Google Summer of Code

Sugar Labs

Music Blocks Project Blocks Reorganization

Basic Information:

Name	Daksh Doshi
Github	daksh4469
LinkedIn	Daksh Doshi
Email	ddaksh.2711@gmail.com
Phone	+91-7874034708
IRC Nickname	daksh4469
Location	Ahmedabad, India
Time Zone	India (UTC + 5:30)
Languages	Gujarati/Hindi, but I am proficient in speaking, reading, writing, and understanding English.

University Information:

- University: Indian Institute of Technology, Roorkee
- Majors: Computer Science and Engineering
- Current: II Year (Graduation expected in 2023)
- Degree: Bachelor of Technology (4 Year Program)

Contact Information:

Typical working hours include:

- UTC 0400 - UTC 0700 hrs (IST 0930 - IST 1230)
- UTC 0830 - UTC 1400 hrs (IST 1400 - IST 1930)
- UTC 1600 - UTC 1930 hrs (IST 2130 - IST 0100)

I can start my day 2 hours early or late, and I will be reachable anytime through my Mobile No. and Email.

Coding Skills:

Programming Languages and Frameworks:

- Fluent in HTML, CSS, SCSS, Javascript, Typescript, C/C++, Java, Python
- Sound knowledge of OOPs
- **Web Frameworks:** React, Django, Django-REST, Nodejs, Express, Mongoose
- **Libraries:** Bootstrap, EJS, Material UI, jQuery, Socket.IO, CUDA Library, C++ Standard Library
- **Databases:** MySQL, MongoDB
- **Utilities:** Figma, Postman, Firebase, MongoDB Atlas, Heroku, Docker

Development Environment:

- **Ubuntu** 20.04
- **Visual Studio Code** as IDE supported by a range of extensions.
- **Linux Shell**
- Chrome Dev Tools
- **Git** for version control

Apart from the technologies listed above, I have sound knowledge of MERN Stack and MVC Architecture. My other interests include Web Design and Prototyping, and Competitive Programming.

About Me:

I am Daksh Doshi, a sophomore at the **Indian Institute of Technology, Roorkee**, where I am pursuing Computer Science and Engineering. I was introduced to software development and programming in my freshman year. Since then, I have explored various fields such as Cryptography, Web Development, Data Structures, Algorithms, and Computer Architecture. I developed a passion for Web Development and Design in my first semester at college. I have been learning new technologies and their applications every day since. I have been contributing to open-source for about three months now and am adoring it. I have been working with a team to refactor the entire website of UBA-IIT Roorkee. UBA (Unnat Bharat Abhiyan) is an organization that aims to aid the rural areas of India by addressing the issues and solving them through appropriate and sustainable technologies and by organizing suitable events.

Past Projects:

- EndGem: (*Git Repository*)
    - A full-stack Web Application built on NodeJS through Express.js Framework and MongoDB Atlas on the back-end and EJS templating on the front-end along with Passport Authentication for users.
    - Built to organize different types of documents according to their courses. Features include downloadable content, top downloads to date, and the ability to upload and delete documents.

- CovidWelfare: (*Frontend Git Repository* | *Backend Git Repository*)
    - A full-stack Web Application, using a 2FA(2-Factor Authentication) system and built on React(JS) for front-end and Django on the back-end with a REST API built using Django REST Framework.
    - Built to enable people to remotely help people in need of resources in the trying times of Covid-19. Connects the users by SEEK and PROVIDE functionality.
    - Provides the real-time locations of users with the help of Google Maps API. Incorporates a Notification System to notify the users in need of resources and the provider users.

- Sorting Visualizer: (*Git Repository*)
    - A React(JS) based Web Application to enable visualization of Bubble Sort Algorithm.
    - Includes the feature of manually setting the array size for incorporating a better understanding of the algorithm.

This is not an exhaustive list of all the projects. Some of my other projects can be found on my GitHub profile here.

Contributions to Sugar Labs:

I am an active contributor to Sugar Labs for the past 3 months now. This has helped me understand the codebase of musicblocks better, and I now feel comfortable working with it. Contributing to musicblocks has helped me comprehend how it works internally and understand the interactions between various components. I am working with React (Javascript) for the past year and have built several projects using it. Recently, I have been learning Typescript and its applications with React library. Now, I feel pretty comfortable using Typescript with React.

I have contributed to the musicblocks and musicblocks-v4-lib repositories. It has been a great experience contributing to this organization, participating in various discussions, receiving constructive feedback from members and peers, and learning from them. My contributions in sugarlabs/musicblocks can be widely classified into three categories: Porting to ES6/Linting/JSDoc/Documentation/Refactoring, Bug/Regressions Fix, and Enhancements/Features added. The statistics of my contribution are given below:

- Pull Requests (PRs): 44 (28 merged, 8 open, and 9 closed)
- Commits: 61 (1739++, 1188--)
- Issues: 1 (1 closed)

Pull Requests:

sugarlabs/musicblocks:

Porting to ES6, Linting, JSDoc, Documentation and Refactoring:

- #2760 (merged): Update a function to ES6 Arrow Function in js/toolbar.js
- #2764 (closed), #2773 (closed): Add class to js/widgets/temperament.js and port ES6 to syntax.
- #2810 (closed), #2811 (merged): toolbar.js: Prettify, Linting and JSDoc documentation
- #2812 (merged): widgets/jseditor.js: Linting and Prettify
- #2814 (closed): utils/musicutils.js: Pretiffy,linting and JSDoc Documentation
- #2817 (merged): utils/platformstyle.js: Pretiffy, Linting and JSDoc Documentation
- #2818 (merged): basicblocks.js: Pretiffy,linting and JSDoc Documentation
- #2819 (merged): mxml.js: Prettify and Linting
- #2821 (merged): notation.js: Linting and prettify
- #2824 (merged): widgets/statistics.js: Add JSDoc Documentation
- #2827 (merged): FAQ/README.MD: Fixed some typos and grammatical mistakes.
- #2830 (merged): turtleactions/DictActions.js: Linting and Prettify
- #2831 (merged): Linting and Prettify: All files in js/turtleactions
- #2833 (open): blocks.js: Linting, pretiffy and removed debug logs
- #2835 (merged): blockfactory.js: Add global locations and constructor JSDoc
- #2905 (open): palette.js: Linting and Prettify
- #2915(open): musickeyboard.js: Add ES6 Class

Bug/Regressions Fix:

