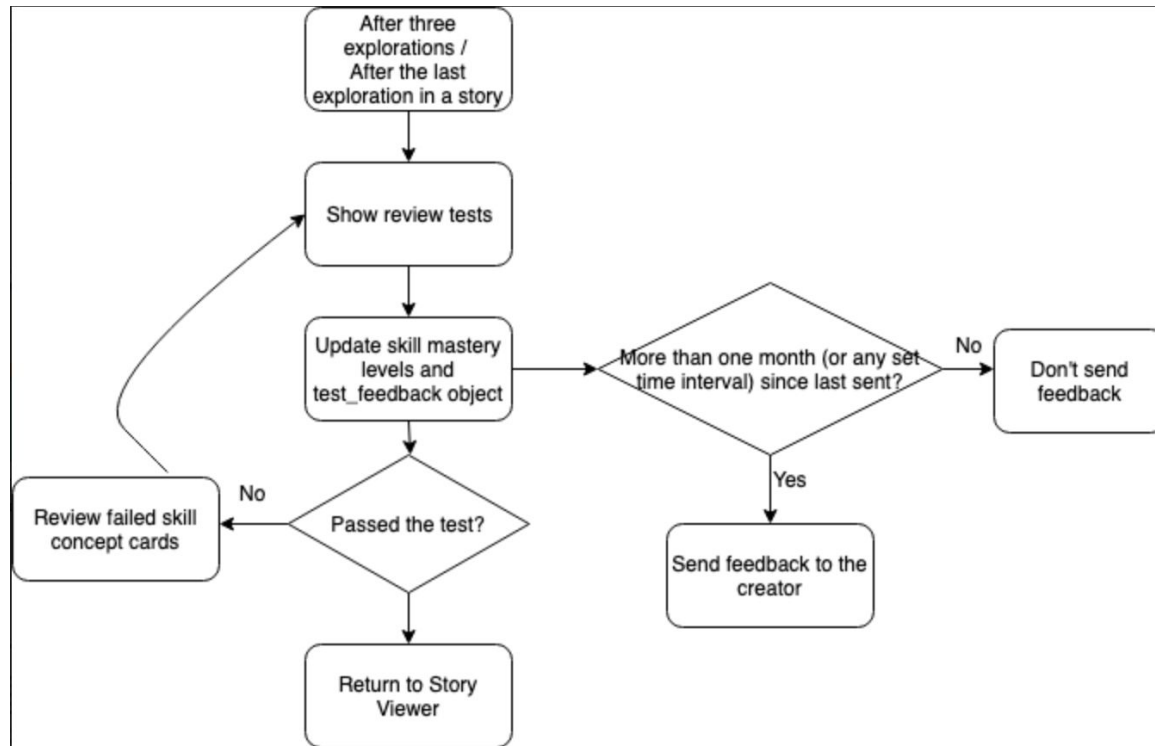
Learner View

After a learner going through three chapters, a review test will show up:
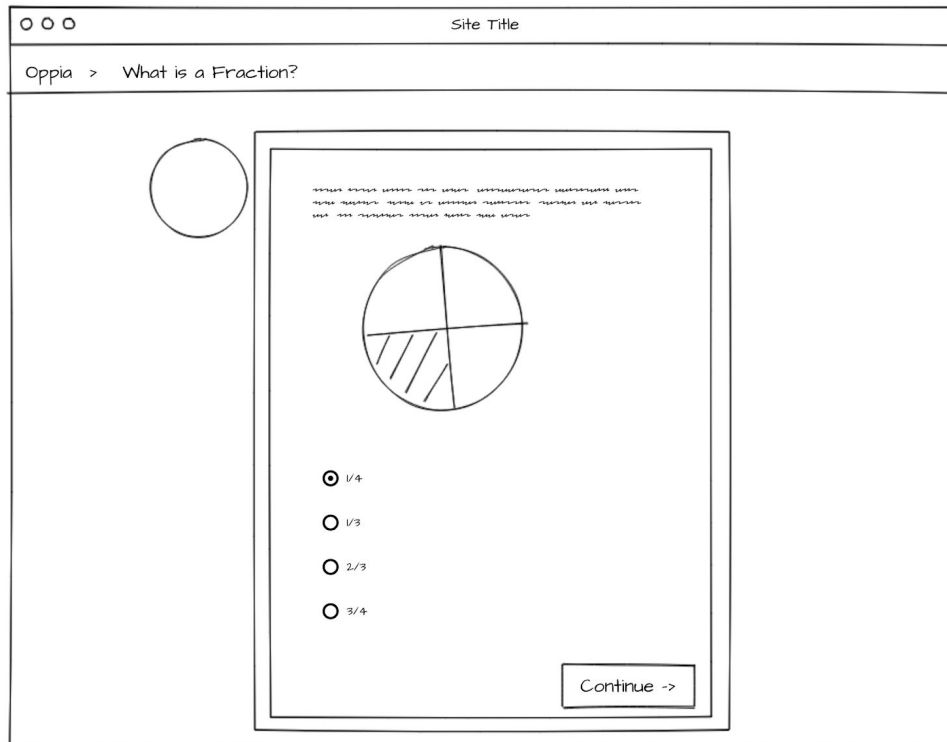
After successfully completing all questions, the learner's performance will be evaluated and the results will be displayed in a modal.

