Replace Fallbacks With Hints

Project Details

Synopsis

Primarily, my interest in working with Oppia is because of the philosophy it has adopted – e*ducation for all*. Making available quality resources and tools to students regardless of where they are; As a college student in a developing country I'm able to recognize the importance of such an initiative and therefore I am excited to contribute to this cause.

What makes Oppia unique is that it offers an interactive learning experience as opposed to static video based lectures or text which are not very effective as they do not engage the learner as much. Therefore, it is important to ensure that students feel engaged and comfortable with the lessons. The concept of Explorations make sure that a lesson is properly paced and interactive by providing the learner with feedback at each step. If the learner feels lost or is stuck at a particular card, a "fallback" is triggered which provides the learner with hints and sometimes full answers to help him/her advance to the next card. Currently, this fallback is enabled/programmed by the exploration creators which gets triggered after 'X' unsuccessful attempts. However, it is found that learners need not always be ready for these "fallbacks", so a better approach would be to have a student-initiated hint system. This way the student would have more control of when he/she receives a hint. Also, cards may have a final hint which explains how to solve the problem, thus helping the student progress to the next card. This project aims to build such a system, which I think is critical to ensuring that students are able to complete the lessons while having fun and feeling engaged in it.

Prior Experience

I have been contributing to Oppia for about a year and have been the team lead for the Collection Learner View project (completed). I'm familiar with technologies such as AngularJS and python which are used in this project.

Links to PRs:

https://github.com/oppia/oppia/pull/2979

https://github.com/oppia/oppia/pull/3197

https://github.com/oppia/oppia/pull/3054

https://github.com/oppia/oppia/pull/2780

https://github.com/oppia/oppia/pull/2634

Project Plan

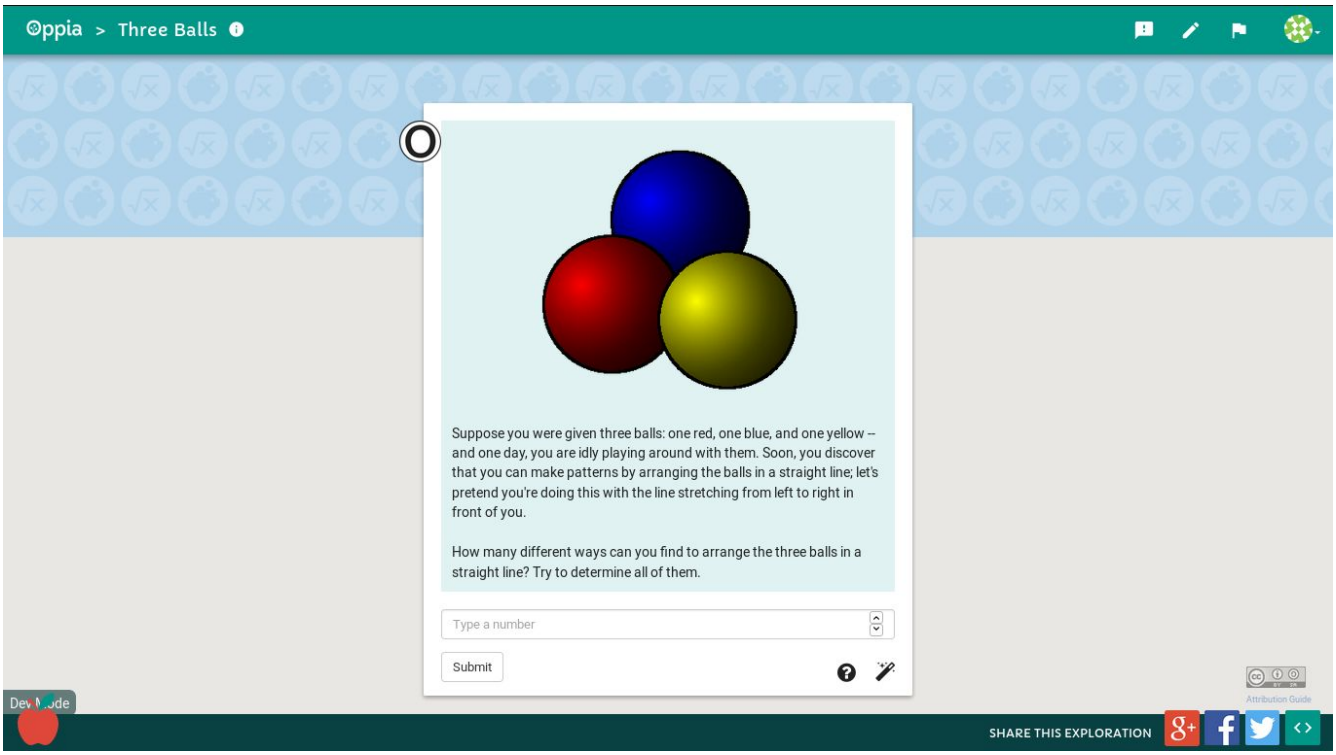
The new “hints” system will consist of two types of hints – (a) one which suggests corrections or provides feedback so that the student gets the answer right on subsequent attempts and (b) one which will provide the right solution to the problem. Both type (a) and (b) hints are student-initiated. Type (a) hints can be triggered on clicking a “question mark” icon which will be present at the bottom right corner of the exploration player, this will trigger a fallback-like response/hint. This response is specified by the exploration creator at the time of creation. Type (b) hints can be triggered by clicking on the magic wand icon at the bottom part of the exploration player, type (b) however does not provide hints but instead solves the current problem directly and presents the student with the “continue” option. Thus, allowing the student to know what the solution was and also be able to progress to the next card. Since there can be multiple solutions to a particular problem the exploration creator would have to specify which solution the magic wand icon would use and in turn which card the student would be directed to.