- #2807 (merged): Fix Bug in Arbitrary Edit Tab and improvements in Tempo Widget.
- #2837 (merged): pitchstairecase Bug Fix: Sound keeps playing even after closing the widget.
- #2845 (merged): Bug Fix, Temperament Widget: Play and Stop not working properly
- #2848 (open): phrasemaker.js: Bug Fix, Linting, and Prettify.
- #2854 (merged): Bug Fix: Pitch Staircase Widget
- #2863 (merged): Fix regressions in MusicKeyboard widget
- #2878 (merged): Bug Fix: Tooltip of Collapse Icon
- #2891 (open): Bug Fix: phrasemaker widget plays when no notes are added
- #2900 (open): statistics.js: Add global locations and bug fix
- #2902 (open): BugFix: rhythmruler widget does not render individual pause buttons

Enhancements/Features added:

- #2776 (merged): Update UI of Temperament Widget
- #2832 (merged): WidgetWindows: UX Enhancement
- #2838 (merged): widgets/status.js: Improved UI of status widget
- #2855 (merged): Oscilloscope: Error Fix and UI modification
- #2841 (closed), #2857 (merged): Update the MusicKeyboard widget on maximizing.
- #2874 (merged): Enhanced the UI of search-bar and its suggestions
- #2903 (open): pitchdrum-mapper: Implement Stop Functionality and enhance UI

sugarlabs/musicblocks-v4-lib:

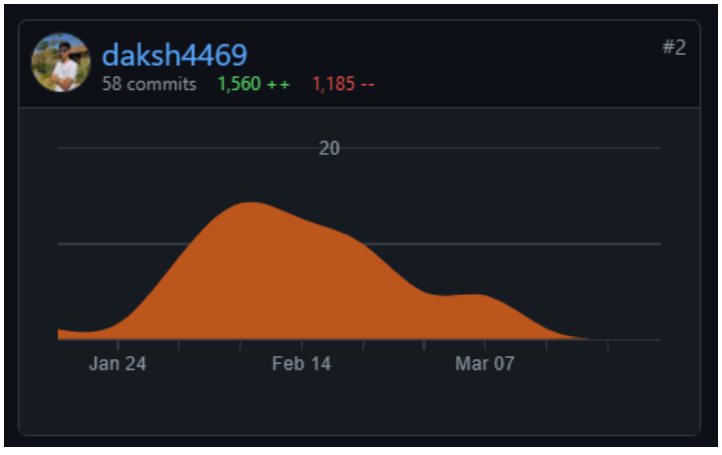
- #27 (closed but changes helped in #28) : Port musicutils.py to Typescript.
- #29 (merged) : Add testcases to musicutils.test.ts
- #33 (closed): Update Scale section in README.md
- #34 (open) #49 (merged): Add documentation for musicUtils.ts
- #42 (merged): Add error testcases to musicutils.test.ts

Issues:

- #2872 (closed): Blocks lose their color on hovering over them

Commits:

While contributing to Sugar Labs, I have made a total of 61 commits (till the date of writing). All of these commits can be found here.



(Contribution Statistics of sugarlabs/musicblocks repository from Jan 17 - Mar 27)

Project Details:

Title: Music Blocks Project Blocks Reorganization

Coding Mentors: Anindya Kundu, Walter Bender

Assisting Mentors: Peace Ojemeh, Devin Ulibarri

Music Blocks is being refactored to musicblocks-v4. This project aims to address the issues related to the presentation of the blocks in the canvas in the current version of musicblocks(version 3) and build upon them to design and implement the blocks' organization in musicblocks-v4.

As Music Blocks is going through a complete overhaul, blocks graphics and the overall UI is also going to change for good. Hence, this project might need some coordination with other projects i.e., blocks graphics refactoring, music blocks menus and palettes, and music blocks debugging aids.

Project Tasklist:

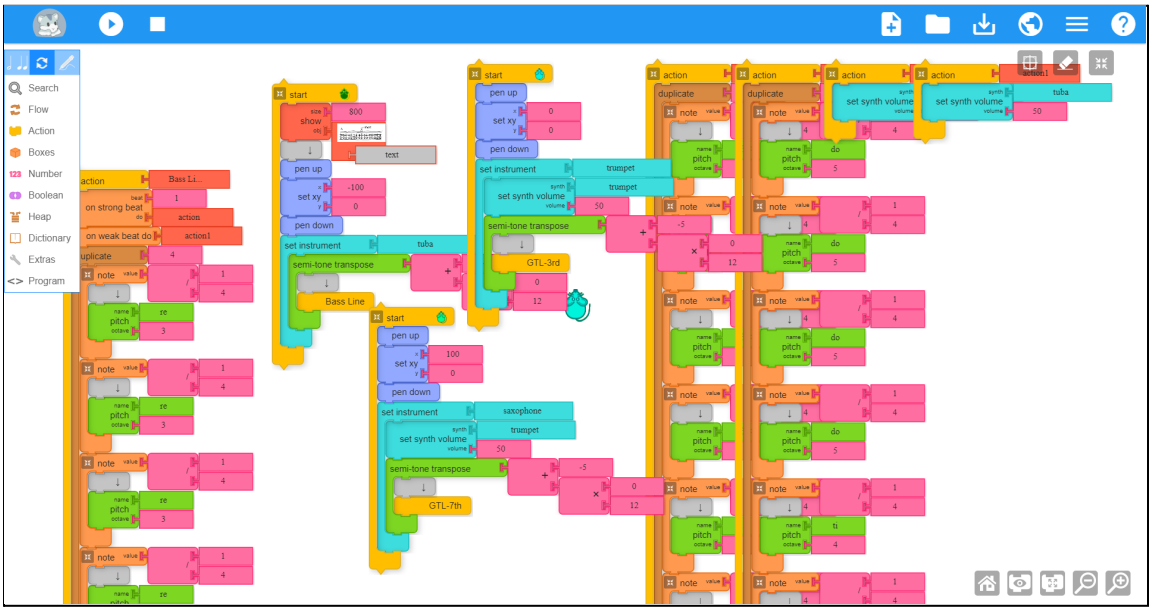
- Familiarize yourself with the current implementation.
- Come up with a framework for better project structuring.
- Design a flexible guidance layer on the top of the canvas.
- Implement the above in React (Typescript).

