Google Summer of Code

Sugar Labs

Music Blocks Project Menus and Palettes

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: 63 (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:

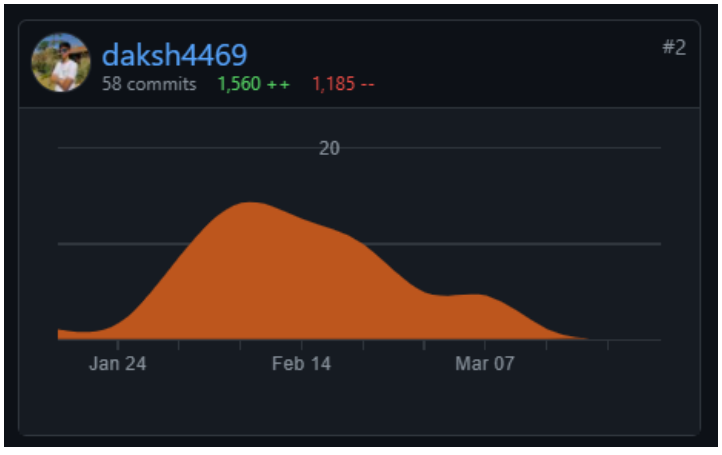
- #27 (closed but changes aided in #28) : Port musicutils.py to Typescript.
- #29 (merged) : Add testcases to musicutils.test.ts
- #33 (closed): Update Scale section in README.md
- #34 (closed) #49 (merged): Add documentation for musicUtils.ts
- #42 (merged): Add error testcases to musicutils.test.ts

Issues:

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

Commits:

While contributing to Sugar Labs, I have made a total of 63 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 Menus and Palettes

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 toolbars and block palettes in the current version of musicblocks(version 3) and improve upon them in designing and implementing them in musicblocks-v4.

Project Tasklist:

- Familiarize yourself with the current implementations of toolbars and palettes.
- Familiarize yourself with the Music Blocks v4 sketches wireframes.
- Come up with a framework for the new toolbars and palettes.
- Design the class structure for the toolbars.
- Design the class structure for the palettes.
- Implement all of the above in React(Typescript).

The Problem:

In the current version of musicblocks, there are some issues pertaining to the implementation and UI/UX of the menus and palettes. These issues can be tackled and improved upon in the new version of musicblocks. These issues are as follows:

1. A minor issue is that **some icons do not speak for themselves**. I think the icons should fit into the overall UI and the visual system of musicblocks.
   For example: The ***Display Statistics*** icon perfectly suits its functionality and is an excellent fit in the toolbar. Now, the ***Load Plugin*** icon does not fairly signify its utility in the project. **We can make such icons more expressive.**

   a. Display Statistics Icon b. Load Plugin Icon

2. **Heavy-Interdependence among files and non-modular components:**
   For example, in the current implementation, constants like DEFAULTPALETTE, MULTIPALETTES, PALETTEICONS, MULTIPALETTEICONS, and many more are declared inside a file (artwork.js for these constants in the current version) and are used in numerous other files. The same is the case for various methods.

   Hence, this increases the dependence of a component on other components and thus **it gets overwhelming to manage these components** and refactor them in case a new utility is to be added. Due to this issue, there persists another issue in the current musicblocks, that **the window needs to be reloaded if the mode is changed**(from beginner to advanced or vice-versa). This has been addressed earlier in #2094.

3. **Unoptimized state management:**
   a. It is difficult to **track the sequence of events** occurring as a result of a change in a state.
   b. Currently, the state of certain variables are passed as **global states** and are modified in different files. This can result in various anomalies as the state can get altered in various files.
   c. This also **hinders the synchronization and scheduling** of various events which involves changes to the state to be reflected in the components dependent on the same state.

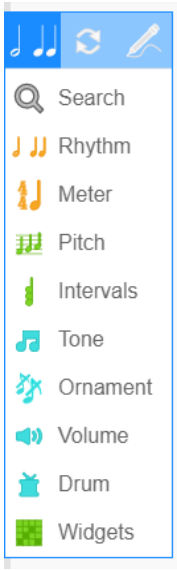
4. **Cluttered UI:**
   a. The current version's toolbar and palettes have too many buttons and features that a user can access. **While this is a utility to an experienced user, this can also confuse a new user who has just started building projects on musicblocks**. Many buttons and features can confuse a new user by giving too many options to choose from.
   b. The **floating buttons on the canvas** could also be better placed inside the toolbar itself.
   c. The **palette menus are significantly wider** and can be more compact.
   d. The **position of the search bar** does not suitably fit in the current palette menu bar.

Current Implementation:

In the process of contributing to musicblocks, **I am fairly familiarized with the current implementation of toolbar and palettes**.