The fallback system currently in place need not be completely replaced (implementation-wise), it can be reused in a way. Type (a) hints will work in a similar way compared to the fallback system, the difference is that type (a) hints are triggered by the student when the corresponding icon is clicked-on as opposed to the automatically triggered fallbacks. Therefore, the fallback system can be modeled such that it gets triggered only when the icon is clicked instead (eg. submitAnswer() in PlayerServices.js need not check if the number of resubmits matches the num_submits specified in the fallback to change the dest, feedback and param_changes, instead a showHint() function might handle that). Also, unlike the fallback system, the new hints system must be enabled by default. This is because the idea of having hints is to enable students to have more control of when and whether they want to see hints, so it is fair to have hints enabled by default so they can choose whether they want a hint (clicking on the icon) or not. However, the creator can choose to not give hint to particular problem, in that case the icon will be disabled for that problem. Type (b) hints can work independent of the fallback system, however as mentioned before, when there are multiple solutions to a problem the exploration creator may need to specify which solution to use and which card to display next, this can be specified in a “hints editor” which will function in a similar way as the current “state fallbacks editor” does. The hints editor will require the exploration creator to specify which card the student will be redirected to on clicking the (type

(b) hint) magic wand icon. Also, the type (a) hint to be displayed when clicking on the bottom right icon can be specified here.

NOTE: The fallback system will be replaced ie. explorations will no longer have fallbacks but instead a student-initiated hint system will be in place.

Mocks for the two types of hints (a & b):



Note: The question mark icon represent type (a) hint and the wand represents type (b).