The Problem:

In the current version of musicblocks, there exist numerous issues concerning the presentation and organization of the blocks on the canvas. These issues are as follows:

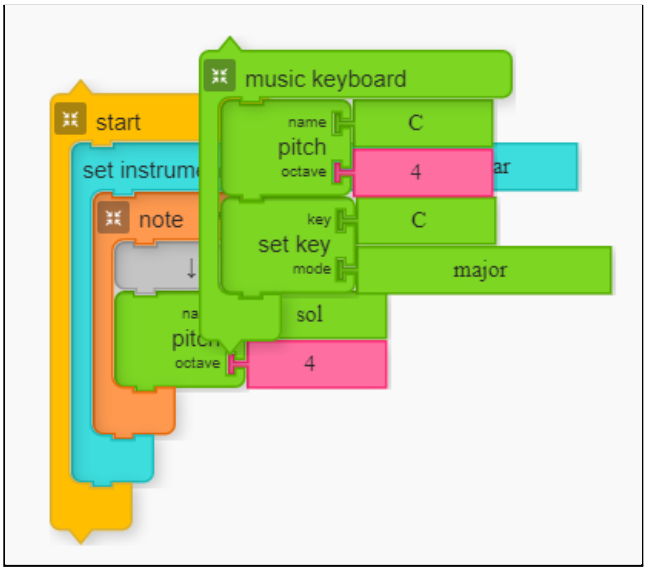
1. Currently, a user can arbitrarily place any block in any position, and thus, it gets quite **overwhelmingly crowded** when a considerable amount of blocks are present in a project. This gets quite noticeable and hampers the project management experience of a user.

An example project where this is visible is shown below (from Planet > Global):



(Project Reference: 12 Bar Blues with a Guitar Tab)

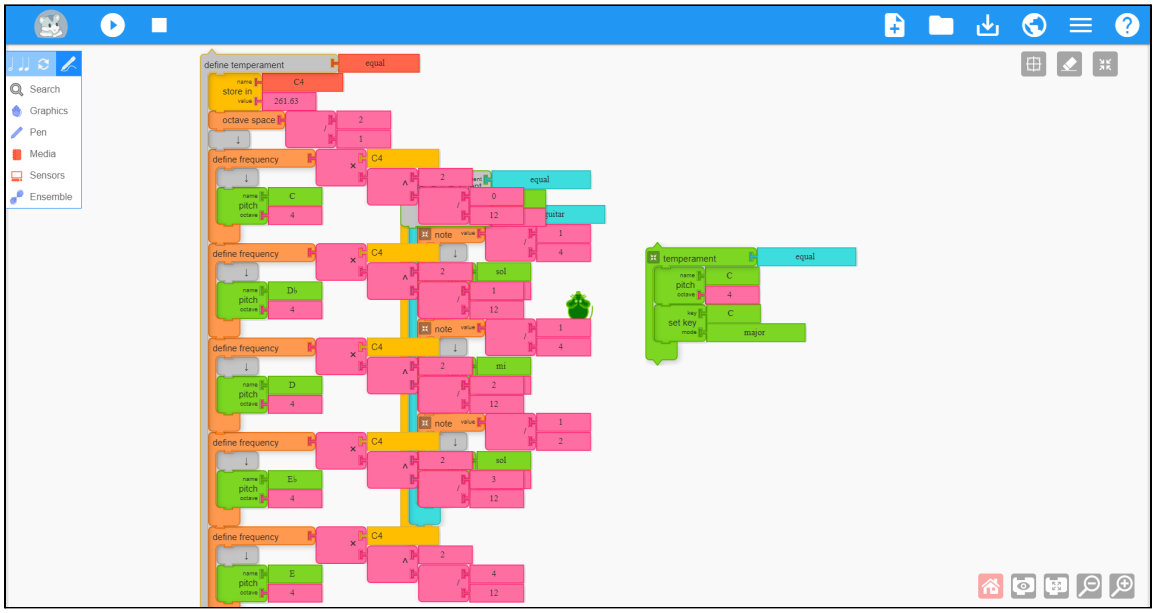
2. The **blocks can overlap each other** without any constraints whatsoever. In the current version of musicblocks, the block or block-stack, that a user has been more recently engaged with, overlaps the other blocks around it. A simple example depicting this is shown below:



(Here, the **music keyboard** widget block overlaps the **start** block as it has been more recently worked with.)

3. There is a feature in musicblocks to 'clamp' some blocks. However, this feature is only present in some of the blocks. The issue of overlapping blocks affects these **clamped blocks** heavily as they can become **almost entirely invisible** to a user if it gets overlapped by another block or block-stack.

4. As mentioned above, the feature to clamp is only available in some of the blocks. Often, the **width of a block-stack** can get congesting, and sometimes, there is no option to inline-clamp these blocks. This results in large parts of the blocks in the project getting overlapped by a single block-stack. A simple example representing this is generating the action block through the temperament widget. This is shown in the below image:



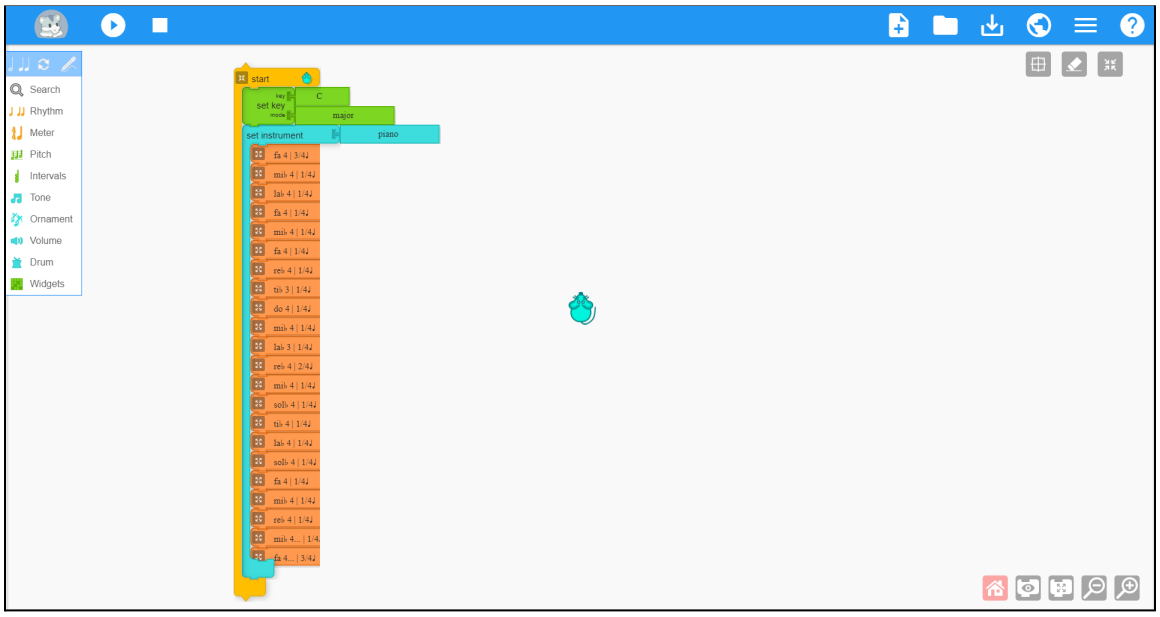
(In this musicblocks project, the **start** block and the **set temperament** block are not visible due to overlapping by the **define temperament** block, majorly due to its large width.)