- In the current implementation of the toolbar, in the class Toolbar, firstly a constructor is called which defines the color state of the STOP button, the language preference, and the tooltips of the buttons. The init() function initializes the strings and sets the tooltips of the buttons in the toolbar. Now, various methods(*renderWrapIcon()*, *renderModeSelectIcon()*, *renderPlanetIcon()*, etc.) are defined for rendering various components of the toolbar and setting up their event callbacks (such as "onclick"). Finally, there are two methods, *disableTooltips()* and *closeAuxToolbar()*, to disable the tooltips and handle the auxiliary toolbar respectively. **The class Toolbar is instantiated in the activity.js file and all of its methods are called in the same file**, which ultimately leads to the render of the toolbar in musicblocks.

- In the current implementation of palettes, **there are 3 classes defined, namely, Palettes, Palette, and PaletteModel**. The Palettes class is responsible for setting up various functionalities such as search, making the buttons, setting the blocks container, etc. Basically, it sets up the entire palettes menus, currently on the left of the musicblocks window, and this is also shown below:



- The Palettes class utilizes the Palette class which in turn utilizes the PaletteModel class. The Palette class adds the functionality to each palette like showing the menu of a palette, making blocks from a palette, making blocks of a palette from search results, etc. The PaletteModel sets the information of the blocks of a palette and maintains a list of SVGs for the blocks. **The whole implementation is carried out after the Palettes class is instantiated in the activity.js file.**

The Solution:

I have shared two wireframes designed on Figma, on GitHub discussions of the musicblocks-v4 repository. These wireframes can be found here and here. The design of the UI is yet to be finalized(on the date of writing) and so is the list of buttons and features that we want to be added in musicblocks-v4. Some buttons in the current implementation can be removed or replaced as per requirements. One such idea that I mentioned in this discussion was that maybe we can get rid of the **STOP** button and instead maintain its state and change PLAY into STOP only when a program is running. Other buttons can be reviewed in the same way.

- The first task in hand should be to settle the list of buttons present in the menu bars of the current musicblocks. Many buttons in the current version can be removed or replaced or can be placed inside a navigation pie-menu/dropdown. Currently, there are a total of 29 buttons overall in musicblocks menus. They are listed below:

Main Menu	Auxiliary Menu	Buttons floating on canvas
Play	Run Slowly	Grid
Stop	Run Step by Step	Clean
New Project	Merge with current Project	Collapse/Expand
Load Project from File	Turtle Wrap On/Off	Home
Save Project	Set Pitch Preview	Show/Hide Blocks

Planet	Toggle Javascript Editor	Expand/Collapse Blocks
Auxiliary Menu(toggle button)	Restore	Increase Block Size
Help	Mode Change	Decrease Block Size
	Select Language	
	Advanced Mode Specific Buttons: • Display Statistics • Load Plugin • Remove Plugin • Enable horizontal scrolling	

- **The buttons can also be distinguished into low-shelf and high-shelf buttons**, as mentioned by @walterbender (Walter Bender) in this comment. Low-shelf buttons would be including the buttons which a user would want to always be at hand (easily accessible) and high-shelf buttons would be including the buttons which would not be required on a regular basis and can be a few clicks away. **Redo and Undo buttons are introduced in place of the Restore button.** Given below is the categorization of all the buttons according to this classification:

Low Shelf	High Shelf
Play/Stop with a dropdown to: • Play • PlaySlowly • Play Step by Step	Turtle Wrap
Project with a dropdown to: • New Project • Save Project • Load Project	Plugin with a dropdown to: • Load Plugin • Remove Plugin
Save Project with a dropdown showing all formats.	Set Language
Undo	Enabling Horizontal Scroll
Redo	Merge with Current Project

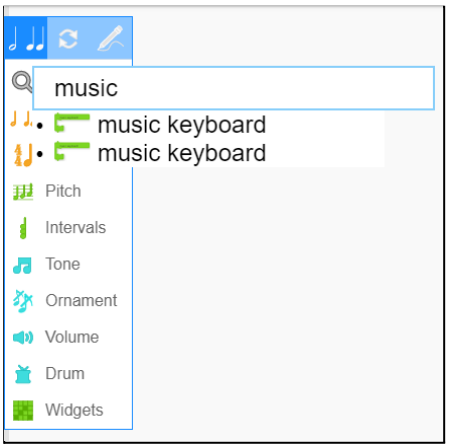
Increase/Decrease Block Size	Display Statistics
Clean	Mode Selection
Grid	Javascript Toggle Editor
Show/Hide Blocks	Set Pitch Preview
Expand/Collapse Blocks	

- There are a total of 19 palettes in musicblocks currently, as mentioned below:

Rhythm	Tone	Widgets	Number	Extras	Media
Meter	Ornament	Flow	Boolean	Program	Sensors
Pitch	Volume	Action	Heap	Graphics	Ensemble
Intervals	Drum	Boxes	Dictionary	Pen	

The objective is to combine some palettes by common and niche blocks so as to reduce the total number of palettes to allow a more compact UI and much more organized segregation of blocks. For example: The Ornaments palette includes a total of 4 blocks. This palette could possibly be merged with the Tone palette. To achieve a more compact UI, we can possibly make nested palettes. This means that two or three palettes may be nested inside one parent palette.