Passed:

The skill names are clickable. When learners click on them, they will be directed to the corresponding skill concept cards, so that they can review the skills after the review tests. After exiting the concept cards, learner will be redirected to the review test result page automatically to check other results.

The star shows how many questions were answered correctly the first time.

Failed:
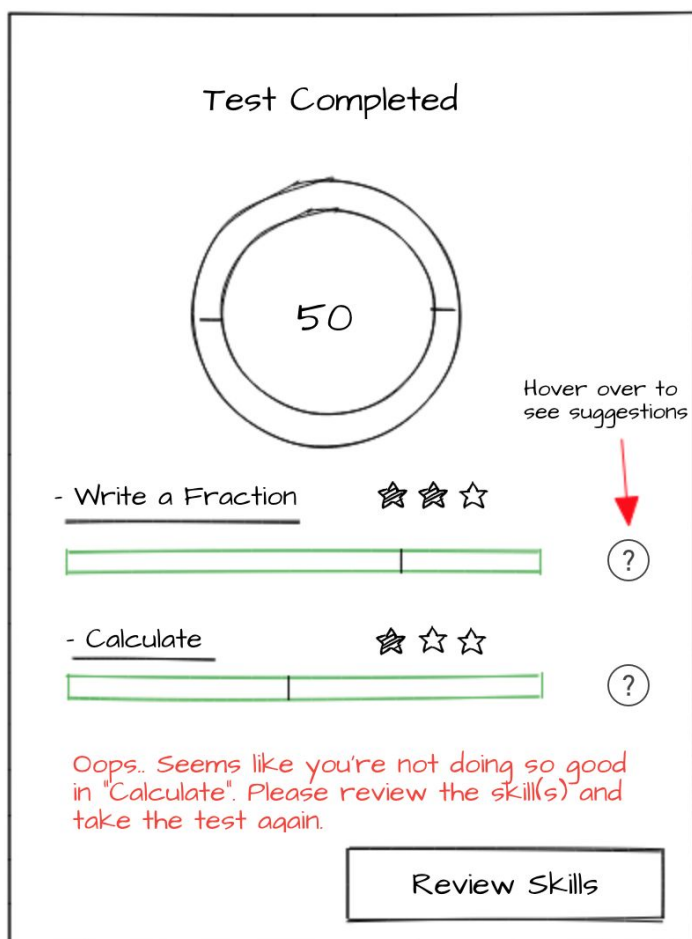
The learner didn't pass the "Calculate" skill (didn't get 2 out of 3 questions correctly), so he/she can't proceed to the next chapter. The learner should click on the button to review the failed skills. After reviewing, a review test containing failed skills will show up. The learner will have to pass this review test to "unlock" the next chapter.

As we can see from the two mockups above, there is a green bar under every skill name. These bars show the mastery levels of corresponding skills. On the right of the bars, the icons show the details of mastery level and give suggestions accordingly.