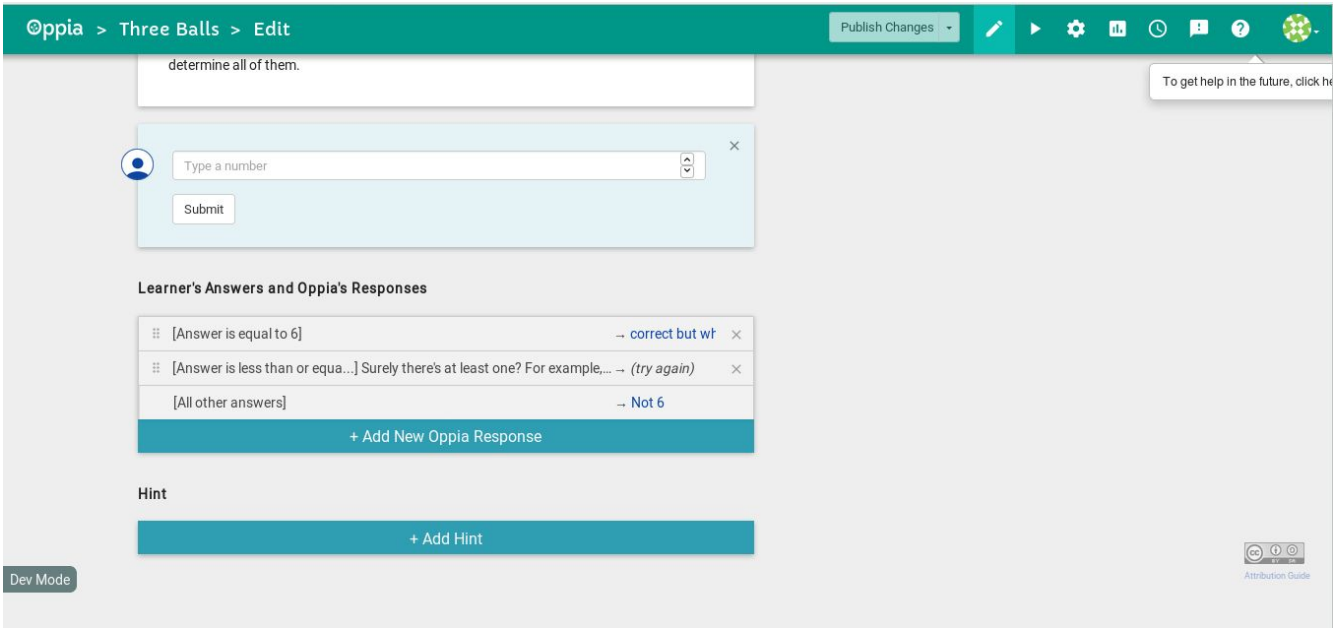
The wand icon will be disabled (not clickable) by default. It becomes clickable once the hint is used (type (a) question mark).

User testing was done with 7 users. The mock-up shown above was provided to the users and they were briefly explained about the context (ie. they had submitted wrong answers previously and are currently provided with this screen.) They were asked what each of the icons/buttons represented and their answers were:

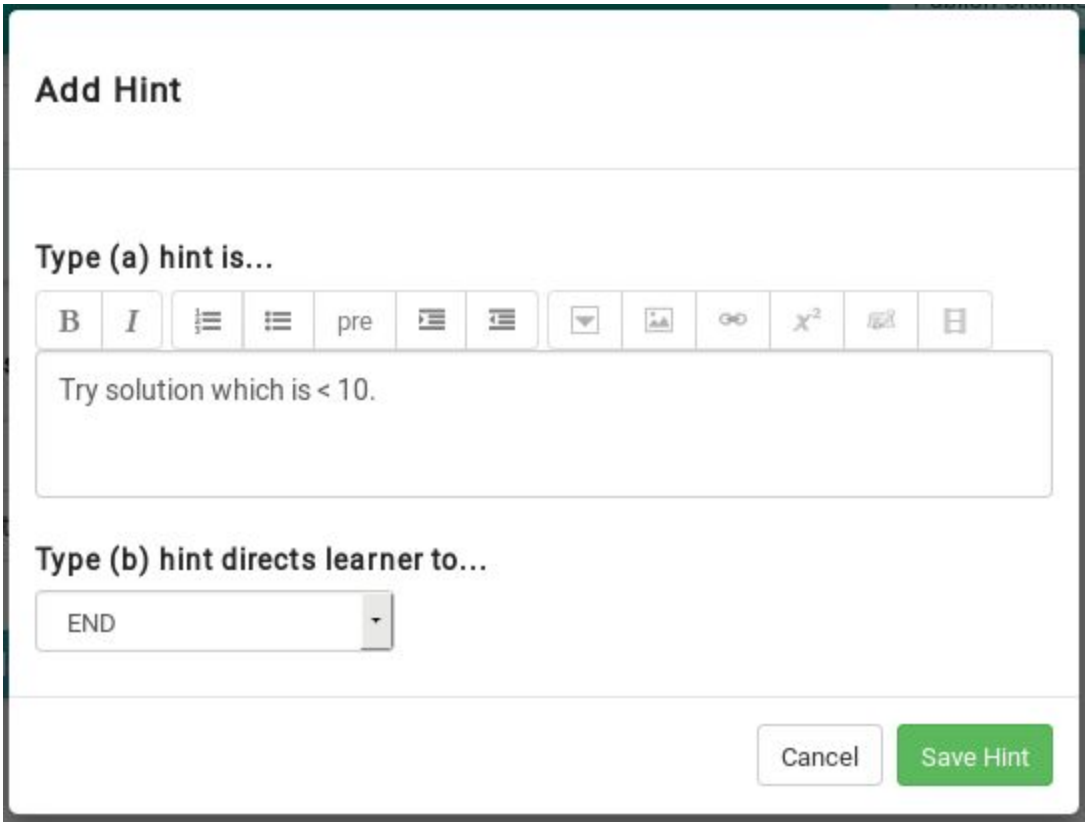
1. SUBMIT -- All 7 of them guessed rightly (ie. submits the answer)