- Search-bar is a major utility to any user of music blocks. Currently, it is positioned inside the palette menus. **Search-bar would be better suited outside the palette menus as a global search-bar.** Currently, the search-bar shows two same results for some inputs as shown below:



The first music keyboard search result generates an empty music keyboard block(with no blocks nested inside of it) while the latter search result makes a default/predefined music keyboard block-stack(example). However, there is no such indication describing this. **This minor issue can be resolved just by adding any kind of assistive text such as "(Default)" or "(Example)" besides the default/predefined block-stack search results.**

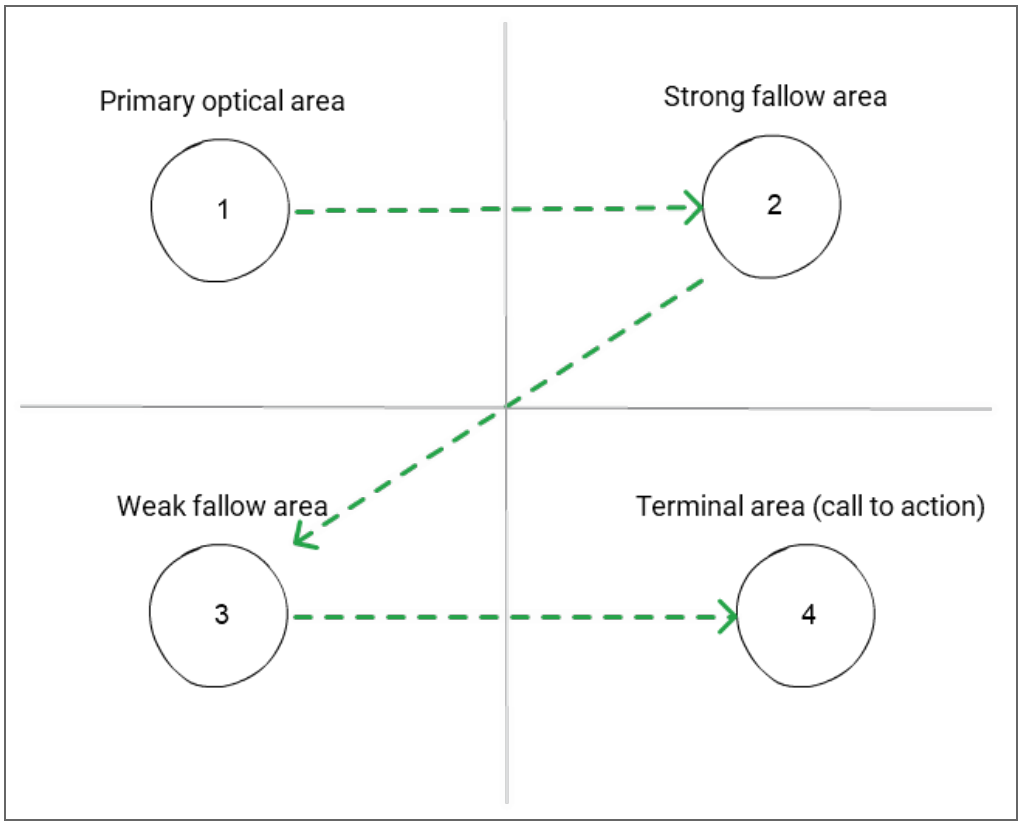
- **Our main focus should be on the architecture of the project.** As mentioned in various discussions, **we should modularize the components so as to differentiate the UI components and the functionality(non-UI) components of a button/feature**. The non-UI specific methods can be grouped in a module instead of defining them along with the UI-specific methods(which would be in UI modules) so that different UI modules can use them. This is also discussed here, where **the UI module part is called the "client" and the non-UI modules part is called the "engine".**

  The "engine" part is built in the sugarlabs/musicblocks-v4-lib repository. This is the core of the new musicblocks application managing scheduling along with implementation of certain algorithms to be exported to the client-end of the application. This also handles the inner state management and interfacing of musicblocks-v4. The music utilities for Audio state management have been developed.

  Using **React library enables comfortable modularization of code due to its component structure**. Modules can be utilized in various components with the help of *import* and *export* keywords. Each module would be responsible for certain functionality and this would also diminish the interdependence between files. This has several advantages such as:

  - Easier debugging and code navigation.
  - Makes tracking the origin of an issue easier.
  - Enhances the reusability of code.

- I would like to propose the following on the UI/UX front, based on the discussions on GitHub. **The UX has been designed following the *Gutenberg Diagram* which states that the typical attention flow of a user and ultimately his/her mouse movements follow a Z-shaped pattern.**



Gutenberg Diagram

- **All the low-shelf buttons are placed on the left-side of the toolbar along with the logo of Music Blocks.** This comprises the Primary Optical Area in the Gutenberg Diagram.