Hover over the icon we will see: (more suggestion details see Overview - Give Suggestions Using Mastery Level)
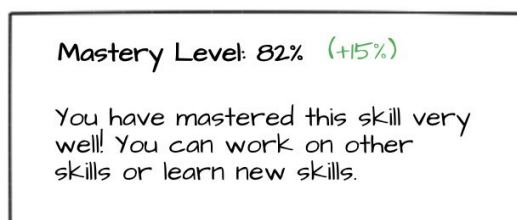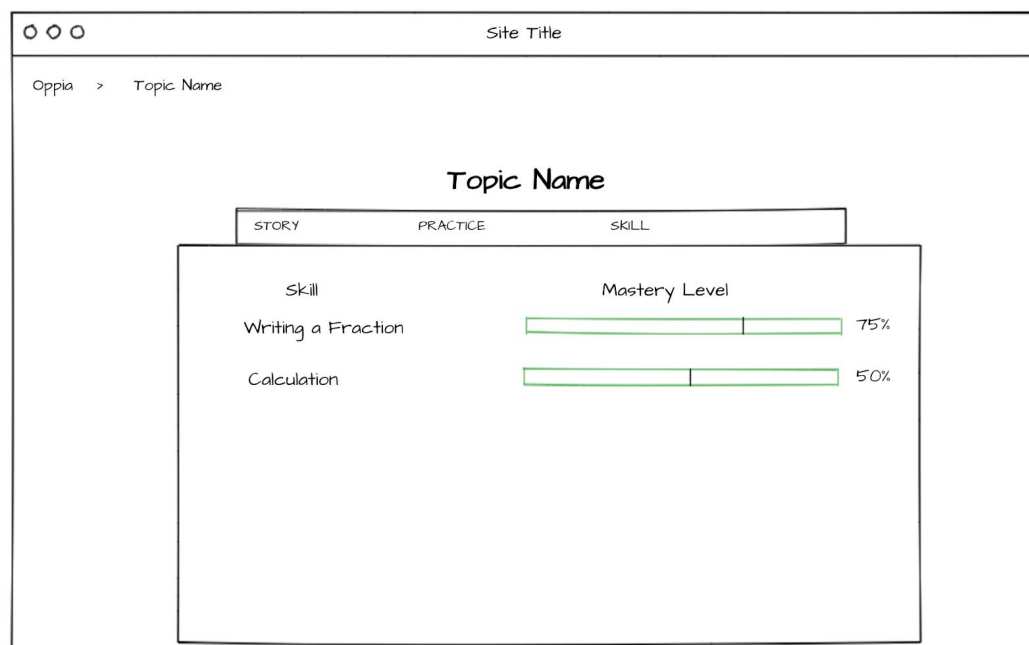
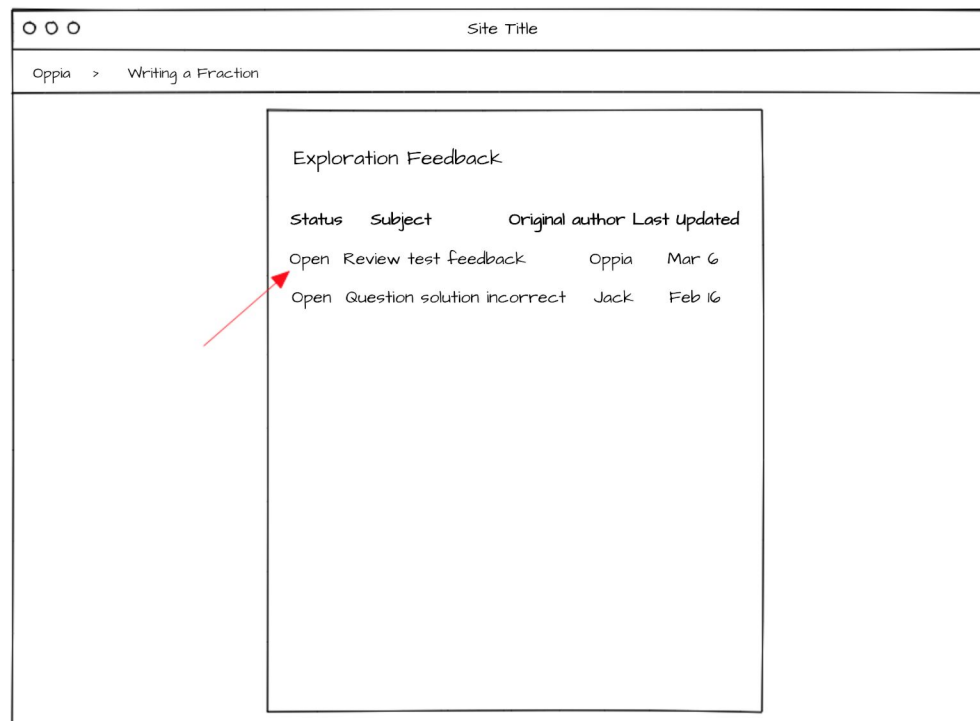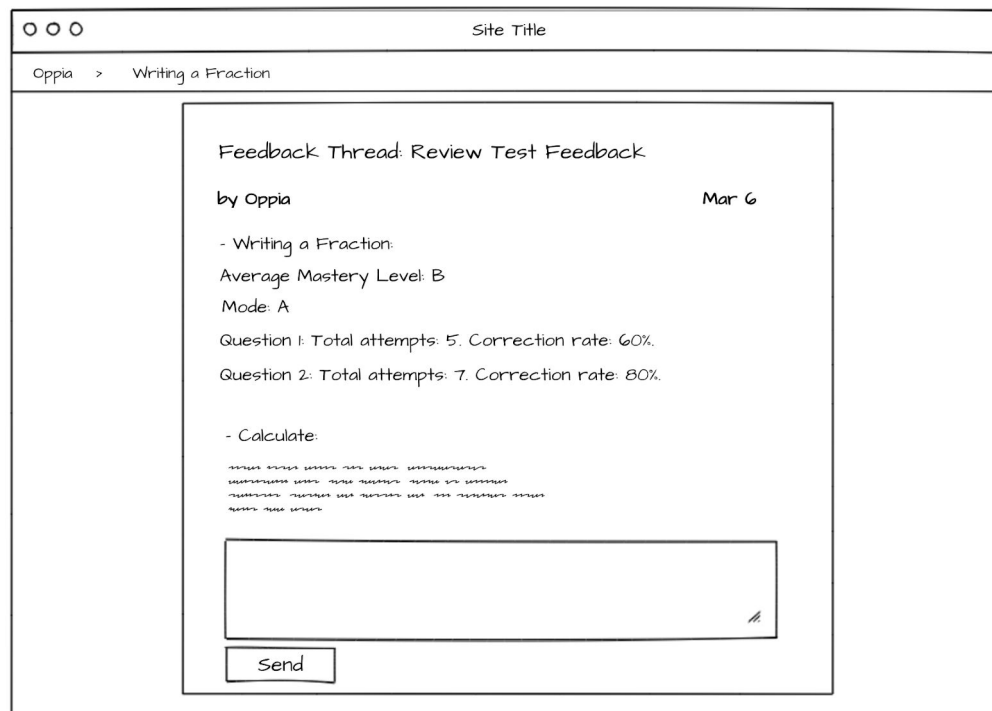Learners will also be able to view a list of skills in the topic as well as their mastery levels. This design allows learners to view the mastery levels anytime they want, and they can practice their skills according to the mastery levels.