2. Question mark icon -- 5 of them called it “help” and the other 2 called it “hint” but when asked about its functionality, all 7 guessed correctly and said it would provide them with advice/hint/help to solve the problem.
3. Magic Wand -- 4 users guessed correctly and said it would “solve” the problem. The 3 other users thought it would “reveal” the answer.

To avoid confusion between the icons, a tooltip feature can be used here. Hovering on the “?” icon the tooltip will display “Hint” and hovering on the wand will display “Solution” or “Solves the problem”.



Option to add Hint in exploration editor.



Hint Editor

The Hint Editor can be accessed by clicking on a “Add Hint” button in the general exploration editor page. It will look similar to the current “Add Fallback” button. However, there can only be one hint to a problem (card) and thus the “Add Hint” button will be disabled once a hint is added. The hint added can be modified or deleted.

Implementation Strategy

Milestone #1: (May 30 – June 30)

1. Finalize the mocks for the player UI as well as the creator UI. (May 30th - June 5th)
2. Perform necessary migrations and implement necessary datastore and backend logic changes. (June 6th - June 16th)
    - Implement “HintsObjectFactory” which will replace “FallbackObjectFactory”.
    - Implement “generateHintsFromBackend()” replacing “generateFallbacksFromBackend()” and other changes to recharacterize fallbacks to hints in “InteractionObjectFactory”.
    - Implement hints functionality in “PlayerServices” and “EditorServices”.

- Implement Hint class in “exp_domain”. Unlike the Fallback object which consists of trigger and outcome, the new Hint object would only require outcome because trigger is no longer necessary to reroute the user flow to the given outcome. Outcome provides the destination state (destination on clicking wand icon) and feedback (type (a) hint text).
- Current production data for fallbacks need to be migrated to adapt to the new hints system, this should be simple considering that the fallback object and hint object are quite similar in structure.

3. Implement necessary UI changes for the new hints system. (June 17th to June 30th)
    - UI components for the two types of hints in the exploration player.
    - UI components in the exploration editor -- recharacterizing “state editor fallbacks” to “state editor hints”and making necessary logical changes by replacing “stateFallbacks” and implementing “stateHints”. Also, implement “HintsEditorDirective” replacing “FallbackEditorDirective”.

Milestone #2: (July 1 – July 28)

1. Iterate on the UI until we have evidence (from user testing) that students find the hints functionality intuitive, and that creators are aware that they need to specify hints and find it easy to do so.
    - A ratings module can be attached to the hints system to find out if a particular hint was useful or not. This module can either be star based which will allow users to specify how useful the hints were on a scale of 1-5 stars or a binary thumbs up / thumbs down type system can be implemented which will tell if a user found a particular hint useful or not. Such a module can be useful in determining what a good hint is.
    - The ratings module will be triggered and displayed along with the hint when the hint icon is clicked. It can be either part of the div containing the hint or it can be a small pop-up type box which will extend out from the hint icon.
        - Implement backend for ratings module. Number of stars or thumbs up/down might be a property of the hint object. (July 1st - July 4th)
        - Implement frontend for the stars/thumbs. (July 5th - July 9th)
    - As for feedback on the UI and overall intuitiveness of the hints system, we could conduct a survey using google forms which students can take up. This could be a part of the playtesters feedback form.