- **The Palette menu bar is also included in the Primary Optical Area**, as the usual attention of a user will go to this area, and thus it will ensure intuitive behaviour while accessing the palette menus. I think keeping the palette menu bar floating rather than connected to the left side on the canvas would make the UI more aesthetic. For enhancing the customizability, this palette menu bar can be made draggable. However, this would ultimately act as a "block" and a user would have one more "block"(of palette menu bar) to manage by dragging.

- **The Palette Menu bar is vertically oriented** so as to match the parallels from the individual vertical palette menus. **The Palette Menu would be expandable/collapsible** as a new user would want to access the palettes by names while an experienced user can work with them without expanding them. This customizability increases the workspace.

- **The Search icon is kept on the right side of the toolbar in the Strong Fallow area.** Unlike the current search bar, **the search bar appears as a modal box.** This is inspired from the Spotlight Search of macOS by Apple, and is suggested here in the discussion of wireframes.

- **The Search Modal Box initially displays the recently used blocks and on any input in the search bar, it displays the usual search results.**

- **The high-shelf buttons are majorly settings and are thus kept in the Settings Menu**(side navigation). Settings button lies to the right of the Search button. This Settings menu is a vertical menu presenting all the high shelf buttons in a compact UI.

- **The currently floating buttons on canvas are grouped together in a menu bar on the right.** These are all low-shelf buttons which a user can access any time. This menu bar lies somewhat around the Terminal Area and is placed vertically centered in the right half of the window. This allows the user to call to action by a single click whenever needed.

- **Optional Ideas:**

  - To increase the workspace further and provide more customizability to the user, **the canvas buttons menu can be collapsible/hidden.** However, this would somewhat compromise their utility of always being at hand. Also, note that the new version may contain a new button to Rearrange Blocks.

  - We can also **incorporate a "Status" button besides settings button which opens as side-navigation to show insights to executing programs** such as showing the current executing block and various debug/log statements. This would require coordination from Music Blocks Debugging Aids project.

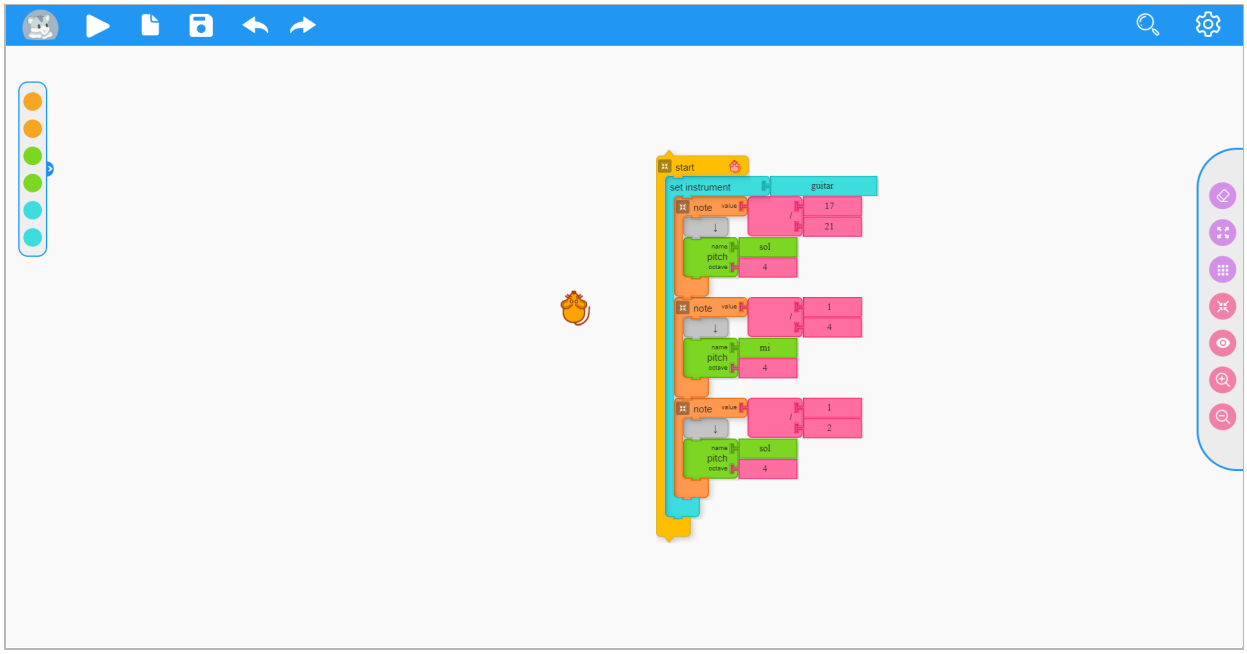
- For implementation, we can take some inspiration from the current implementation. This would involve Toolbar, Palettes, Palette, and PaletteModel as major components(name of these components can be modified). PaletteModel component would be imported in the Palette component and the Palette component would be imported in the Palettes component. Various Art Utilities would be imported from musicblocks-v4-lib. All the buttons can be treated as individual components to encourage code independence and easier managing.