Creator View

Creator will receive feedback in the feedback tab in the skill editor every few months, depending on their preference. Users can change the time interval in the Preference tab.

○ ○ ○   Site Title

Oppia   >   Writing a Fraction

Exploration Feedback

Status   Subject   Original author Last Updated

Open   Review test feedback   Oppia   Mar 6

Open   Question solution incorrect   Jack   Feb 16

The creator can click on the first feedback from above to see how learners do after they take the lessons.

○ ○ ○   Site Title

Oppia   >   Writing a Fraction

Feedback Thread: Review Test Feedback

by Oppia   Mar 6

- Writing a Fraction:

Average Mastery Level: B

Mode: A

Question 1: Total attempts: 5. Correction rate: 60%.

Question 2: Total attempts: 7. Correction rate: 80%.

- Calculate:

Send

Creators can get to know which chapter/skill is not sufficient enough and needs improvements. They can also know what learners are weak or good at, so they can tutor them better in the future.

Technical Design

Display Review Tests

- Add a counter variable storing the number of explorations that the learner went through in a story. When the counter%3 == 0, or it is the last exploration in the story, display the review test.
- Getting questions from the backend:
    - Have a list storing the chapter ids that need to be tested in the incoming review test. Clear the list after finishing a review test. Add ids into the list before going through a chapter.
    - Get all skill ids linked to the chapters. Check the length of the list of skill ids. If length is less than 7, get three questions for each. If length is less than 10, get two questions for each. If length is equal to or greater than 10, get one question for each. If length is greater than 15, only get 15 questions linked to 15 skills with relatively low mastery level (this will be implemented after we have mastery level done). Get questions from the backend question bank randomly.
    - In the frontend, fetches the review test questions using the dictionary returned from the backend.
- Initialize the review tests pages using the existing question player.
- Display a modal at the end of the test. Create a template file, which contains a header, a list of clickable skill names, scores, a list of clickable skill names (redirect to skill concept cards), also an option to review the skill concept card. This file will be called every time the learner finishes a review test.

Review Tests Results

- Implement a function keep tracking of the user's score linked to every skill during a review test. The score will be set to 100 by default. If the learner does not get the correct answer at the first time, the score will be deducted by 100 / number_of_questions. If the learner gets the answer correctly at the first time, the score will remain the same. At the end of the review test, return the total score and the correction rate of each skill.
- Have a list storing failed skill ids (details see Overview - Review Tests Results). After the test is over, check if the list is empty.

- If it is empty: Enable the next chapter, and initialize the "good score" modal.
- If it is not empty: Initialize the "bad score" modal. When the "Review Skills" button is clicked, redirect to the concept cards linked to the skills in the list. After reviewing, another review test will be initialized, and use the same method as above to get questions linked to the failed skills from the backend.

Mastery Level

- Add a MasteryLevelModel class.
  - skill_id: the id of the skill that the mastery level data is linked to.
  - user_id: the id of the user that the mastery level data is linked to.
  - mastery_level: a float between 0.0 and 1.0. Set to 0.0 by default.
- Use timestamps to keep track of the following:
  - How many times answers are submitted by the learner.
  - Number of hints used by the learner.
  - Whether the solution is requested.
- Implement functions to do calculations (as stated in Overview - Mastery Level) after pretests, review tests, and practice sessions.

Add Skills Tab in Topic Viewer

- Add a new tab in the topic viewer template file next to story and practice tabs.
- Get skill names linked to this topic and mastery levels linked to skill_id and user_id.

Send Feedback to Creators

test_feedback model

- feedback_id: Generates a unique id for this feedback.
- topic_id, story_id: The ids of which review tests are invoked.
- skill_id: The skill_id which is linked to the questions in review tests.
- creator_id: Id of the creator who created the skill and questions.
- average_score: The mastery level learners got on average.
- mode_score: The mastery level learners got the most.
- questions_stats: A dictionary that stores the question_id, its total_attempts and its correction_rate.
- last_sent: Stores the datetime when it was sent last time.

Since there is a learners sending feedbacks to creators feature already, don't bother to create an extra entry for this. The test feedbacks will show up in the feedback tab too, with the name "Review tests feedback".

The stats in the test_feedback model (total number, correction rate, etc) will be updated every time after a test, but only send feedbacks to creators every month (or any interval they set). Every time a test_feedback is being updated, check the time interval between the current time and last_sent. If less than a month (or other time interval), don't send feedback to the creator. If more than a month (or other time interval), send the feedback.

