Google Summer of Code

Sugar Labs

Music Blocks Debugging Aids

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:

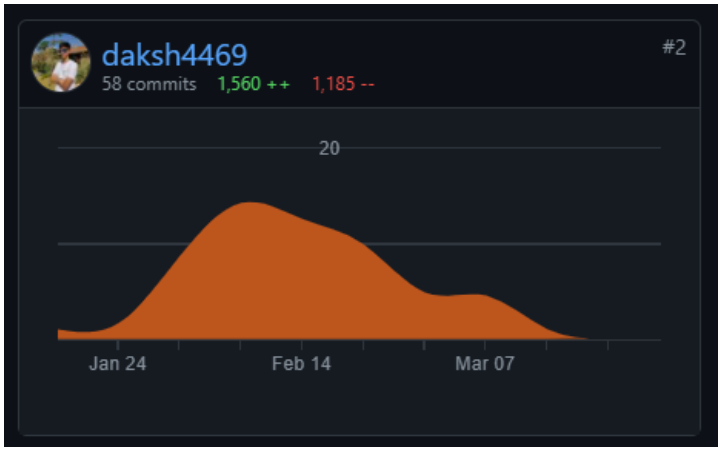
- #27 (closed but changes helped in #28) : Port musicutils.py to Typescript.
- #29 (merged) : Add testcases to musicutils.test.ts
- #33 (closed): Update Scale section in README.md
- #34 (open) #49 (merged): Add documentation for musicUtils.ts
- #42 (merged): Add error testcases to musicutils.test.ts

Issues:

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

Commits:

While contributing to Sugar Labs, I have made a total of 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 Debugging Aids

Coding Mentors: Anindya Kundu, Walter Bender

Assisting Mentors: Devin Ulibarri

Music Blocks is being refactored to musicblocks-v4. This project aims to develop the tools to enhance the debugging process. This will improve the User Experience for Music Blocks as handling various breakpoints, error detection, and program status visualization would be enhanced. This project will need coordination both with the MusicBlocks Block Graphics Refactoring project and the language interpreter.

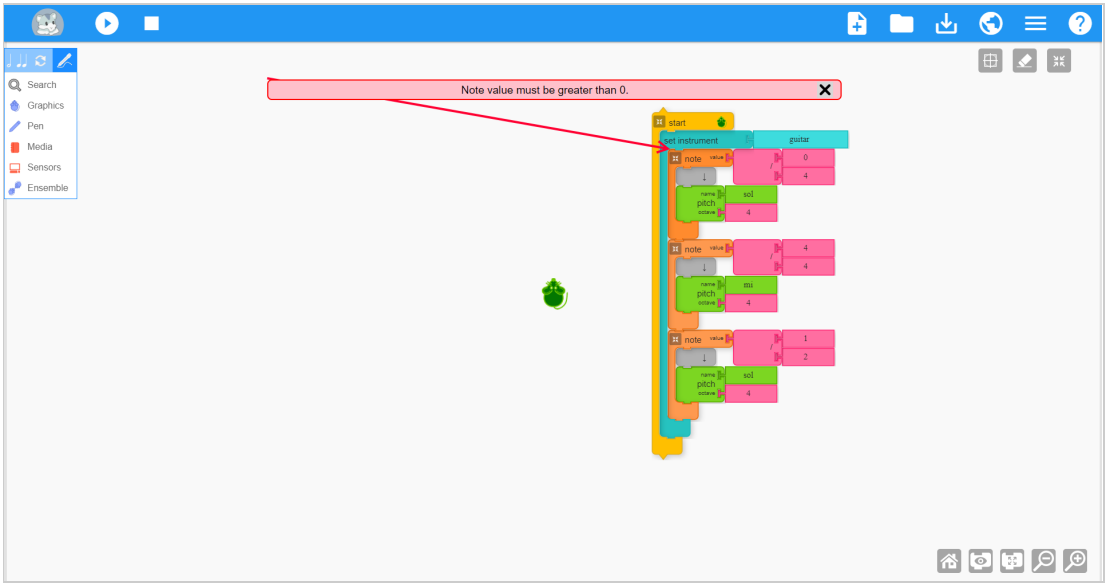
Project Tasklist:

- Familiarize with the current debugging features.
- Come up with a framework for how debugging might work for Music Blocks v4.
- Implement these ideas.