UI/UX Prototypes:

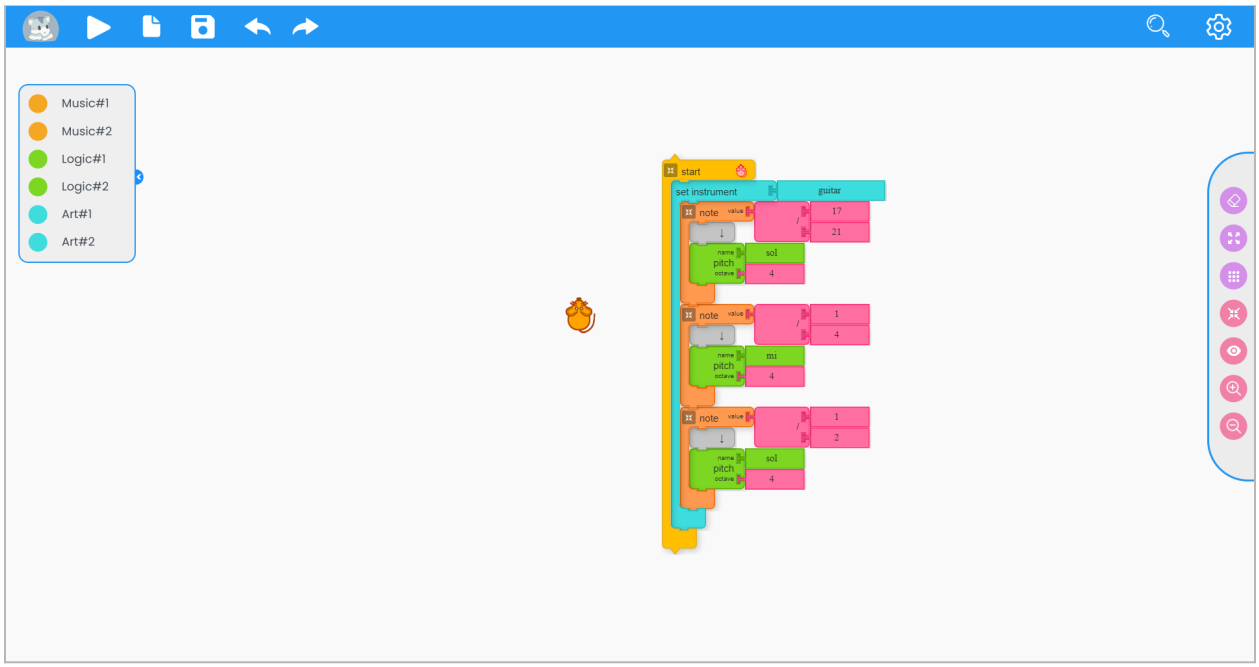
The prototype wireframes following the mentioned points are shown below: (considering the canvas buttons menu collapsible. Note that the blocks may be different eventually in Music Blocks v4.)

Color Scheme:

#2196F3, #444B54, #FF6A9E, #D97DF5, #3EDCDD, #7CD622, #F5A623, #F9F9F9

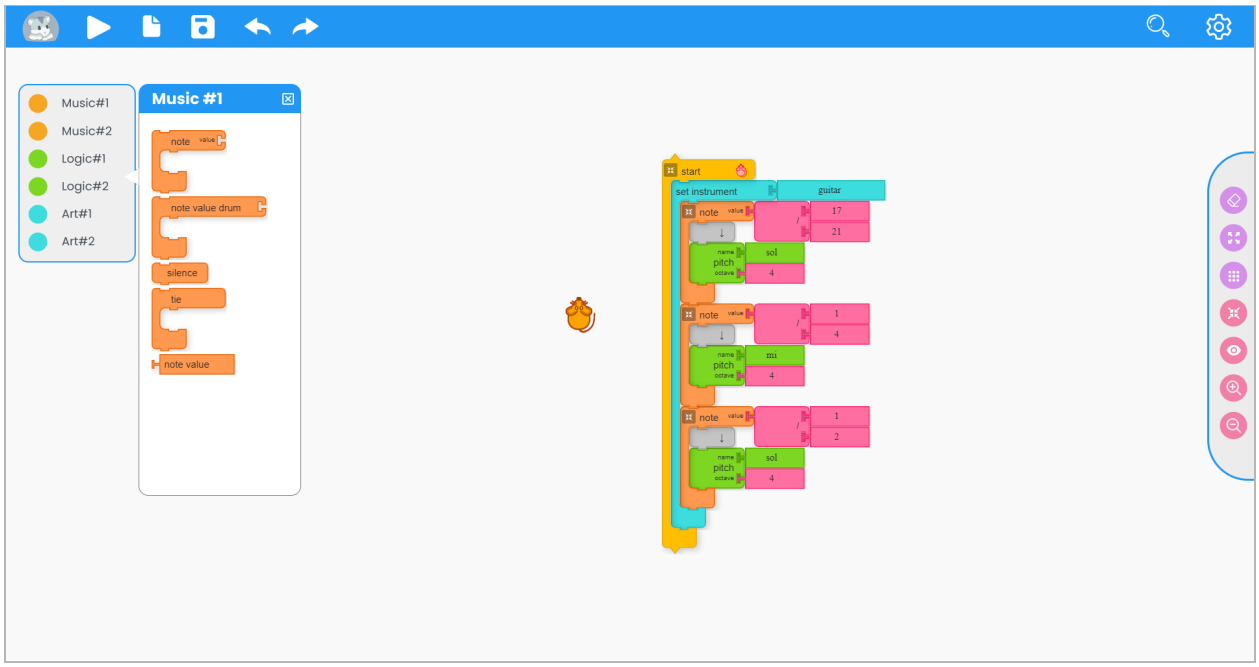


Home Window

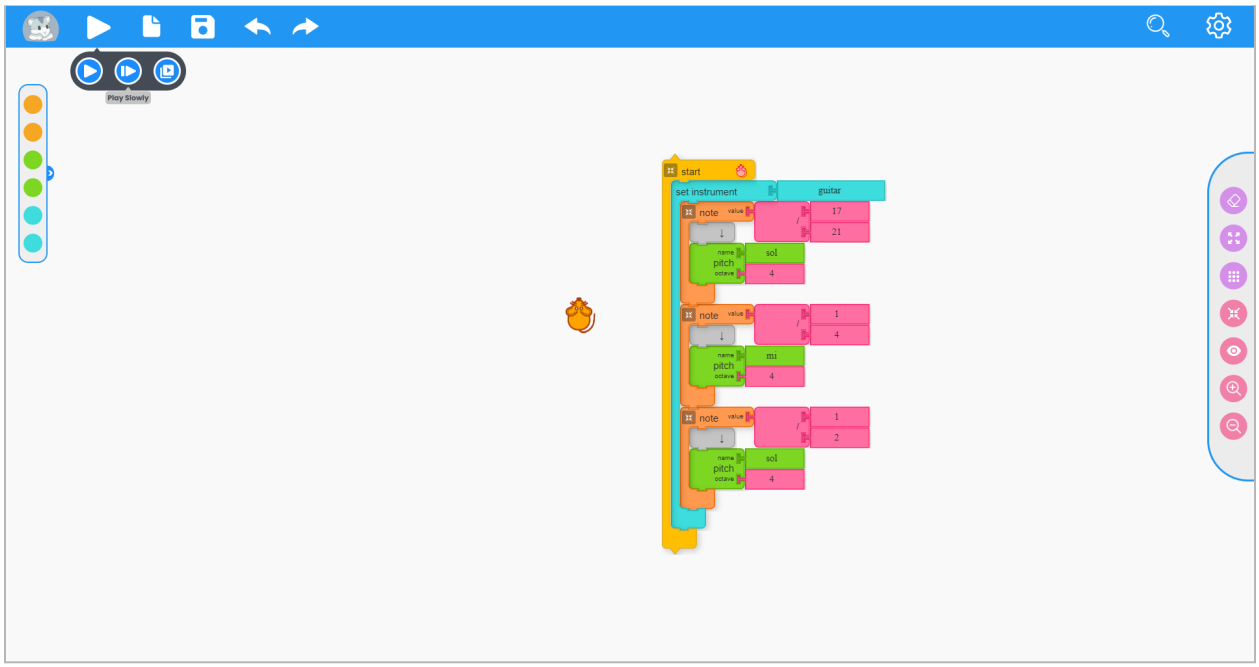


Expanded Palette Menu Bar

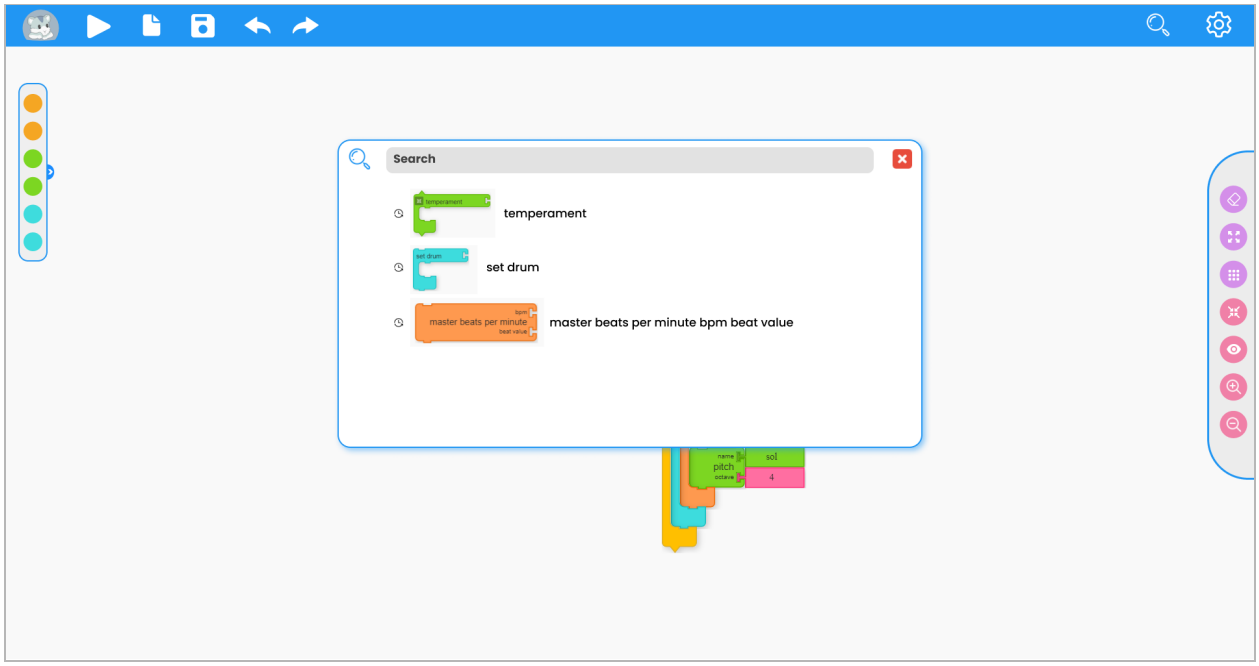
I have also developed a prototype demonstrating this expand/collapse feature of this expandable Palette Bar. This code sample can be found **here.**