        - Finalize questions to be asked in the feedback form. (July 10th - July 12th)
        - Collect feedback and identify tasks for milestone #3. (July 13th - July 20th)
    2. Complete and launch the frontend. (July 21st - July 28th)

Milestone #3: (July 29 – August 29)

1. Based on the feedback gathered in milestone 2, find other ways to help or encourage learners to complete the explorations they start, and implement at least one of these.
    - One way to encourage learners is to involve some form of gamification eg. points being earned on completion of explorations and spending points to unlock hints. But as per the discussion on the mentors mailing list, it was decided that this would require site wide changes and is beyond the scope of this project.
    - Another possible way to encourage learners is to show relevant statistics about other users who have played a particular exploration eg. “you are ahead of 60% of fellow explorers, keep going!”, “Uh oh! Looks like 30% of fellow explorers were stuck here. Would you like a hint?”. This would trigger a competitive spirit among explorers in a positive way, while also encouraging them to complete the explorations they start.
        - Implement necessary backend for usage statistics. (ie. number of students failed at a particular card.) (July 29th - August 8th)
        - Finalize mocks for showing these messages. (August 9th - August 13th)
        - Implement the frontend according to the mocks. (August 14th - August 21st)

Summer Plans

I reside in India and belong to the IST timezone. During the 2nd week of July, I’ll be taking up a job. However, the initial few months will only be a training period and shouldn’t take up too much of time, which means I’ll have more time for the GSoC project.

Before the 2nd week of July, I should be able to spend about 6-7 hours per day (both weekdays and weekends) since I do not have any other commitments during that period. After the 2nd week of July, I would be able to spend 4-5 hours on weekdays and 6-7 hours during the weekends to work on this project. That said, I should be able to spend about 35+ hours per week on average.