Give Suggestions Using Mastery Level

- Add bars to display mastery level in the result modals/pages of pretests, review tests, and practice sessions.
- Create an enum with three values IsInPretestMode, IsInReviewTestMode, and IsInPracticeMode to represent different test states. (could be extended if needed)
- Add suggestions.
  - Suggestions for review tests and practice sessions are the same. Details see Overview - Give Suggestions Using Mastery Level.
  - When IsInPretestMode is true, if at least one of the prerequisite skills of a chapter does not have 80%+ mastery level, the chapter will be disabled.
- Check mastery levels of prerequisite skills before pretests. If all of them are above 80%, don't show pretests. If some of them are lower than 80%, show pretests only with skills with lower than 80% mastery level.
- Improve questions selection:
  - Get learner's mastery level from the backend.
  - Add functions to sort questions in the order of abs(question_difficulty - mastery level). The smaller the value is, the more likely it is selected.
  - Select the top n (1-3) results in the sorted list.

Milestones

I plan to have three big milestones (four weeks) based on the timeline provided by Google, and four small milestones (one week) in each big milestone.

Preparation / Community Bonding Period (Now - May 26)

During this period, I will keep making contributions to the community, getting more familiar with the codebase and mentors. One of the issues I am going to work on is implementing the frontend part of question difficulty.

Milestone 1: Display Review Tests and Calculate Scores (May 27 - June 23)

Display Review Tests Breakdown:

- Implement ReviewTestEngineService.js which does the following:
  - Handle answer submitting.
  - Get corresponding questions from the backend.
  - Load the next question when the answer is correct.
  - Stay in the same question and give suggestions when the answer is wrong.
  - Return to the story player after the last question.
- How to get questions from the backend:
  - domain/story_domain.py: Add a function to get all the linked skill_ids for each exploraion_id.
  - Add a function to decide how many questions are supposed to be selected based on the number of skill_ids in total.
  - controller/reader.py: Add a ReviewTestHandler class to handle a GET request and return a review_test_questions_dict, which contains one to three random questions for each skill, using domain/question_services.py.
  - dev/head/domain/question/ReviewTestBackendApiService.js: It fetches the review test questions using the dict returned from the backend. The questions in the dictionary will be used to display in the review tests.
- Tests will be added to make sure:
  - Questions are randomly selected from the question bank linked to each skill.
  - Correct number of questions are selected from the backend for each skill. (number details see Overview - Display Review Tests)
  - Questions can be displayed successfully, and in the correct order.
  - Move on to the next question if answer is correct.
  - Remain on the same page and is able to try again if answer is incorrect.
  - Suggestions will be given is answer is incorrect.
- Add a counter to keep track when to display the review tests:
  - In ExplorationPlayerStateService.js, when the counter%3 == 0, or it is the last exploration in the story, set variable inReviewTestMode (default false) true.
  - When inReviewTestMode is true, start ReviewTestEngineService and initialize the services.
  - Once the final question is done, should have a function to move back to the story player.
- Tests will be added to make sure:
  - Review tests always show up at the right time.
  - Learners will be returned to the story player page after they are done.

Score Calculation Breakdown:

- Add functions in ReviewTestEngineService.js which do the following:

- ○ Have a score variable which is set to 100 by default, and do calculation every time after the learner submits an answer, based on the value of answerIsCorrect (boolean).
  - ○ Return the skill_id being tested, question_id, and scores accordingly.
- ● Tests will be added to make sure:
  - ○ The score calculation gives the correct result.
  - ○ The function returns the same information as what was in the review tests.
- ● Display the modal every time after finishing the review test:
  - ○ Add a template file review_test_result_directive.html.
  - ○ In ReviewTestEngineService.js, open the modal after the last question.
- ● Tests will be added to make sure:
  - ○ The result modal will show up every time after the last question.
  - ○ The information being shown is correct in different situations. (Single/multiple skills, different scores etc.)

Sub-milestones

Submission Date	Merge Date	Description
June 2	June 5	Get questions dictionary from the backend. Implement ReviewTestEngineService and relevant functions (loading questions, redirecting to next question cards, etc). Add tests. (Bullets 1-3 in Display Review Tests Breakdown)
June 9	June 12	Handle displaying review tests. Add relevant tests. (Bullets 4-5 in Display Review Tests Breakdown)
June 16	June 19	Handle review tests scores calculation and display the result modal. Add relevant tests. (Score Calculation Breakdown)
June 23	June 25	Buffer period. Do manual testing to look for bugs and fix them.