Expanded Menu of Music#1 Palette

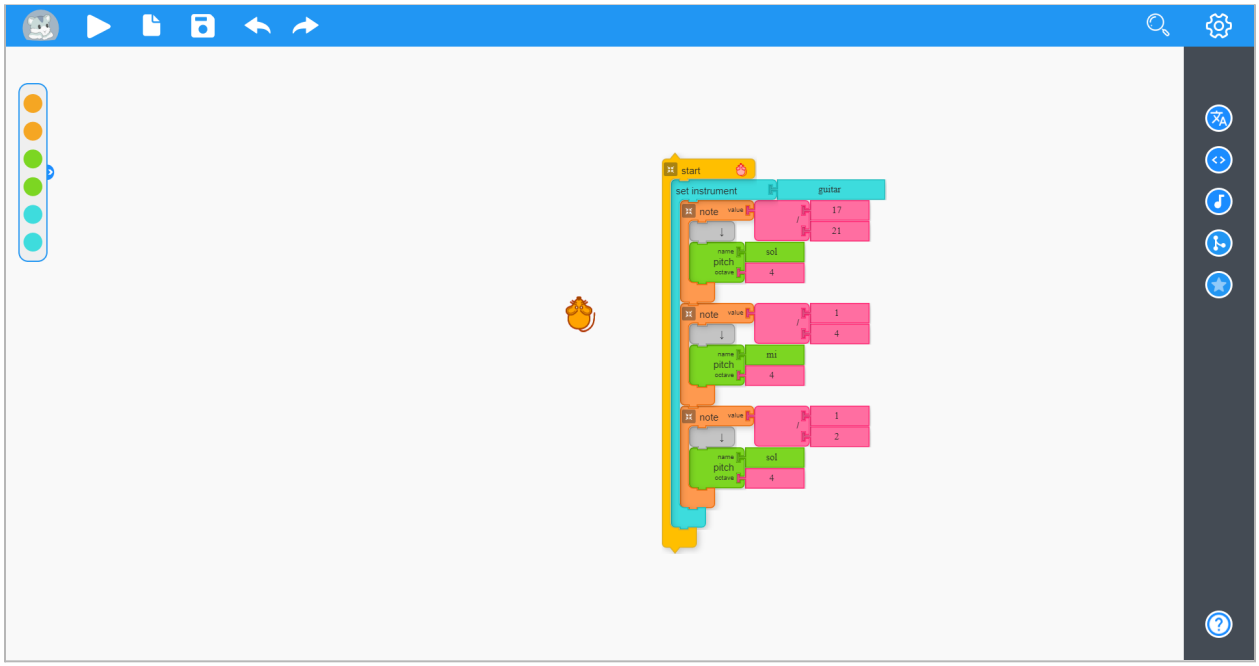


The Play button, on click, generates a dropdown menu containing the buttons: Play, Play Slowly, and Play Step by Step. The Project and Save buttons on its right also generate similar dropdowns.