Communication

Email: kevintab@tutanota.com

Skype: kevin.thomas.ab

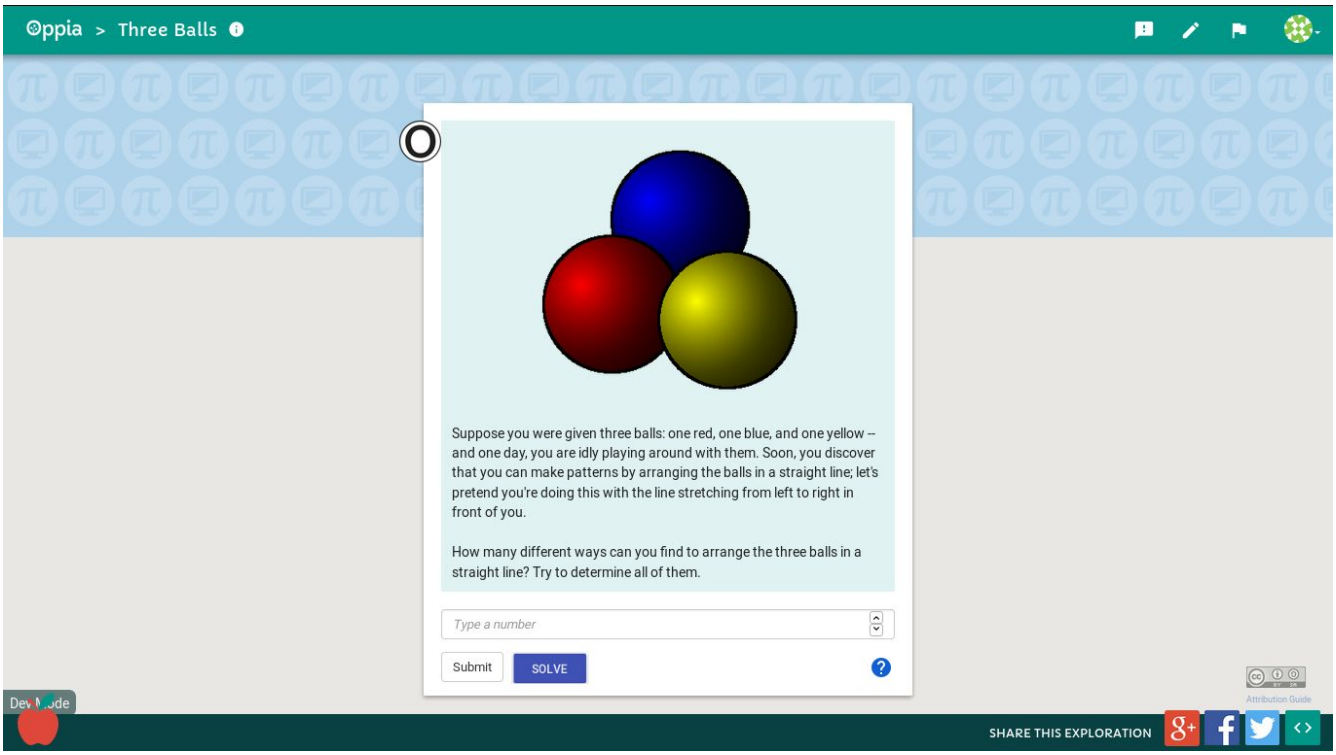
GitHub and Gitter: @kevintab95

Preferred method of communication is email and gitter. I will be able to send weekly updates on the current progress and attend skype meetings when required.

Appendix:

Alternative Mocks for user testing:

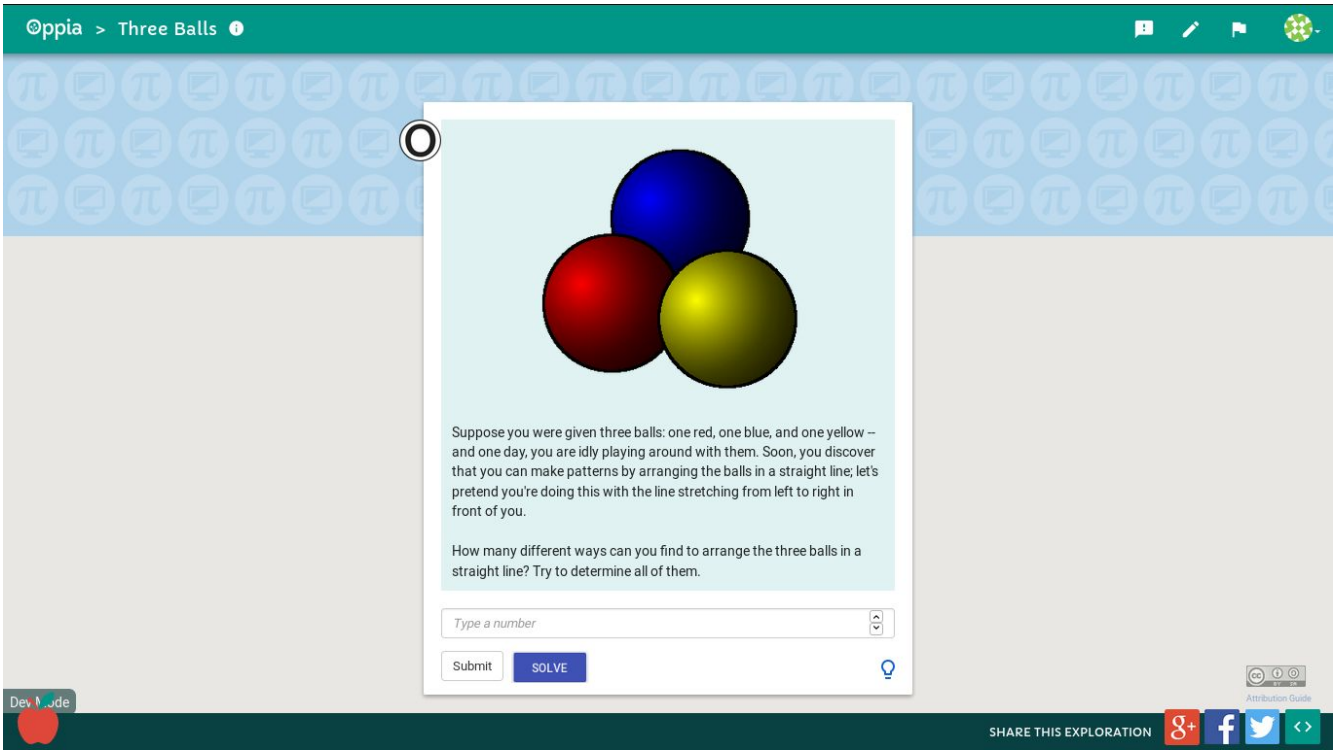
1. Question mark icon for type (a) and SOLVE button for type (b):



User Feedback:

Users guessed the right functionality for the question mark icon. However, some users were confused whether to hit the submit button or the solve button on entering an answer.