Milestone 2: Review Tests Results and Mastery Level (June 24 - July 21)

Review Tests Results Breakdown:

- Add a new list in ReviewTestEngineService.js. After the test is over, add id of every skill which has under 1/2 correction rate.
- Change the previous mastery level modal to two situations (pass or fail). Update corresponding template files.
- When the list is empty, change nothing. (Keep milestone 1)
- When list is not empty:
  - Open the mastery level failed modal.
  - Use ConceptCardBackendApiService.fetchConceptCard() to get a dict of concept cards linked to the failed skill_ids. Call ConceptCardObjectFactory to create the concepts cards from the backend and load them.
  - After closing the concept cards, a review test will be initialized again, using the same method as above. The difference is this time it only get questions linked to the skill_ids in the failed list.
  - After completing the review test, check the list again. If empty, proceed. If not, repeat the same steps as above until empty.
- Tests will be added to make sure:
  - Failed skill ids are added into the list.
  - Two different modals are triggered under different situations.
  - Concept cards linked to the skills can be get from the backend.
  - Concept cards can be opened after clicking the button.
  - Review tests open automatically after reviewing concept cards.

Mastery Level Breakdown:

- Add a new class MasteryLevelModel in storage/skill/gae_models.py and tests in gae_models_test.py. (Parameters see Technical Design - Send Feedback to Creators).
- Define class in domain/skill_domain.py to create a domain object.
- Define functions in domain/skill_services.py to get mastery level data linked to skill_id and user_id, and other relevant functions.
- Backend tests will be added to test functions above.
- Update mastery level:
  - Add functions to handle calculations under different situations.
  - Every time learner submits answers, uses hints or solutions, functions above will be called to do corresponding calculations.
  - In each test's Handler, handles PUT requests for updating mastery levels at the end of tests.
- Tests will be added to make sure:
  - Mastery level data will be updated successfully when answers/hints/solutions are triggered.
  - Mastery level data will be increased or decreased by the correct amount under different situations.

- Display mastery level information on the result modals.
  - Update the template file to add mastery level bar.
  - Add frontend tests to make sure it can be displayed correctly.

Sub-milestones

Submission Date	Merge Date	Description
June 30	July 3	Divide results into passed/failed modes and store failed skill id for review used. Add relevant tests. (Bullets 1-2 in Review Tests Results Breakdown and relevant tests)
July 7	July 10	Redirect learners to concept cards and automatically start a review test afterwards. Add relevant tests. (Bullets 3-5 in Review Tests Results Breakdown)
July 14	July 17	Add new mastery level model, domain, services, and tests for each of them. Handle updating mastery levels after every test/practice. Add relevant tests. (Bullets 1-6 in Mastery Level Breakdown)
July 21	July 24	Add mastery level information on every result modal/page. Do manual tests and write integration tests. (Bullet 7 in Mastery Level Breakdown and test the entire milestone)

Milestone 3: Add Skill Tab, Send Feedback to Creators and Use Mastery Levels to Improve Tests (July 22 - Aug 18)

Add Skill Tab Breakdown:

- Add a skill tab in topic_viewer.html.
- Add a Handler in controllers/topic_viewer.py to get skill names and mastery levels from the backend.
- Tests will be added to make sure:
  - The new added tab will be displayed in the topic viewer.
  - Skill names and mastery levels can be gotten from the backend correctly.
  - Skill names and mastery levels can be displayed correctly.

- ○ If a skill is linked to multiple topics, the skill mastery level can be updated in the skill tabs of all of them.