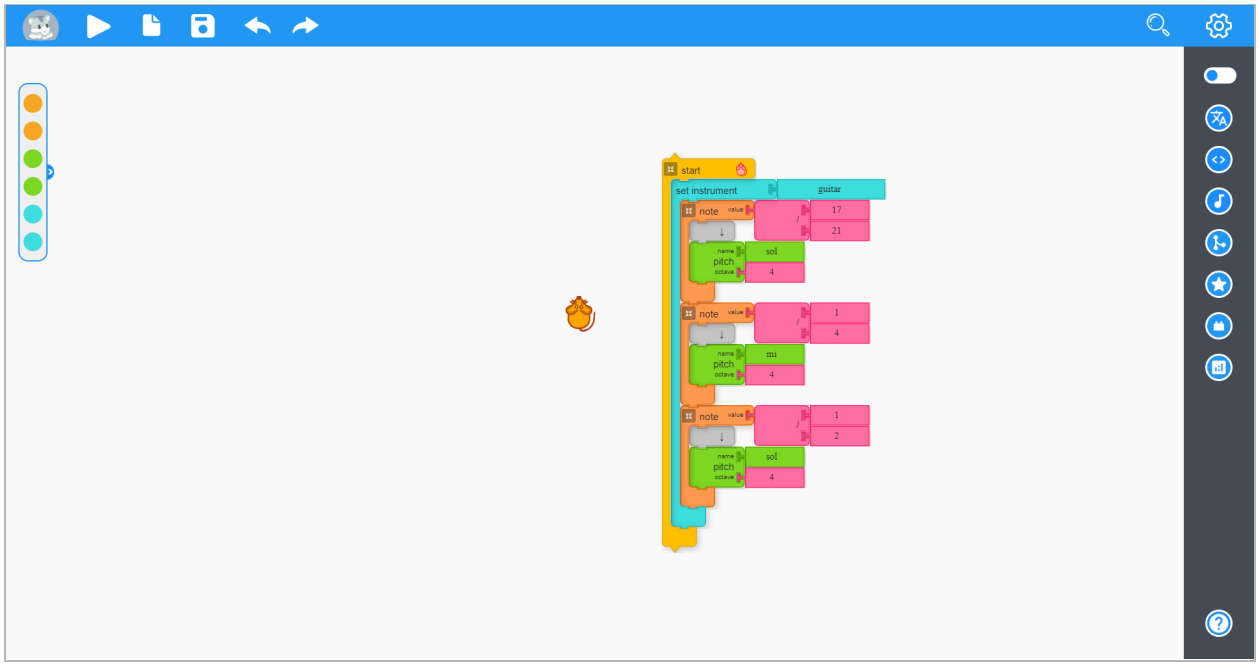


Search Modal Box showing the recently used blocks

A **SearchModal component** would be made for the modal box. It will **import components of SearchBar and RecentBlocks**. **The recent blocks functionality can be implemented using the queue data structure.** Just as a stack of blocks is maintained, and a block is popped on using the Restore feature, a queue of length up to a predefined number(maybe 5/6) can be maintained for storing the recently used blocks. We would be initially checking if the length of the queue has reached its maximum length or not. Whenever a block is used, first, we will check whether it is already present in the queue. If yes, then we bring this block to the top(queue's back); else we remove the block present at *queue.front()* and add this new block.



Settings Menu opened with Beginner Mode



Settings Menu opened with Advanced Mode enabling the features of horizontal scroll, displaying statistics and using plugins. These advanced mode specific buttons are conditionally rendered depending on the state of "mode" (beginnerMode in current implementation)

The color filled circles in the palettes menu would be containing the icons representing the palette(as presented in the canvas buttons menu bar in the above wireframes). **The wireframes are designed in Figma and can be found here for further inspection and interaction.** This is a working prototype and can be run by clicking the PLAY button on the top-right side of the Figma window and some of the UI can be then interacted with. You can also visit the prototype directly from **here**, and can watch the execution of this prototype in this **video**. This Figma project also contains wireframes for optional ideas like Status Section and Collapsible Canvas Buttons Menu.

Links mentioned:

1. Basic code implementation of expandable palette menu bar.
2. Figma prototype wireframes for the UI/UX of Music Blocks v4.
3. Video demonstrating the working of the prototype.
4. Github Discussions of various wireframe drafts.

Timeline:

Pre GSoC	• Brainstorm ideas regarding the project. • 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.
Week 1 (7th June - 14th June)	• Come to an agreement about the list of buttons in the toolbar. • Discuss the categorization of blocks into palettes and reorganize them.
Week 2 (14th June - 21st June)	• Finalize the icons and names of all the palettes and buttons. • Discuss and finalize the wireframes for toolbars and palettes.
Week 3 - 4 (21st June - 5th July)	• Set up the basic framework of the components. • Work on the implementation of the toolbars(menus). • Fix encountered bugs and improve code quality.
Week 5 (5th July - 12th July)	• Add the necessary documentation for the toolbar(menus) and its utilities. • Review the work done with the mentors and make necessary amendments.
Evaluations | 12th July - 16th July	
Week 6-7 (16th July - 30th July)	• Work on the implementation of block palettes menu bar. • Work on rendering the block menus of every palette. • Start working on the search bar and its functionality. • Add the feature to show recently used blocks and finalize the search functionality.
Week 8 (30th July - 6th August)	• Test the search functionality and make necessary changes. • Discuss with mentors and work on any additional enhancements needed.

	• Add the necessary documentation for palettes and their utilities.
Week 9 (6th August - 9th August)	• Check for any issues/regressions and solve them. • Improve the documentation.
Week 10 (9th August - 16th August)	• Cleanup and testing. • 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.