5. Adjusting the **placements of the blocks can be quite tedious**. If the project is quite large, complex, and utilizing many blocks, the task of dragging a block/block-stack through the entire project can be pretty cumbersome.

The Solution:

The idea is to introduce **swim lanes** to guide a user to encourage better project management. Swim lanes can help a user arrange the blocks in columns and help **better project management** and much cleaner projects. I believe that a user should have a significant role in choosing how he/she wants to organize the blocks as each project and each user is different. Thus, this utility should be optional for a user to choose as, most likely, this feature would be much more helpful in more extensive and more complex projects than in beginner projects. We can include this as an **Advanced Mode or High Shelf feature** (depends on the new UI/UX). This utility has the following deliverables:

- **Provide the user the choice to use this feature or not:**
  A user can choose to utilize the feature and can also disable(stop using) it. This feature is ***more likely to be used in more extensive and complex projects*** than in small beginner projects. For example: In the below-shown image of a musicblocks project, there is no such need to organize the blocks in swim-lanes. It uses just a single block-stack that can be managed comfortably with the standard organization.



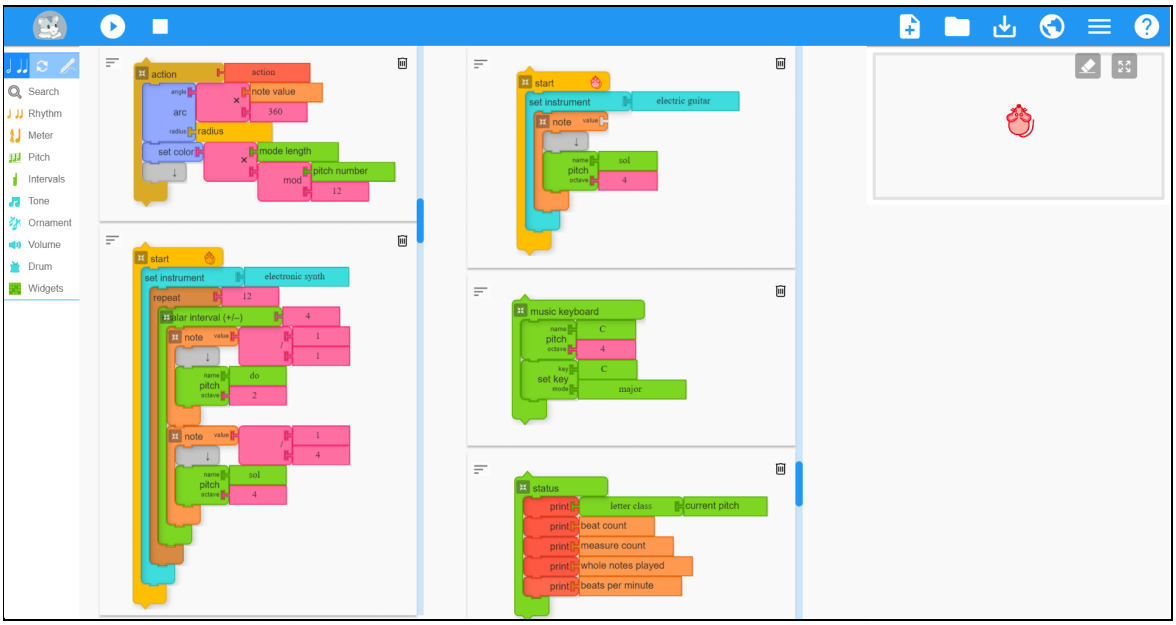
(Project: SandyChopinBerceuse *from* Planet > Global)

- **Set the number of swim-lanes:**

  If the feature is enabled, a user can set the number of swim-lanes according to the project. However, there may be an ***upper limit to the number of swim-lanes***, which can be calculated by the maximum width of a block/block-stack present in the project. Although, we can also pre-define this upper limit but this would often not compensate for wide block-stacks. We can give a user two ways to add swimlanes, either he/she can input the number of swimlanes required upfront or manually add/remove them. I think taking input and generating swimlanes automatically would make a user's life much easier.

- **Individually scrollable swim-lanes:**

  One potential advantage that swim-lanes can provide is the freedom to individually manage each swim-lane. Each swim-lane can be scrollable which gives the user the freedom to manage individual swim-lanes intensively rather than scrolling through the whole project altogether. ***This resolves the tedious job of dragging*** a block/block-stack through the whole project. In the case of individually scrollable swim-lanes, a user can simply drag a block/block-stack from one lane to another without holding and dragging it through the entire canvas. An inceptive **prototype** of implementing swim-lanes:

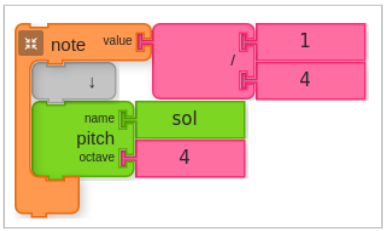


(Note: The above design is made considering the current block design, and could be modified after refactoring the block design in musicblocks-v4. This also does not demonstrate the top-level blocks specific swimlane mentioned in points below.)

- **Collapse the turtle stage:**
  One aspect to keep in mind while developing the swim-lanes is the placement of the mouse in the canvas. In the presence of swim-lanes, the mouse could adequately be placed using the *Collapse* feature. ***Also, on clicking the Play button, the swim-lanes would hide just like the blocks in the current version of musicblocks.***

- A significant aspect in developing swim-lanes is each swim-lane's width since every swim-lane will have a unique set of blocks or block-stacks. The upper-bound to the number of swim-lanes is also determined by the width of block-stacks. Thus, large widths can result in a low number of swim-lanes. One way of handling this is by providing the user **the option to set the widths**(probably by dragging) of each swim-lane. However, this would be a tedious job for a user when reorganizing project blocks. Another way of handling this is by **adding the "inline-clamp" functionality** to some more blocks. In the current version of musicblocks, there are only three **INLINECOLLAPSIBLES** blocks:

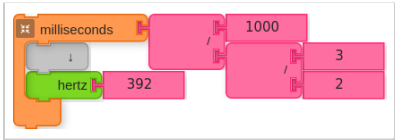
  - newnote (note value block)

  - interval (scalar interval (+/-) block)