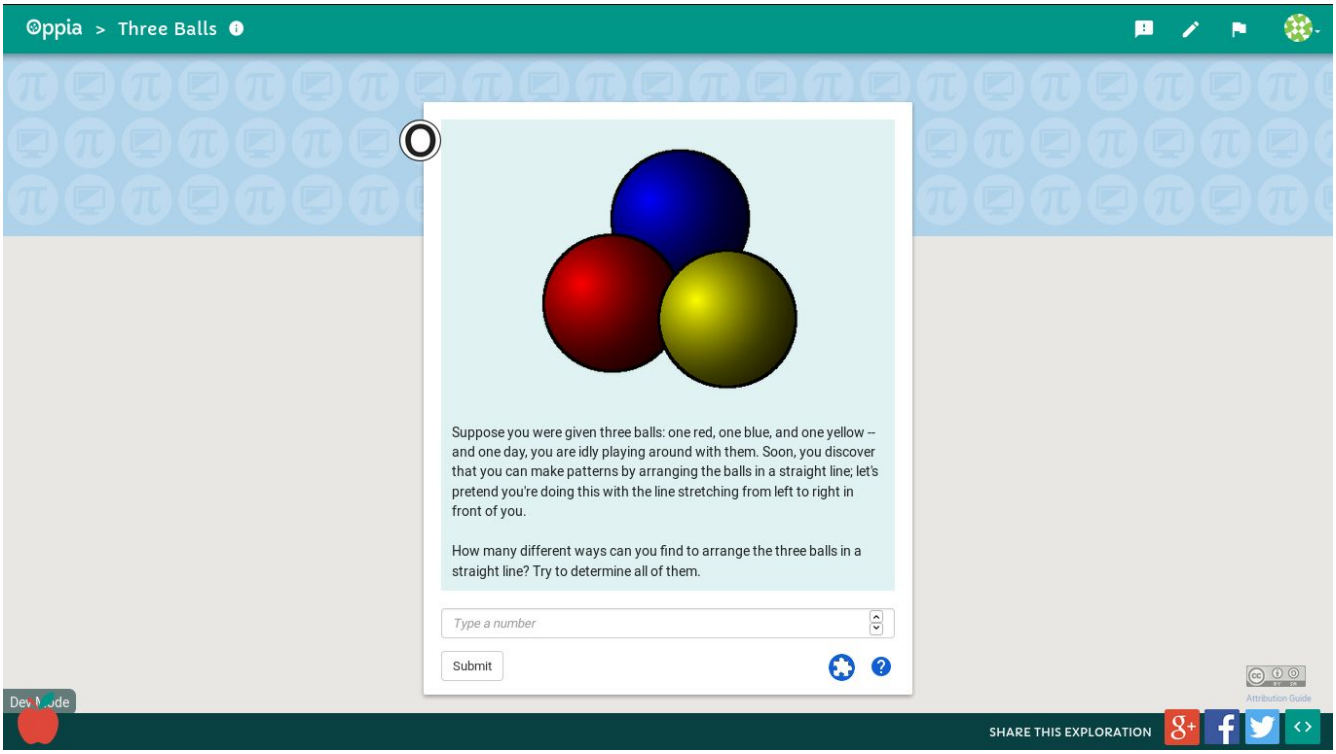
2. Lightbulb icon for type (a) and SOLVE for type (b):



User Feedback:

Most users were able to rightly guess that clicking on the lightbulb would give them a hint or “clue”. They were still confused about whether to hit submit or solve to answer a question.

3. Question mark icon for type (a) and Jigsaw puzzle icon for type (b):



User Feedback:

The function of the submit button and the question mark icon were rightly guessed by users. But most of them were not sure about what the jigsaw icon would do (Mostly because the icon didn't resemble a puzzle piece well enough).