Feedback Breakdown:

- Add the new TestFeedbackModel as well as its related functions in storage/feedback/gae_models.py and tests in gae_models_test.py.
- Only create a new feedback message every month:
  - ○ Check if a test feedback object linked to a topic_id already exists, update the corresponding data. If not, create a new one.
  - ○ Everytime the test feedback is updated, check the time_interval (current_time - last_sent). If the time_interval is smaller than a month, don't do anything. If it is bigger than a month, create a new feedback message, set last_sent to the current time, and set other data to 0.
- Make use of and edit the existing feedback_services.py and feedback_doamin.py to create a new feedback message and send the message to the creator.
  - ○ Add a create_test_feedback_message function to format the message body in the designed format (details see Overview).
  - ○ Add a function in ReviewTestEngineService.js to trigger the creating action (make use of FeedbackThreadObjectFactory.js).
- Tests will be added to make sure:
  - ○ test_feedback object will be created/updated every time after a review test.
  - ○ Feedbacks can be formatted correctly.
  - ○ Feedbacks can be sent to the creator and show up in the feedback tab.

Tests Improvement with Mastery Levels:

- Since displaying mastery level information is already handled in milestone 2, we only need to consider adding suggestions for different mastery levels.
- Suggestions details see Overview - Give Suggestions Using Mastery Level.
- Tests will be added to make sure:
  - ○ Correct suggestion will be given under every different situation.
  - ○ In pretest mode, chapter is disabled if its prerequisite skill's mastery level is under 80%.
- For pretests:
  - ○ Check mastery level of every prerequisite skill before pretests.
  - ○ In ExplorationPlayerStateService.js, don't set isPretestMode to True when every skill's mastery level is above 80%.
  - ○ Currently pretests get questions from the backend using prerequisite_skills_list. Replace that with a list containing only skills with mastery level under 80%.
- Tests will be added to make sure:
  - ○ Only show pretests when some of the prerequisite skills are under 80%.

- ○ Only show questions linked to those skills.
- Select questions based on difficulty and mastery levels:
  - ○ In controllers/reader.py: Add a function to sort the questions got from the backend based on abs(question_difficulty - mastery level) in ascending order, and return the top n results.
- Tests will be added to make sure:
  - ○ Questions selected are the ones that have the most similar difficulty as the learner's mastery level.
- Set an upper limit 15 to number of questions in review tests, and only select 15 questions linked to skills with relatively low mastery level when there are more than 15 skills linked to the three chapters. Use the same way to get questions from the backend as milestone 1. But after getting skill_ids, sort the list in the ascending order of corresponding mastery level. Only use the skill_ids of index 0 to 14.
- Tests will be added to make sure:
  - ○ When number of skills linked to the three chapters is larger than 15, only select 15 questions.
  - ○ The questions are selected from skills with relatively low mastery levels.

Sub-milestones

Submission Date	Merge Date	Description
July 28	July 31	Add skill tab frontend and its handler to get data from the backend. Add relevant tests. (Add Skill Tab Breakdown)
Aug 4	Aug 7	Add TestFeedbackModel and use existing feedback features to send review test feedbacks to creators. Add relevant tests. (Feedback Breakdown)
Aug 11	Aug 14	Implement suggestion feature and other improvements for pretests, review tests, and practice sessions. Add relevant tests. (Tests Improvement with Mastery Levels)
Aug 18	Aug 21	Buffer period. Fix any bugs reported and manually test to look for bugs. Prepare for final submission.

Future Projects

1. Expand test feedback to all tests, including pretests, review tests, and practice sessions. This improvement will be done during GSoC if time allowed.
2. Improve the content of the test feedback to show more useful information.
3. Improve questions selection based on mastery level for pretests and practice sessions.
4. Add an individual skill player so that learners can review skills anytime they want.

Summer Plans

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

I will stay in America (west coast) throughout the summer. The time zone will be Pacific Daylight Time (GMT-7).

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

I won't have any other full-time work during summer. For most of the time, I will spend 6-8 hours a day, 6 days each week (36-48 hours a week) working on this project. My finals week will be in June 10 - June 14, so during the five days, I will only spend 3 hours a day, but I will catch up on the weekend (June 15 - June 16).

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

My final exams end on June 14. I won't have any other jobs, summer classes after that.

Communication

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

My contact information:

email: wushiqi1998@gmail.com (preferred)

phone number: +1-4252867071

I will also be active on Gitter.

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

I plan to contact my mentor everyday to share my progress and ask questions on Gitter/Google Hangout based on mentor's preference, and there will be a weekly meeting so we can talk about my progress during every small milestone (one-week duration) and my plan for the next week.