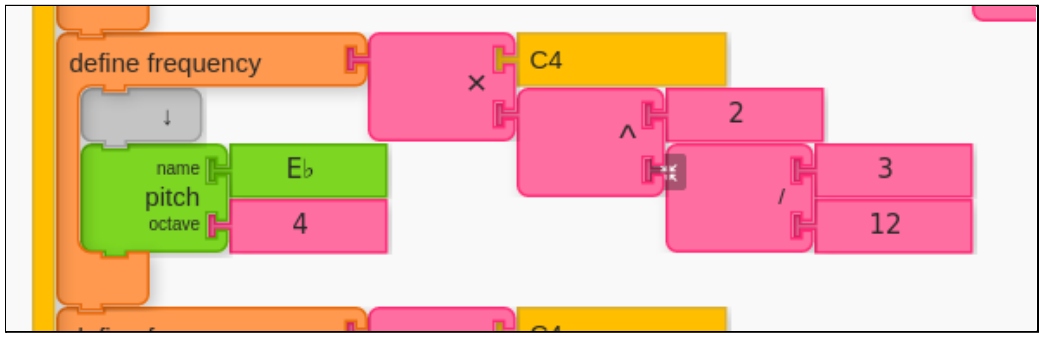
  - osctime (milliseconds block)

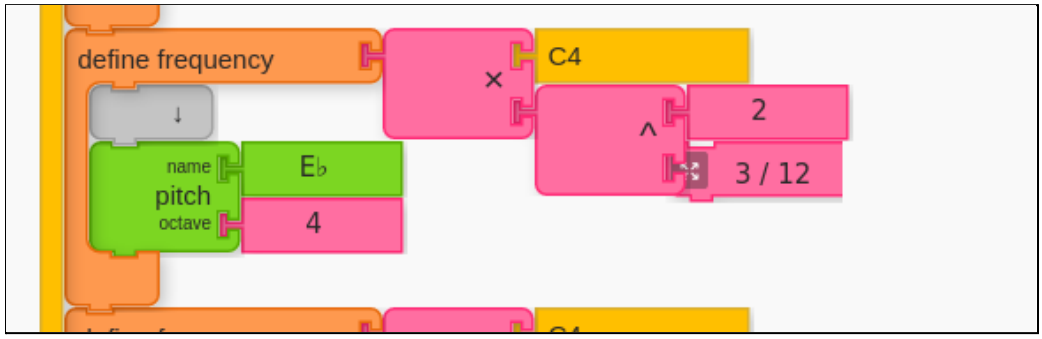
  However, in most of the cases (including the Define Temperament one mentioned on Page 10), the width of a block-stack is increased due to nested number-blocks (mainly add(+), subtract(-), multiply(x), divide(/), mod and power(^)).

I think we should **make these number blocks inline-collapsable**. This would help reduce the overall width of a block-stack and thus, fewer cases of overlap in the regular block organization. A very basic implementation of this would involve representing the answer, after operation between the connected numbers, as the name of the clamped block.

Before clamping the divide block:



After clamping the divide block:



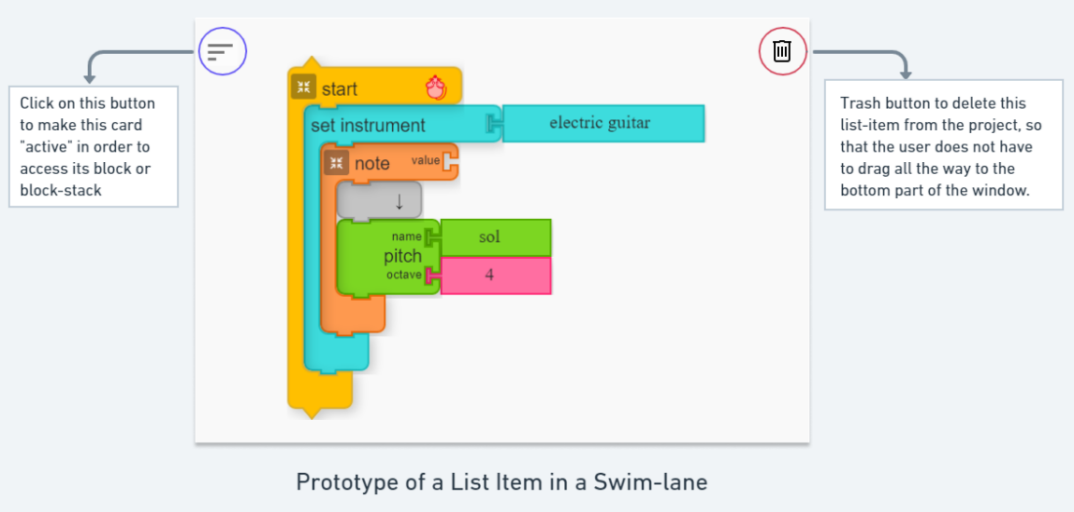
Above shown is a very initial example of this idea. The label "3 / 12" can be calculated using the following function in the *block.js* file of the current MB.

```
// function to allot the text to be presented on clamping the "divide" block
_divideLabel() {
    const number1 = this.connections[1]; // first block connected to the divide ("/") block
    const number2 = this.connections[2]; // first block connected to the divide ("/") block
    const num = this.blocks.blockList[number1].value; // value of the first block
    const den = this.blocks.blockList[number2].value; // value of the second block

    //by default, this.collpaseText.text = "divide";
    if(number1!=null && number2!=null) {
        this.collapseText.text = num + " / " + den; //text to be appeared on clamping
    }
}
```

Similarly, if all the number blocks are made inline-collapsible, widths of the block-stacks can be reduced significantly. Value of these number blocks is already being stored as the *value* field of a block in the *blocklist* in the current version of musicblocks. Thus, this idea would work similarly in case of *nested number blocks*.

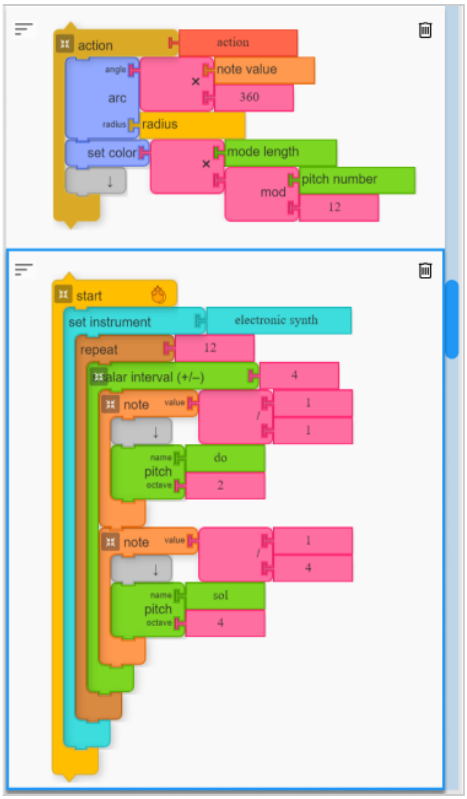
- **Transfer of blocks/block-stacks:**
    - In the process of block/block-stack transfer, an important point to note is that **every card would contain a single block/block-stack**. This also helps in **solving the overlap issue**, as now, on dropping ("onmouseup") event, the block/block-stack will get positioned in the swim-lane accordingly, without overlapping.



Prototype of a List Item in a Swim-lane

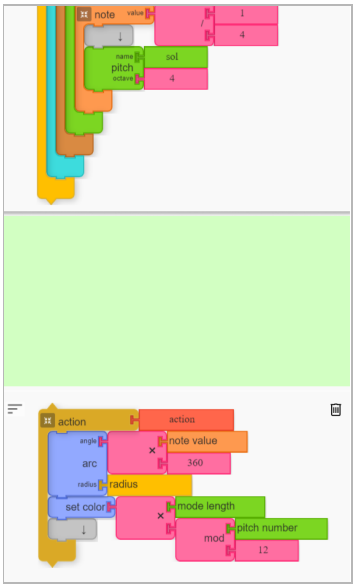
    - **To transfer an entire block/block-stack of a card**, the user can simply drag and drop this card to the desired position.
    - In case a user wants **to transfer a part of block/block-stack from one card to another**, the process is as follows:
        - **Click on the left button of the card to make this card "active" so as to access the block/block-stack of the card.** The active card has a blue outline to indicate that it is active to the user. Note that only one card can be active at any time.

- **Once the card is "active", the desired part of the block/block-stack can be dragged around the swimlanes.** The card over which this dragged block/block-stack is hovered gets "activated" and hence, this block/block-stack can be dropped in this "activated" card in the user's desired position(inside this card's block-stack). If it is dropped outside the block/block-stack of this "activated" card, it gets returned to its original card to demonstrate that one card can only hold a single block/block-stack.



Demonstration of an "active" card. In this image, the start block is "active".

- **To transfer a part of block/block-stack(from an "active" block) from an existing card to a new card**, a new card would be created for this removed block/block-stack at the "onmouseup" or drop event. This should function just as any other block transfer by drag and drop. Before dropping the block/block-stack in the latter case(of moving out of a block), a card outline/foreshadow between already existing cards can be shown to indicate this functionality, as shown below:

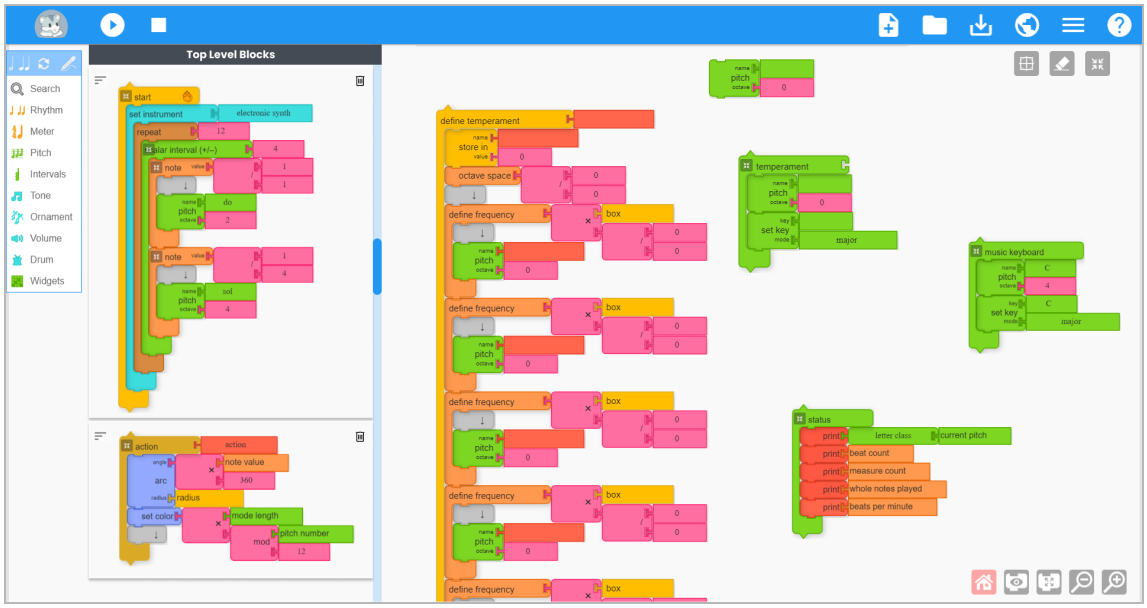


Overlay/Foreshadow shown on block/block-stack hover as an indication for the card of incoming block/block-stack.

- **To transfer a block/block-stack from a card to the general space of the canvas, i.e., the workspace region with no swimlanes (if present)**, this would be achieved using the conventional method of drag and drop of the whole card, and its card would be deleted immediately on dropping it to the canvas.

- The **Trash functionality** would also be needed to manage in presence of swim-lanes as currently, one has to drag a block/block-stack to the bottom of the screen and hover over the displayed trash icon. However, since the blocks/block-stacks are presented as list items in a swim-lane, it would be **preferable to add a Trash icon for every card/list-item in a swimlane**(just as in any ToDo list).

- **The action and start blocks can be managed distinguishably as they are eventually responsible for execution of the program in any project.** Hence, these can be represented in their own swim-lane of "Top-Level Blocks" so as to enhance the user's experience in managing these two blocks and all the other blocks separately. **The swimlane of these two blocks can be presented at the leftmost position.** A good use case of this utility would be when a user just wants these two blocks in a swim-lane and wants to manage all other blocks as usual. To further provide customizability, a user can opt for this specific swimlane or can choose to use custom swimlanes as per choice. A prototype showing this is given below:



*A working prototype of this wireframe demonstrating just the reorganization of top-level blocks can be found **here**.*

On disabling this feature, the canvas is re-rendered with the blocks/block-stacks(belonging to swimlanes) at new positions calculated by their positions in the swimlane layout and the position of their swimlane. Their positions in respective swimlanes would calculate these positions as their index(position) in the swimlane would be known. *Each swimlane would get "opened"* so that the positions would be calculated considering the length of the swimlane, i.e. the blocks in the lower part(lower than the window's height) of the swimlane would be positioned accordingly and would be accessed by scrolling through the window(as in current implementation). **Thus, on disabling this feature also, the user would have a much cleaner project(without overlaps and more white space) than before using/enabling this feature. This also can be viewed here in a video.**

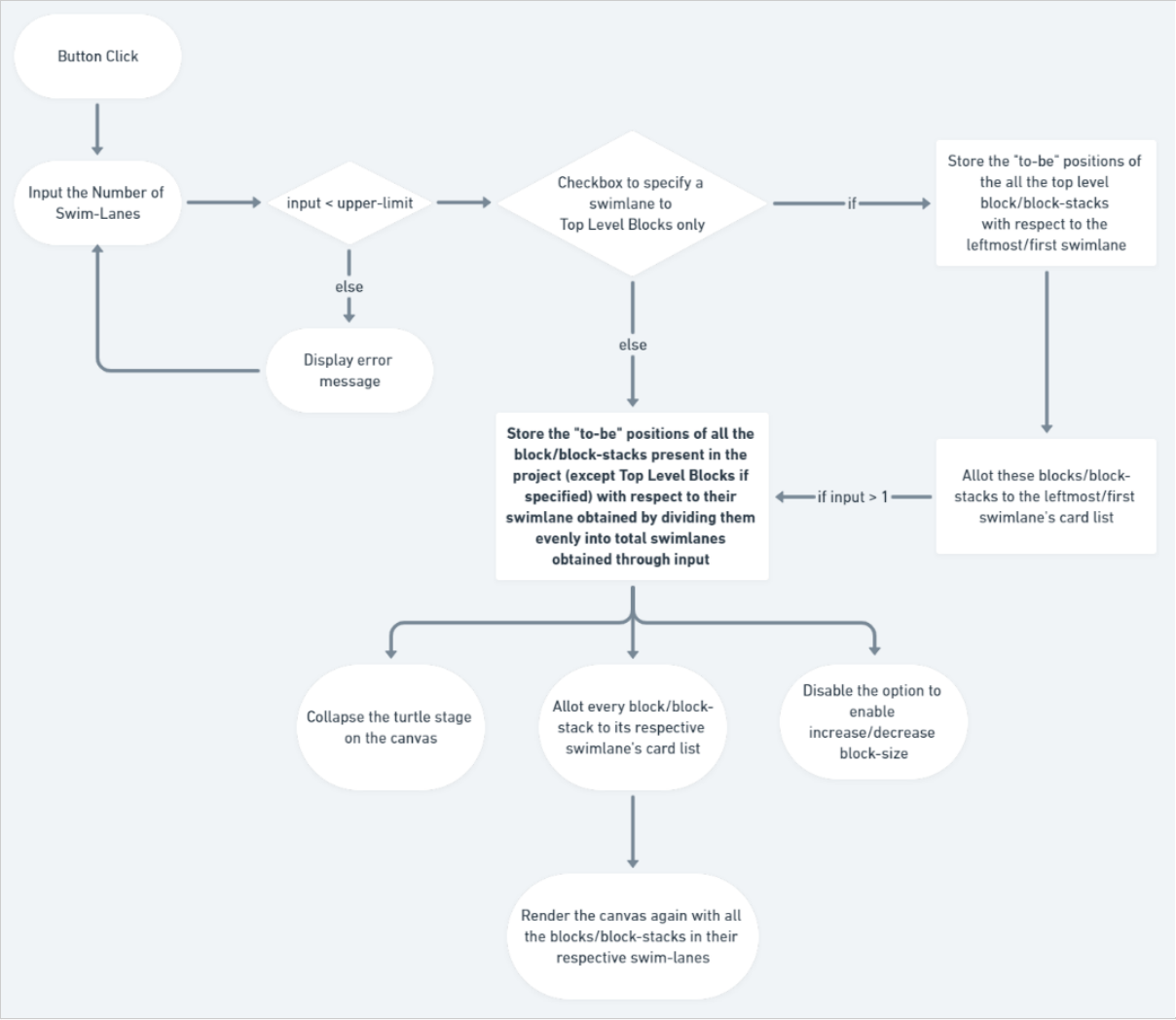
Summary of the Proposed Solution:

1. Add a feature (in advanced mode/high-shelf) to **organize the blocks in swim-lanes.** (*optional for user to enable/disable*)

2. **Every swim-lane will be scrollable**, hence, providing the user the advantage of managing each set of block/block-stacks individually.

3. **The user can set the number of swim-lanes** up to an upper limit based on the width of the block-stacks, and the turtle stage would be collapsed. Also, the user can opt to specify if he/she wants a swim-lane specific to Top-Level blocks (start and action blocks).

4. Below shown is the workflow activity diagram describing the process of rendering the swim-lanes organization of a project.



(***Workflow*** *of the proposed reorganization feature)*

Each block/block-stack will be arranged in a swim-lane as a draggable list item(card). Also, the turtle stage would need to be collapsed so as to incorporate these swim-lanes. One idea is that maybe, the collapsed turtle stage could appear by a pop-up.

5. **The order of blocks/block-stacks can be arranged in a swimlane by dragging and dropping the cards.** Any card can be transferred among swimlanes similarly by dragging and dropping the cards. **I have also implemented an initial basic version of swimlane, using react-dnd-beautiful npm package, which can be viewed here.**

6. **The transfer of blocks is achieved by the concept of "active" cards.** The cards of which the block/block-stack need to be accessed can be activated by the left button of the card. Once the card is active, its block/block-stack can be transferred by dragging and dropping.

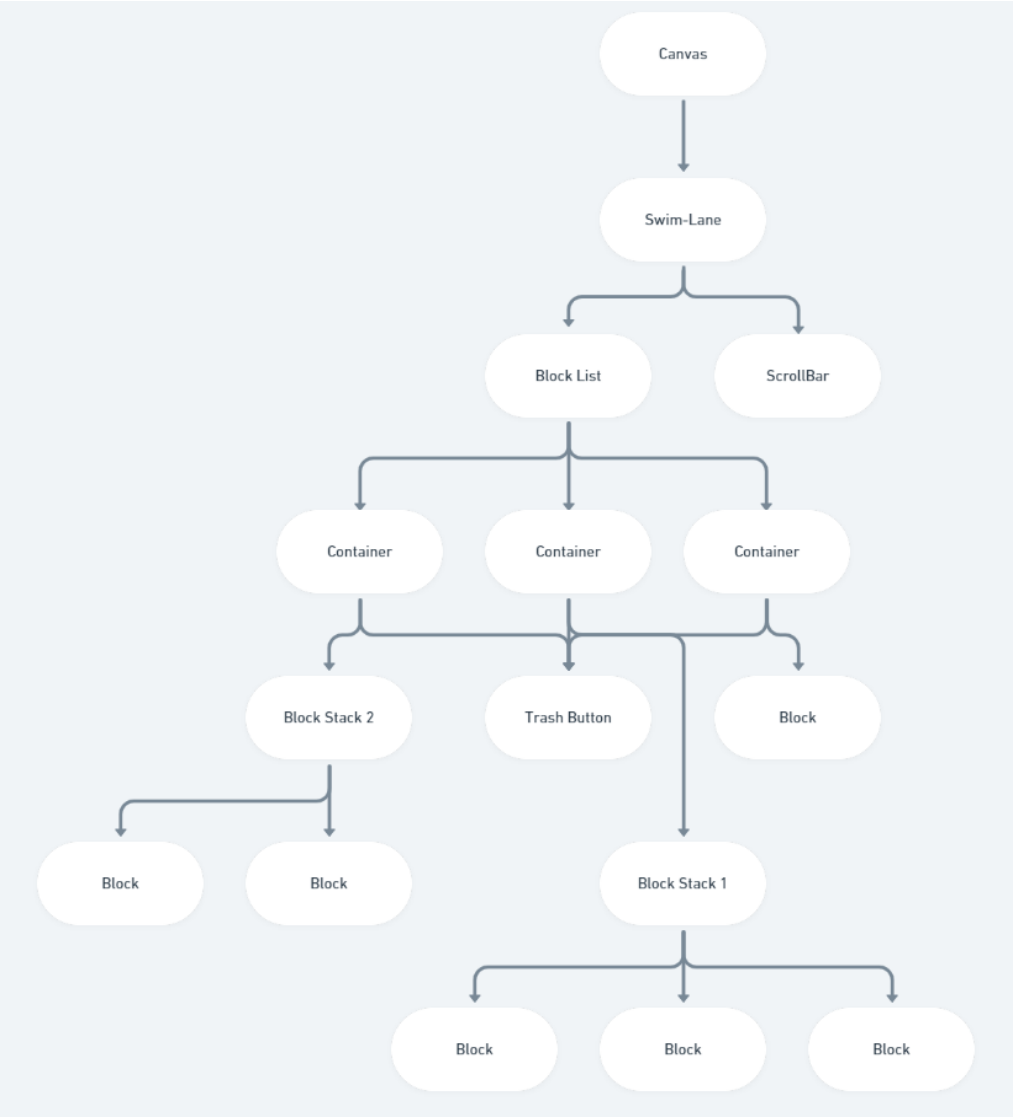
7. **A block/block-stack can also be added by "click" in palette menus. In this case, it will be rendered on the top of the active swim-lane.**

8. **On disabling this feature, the canvas is re-rendered with a much cleaner UI** of blocks with ample space between them and without overlaps, then before this feature is used. Each swimlane would get "opened".

9. Issues that may need handling:
    a. The feature of **increasing and decreasing the block size** while reorganization feature is enabled.
    b. The **width of certain block-stacks** may result in a decrease of the upper limit of the number of swim-lanes.
    c. A user should not be allowed to enable **horizontal-scroll** while using this feature.

10. Additional utility ideas:
    a. Some more blocks may be added to the **INLINECOLLAPSIBLES** blocks so as to control the width of a block-stack.
    b. **Resizable swim-lanes** by adjusting the width by dragging.
    c. Enable horizontal-scroll to **incorporate more swim-lanes without any constraints.**

Using React would be assistive in developing every list-item(card) as a component and every swimlane as an individual component utilizing the list-item's components. Every swimlane would be *mapped* in the canvas when the user opts to use this feature. The blocklist is each swim-lane can be managed using useState() and useEffect() hooks. Below is a sample component tree of a swim-lane in the canvas of musicblocks-v4.



(Component Tree of a single swim-lane in the canvas)

Links Mentioned:

1. Figma Prototype demonstrating reorganization of top-level blocks only.
2. Figma prototype of reorganization by swimlanes.
3. Code implementation of a basic swimlane with draggable cards.
4. Flowchart depicting the workflow of proposed reorganization feature.
5. Video demonstrating the UI improvement on disabling this feature.

Timeline:

Pre GSoC	• Familiarize myself with the current implementation. • Continue contributing to Sugar Labs.
Community Bonding (17th May - 7th June)	• Get to know the community better and bond with the mentors and developers. • Receive feedback on this project if something needs to be amended. • Explore the tech stack required for the project.
Week 1 (7th June - 14th June)	• Discuss and finalize the ideas to be implemented in this project. • Start working on the final UI/UX of this feature.
Week 2 (14th June - 21st June)	• Work out the final framework, workflow and the structure for this project.
Week 3 - 4 (21st June - 5th July)	• Decide the title and icon of this feature after discussion with the mentors. • Implement the allocation of blocks or block-stacks into their respective swim-lanes. • Add the utility to give the option to make a Top-Level blocks specific swimlane. • Setup the re-render of canvas and mode transitioning on enabling/disabling this feature.
Week 5 (5th July - 12th July)	• Review the work done with the mentors and make necessary amendments.
Evaluations | 12th July - 16th July	

Week 6-7
(16th July - 30th July)
	• Add the delete card functionality.
	• Setup the rearrangement of the list of cards in a swimlane.
	• Work on the transfer of block/block-stacks:
	  ○ Among various swimlanes
	  ○ Among different cards
	  ○ Addition/Removal of a card due to transfer of block/block-stack.
	  ○ Among the swimlane area and the usual area(region without swimlanes).

Week 8 (30th July - 6th August)	• Discuss and work on any new enhancements in UI or functionalities. • Add the necessary guide/documentation for this feature.
Week 9 (6th August - 9th August)	• Testing of the feature. • Improve the documentation.
Week 10 (9th August - 16th August)	• Cleanup and wrapping up the work. • One week buffer to compensate for any delay.
Final Evaluations | 16th August - 23rd August	

How many hours will you spend each week on your project?

My University End-Semester exams are scheduled from 19th to 25th May. Although this does not coincide with the coding phase, this may reduce my working time by 2-3 hours a day. My college summer vacations are scheduled to take place from 10th June to 2nd August, which is almost the whole of the coding phase. I will be able to devote around 40-45 hours a week efficiently. I have no other commitments for the summer vacations other than GSoC. So, I will be able to devote most of my time to GSoC. I am also free on weekends and will keep the community updated about my progress and maintain transparency about the project.

How will you report progress between evaluations?

I will be active on GitHub as I will be continuously working on the project while interacting with the mentors. Thus, my progress will always be reported thoroughly on GitHub. I am also planning to write weekly or fortnightly blogs about my progress in the project. I will be reachable anytime through IRC, Email, or a planned video session.

Discuss your post-GSoC plans. Will you continue contributing to Sugar Labs after GSoC ends?

After GSoC, I plan on continuing my contributions to Sugar Labs as I am amazed by the community relations and the work carried by this organization. I will contribute to the ongoing issues and the enhancements in the organization as there is always a scope of betterment on the web. I vision to hone my skills further and put them to use to give back to the community. I aim to develop mentorship skills and the ability to guide others and try to give back to the community by mentoring and guiding others. I hope to mentor future GSoC students.

I am looking forward to contributing to Sugar Labs this summer season.
Kind Regards.