The Problem:

1. **The Positioning of Error/Warning Messages:**

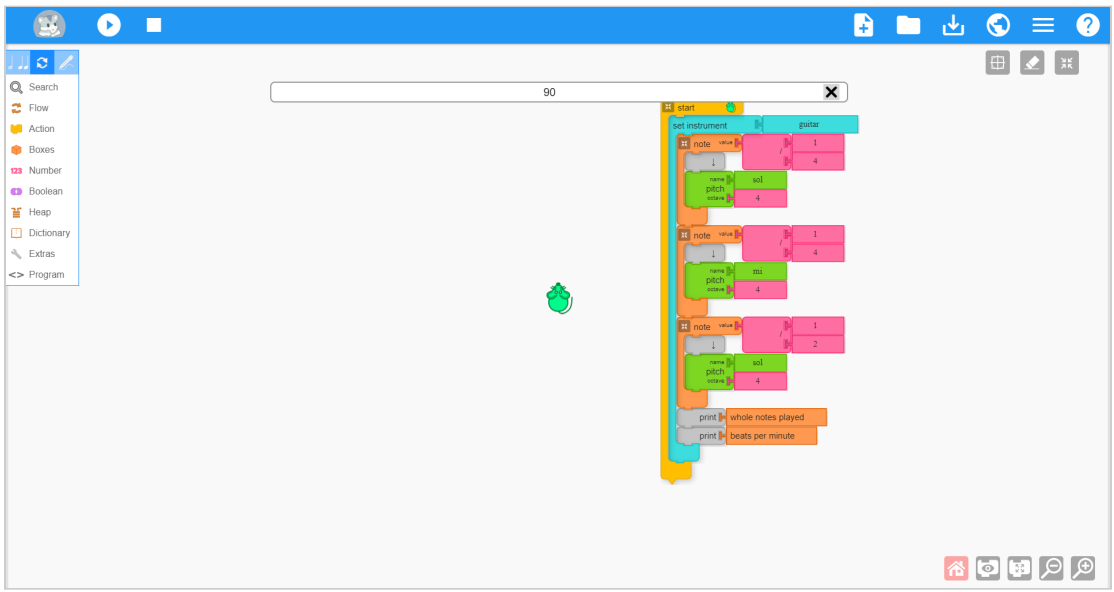
   In the current version of Music Blocks, the error and warning messages are shown on the upper-half/top of the window. So, in case, there are more than one errors caught in a program, the errors overlap, i.e. the new error overlaps the older one. Hence, different arrows pointing to different blocks are shown coming out of a single error. This can hinder the user experience, as a user cannot see old errors once the new ones are generated and thus can find it difficult to rectify the mistake in their projects.



Error Message Positioning

2. **The Positioning of Print/Comment Messages:**

The print and comment messages are generated through the print and comment blocks of the Extras Palette. These can be used to debug the project by printing or commenting relevant messages in between blocks so as to check out the source of error. They also are positioned on the upper-half of the window, just below the toolbar. The old messages get overlapped by new messages. Moreover, the error/warning messages and print/comment messages can overlap each other.



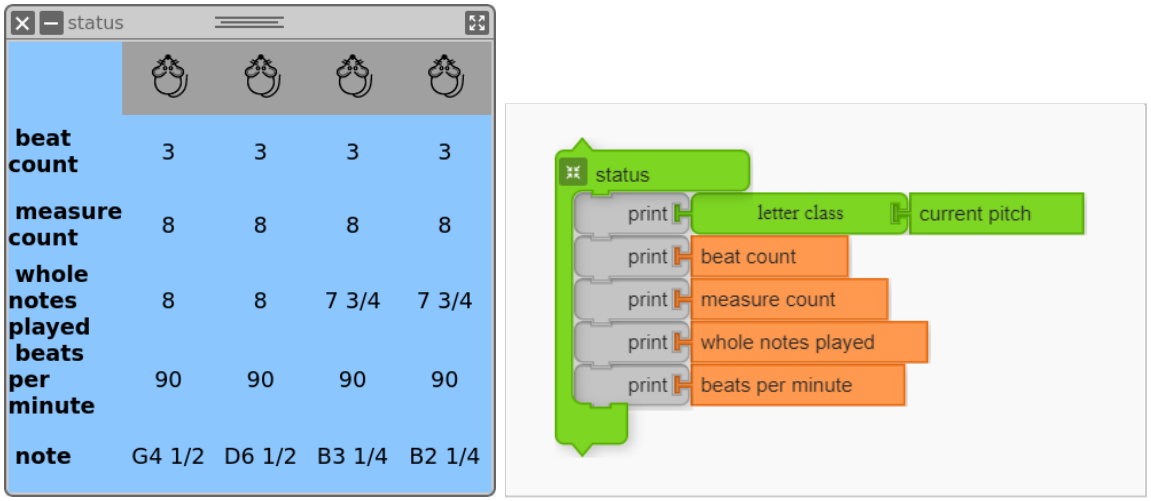
Print Message Positioning

3. **Show/Hide Blocks:**

The show and hide blocks of Extras Palette can be useful in small projects to set breakpoints in order to show a specific section of blocks(by show blocks) or hide a section of blocks(by hide blocks). This however, can be a tedious job in larger, more complex projects where there are many block-stacks present and thus, adding these show/hide blocks to them can be cumbersome for the user. The show block changes the play mode to Playback Slow mode, while the hide block changes to mode to Play at full speed.

4. **Status Widget:**

In the current version of Music Blocks, there is a Status widget which can be accessed through the Widgets palette in the palette menu bar. It is used to inspect the status of the running project. It utilises print blocks inside itself to print various music factors such as BPM, beat count, measure count, etc.



Status Widget

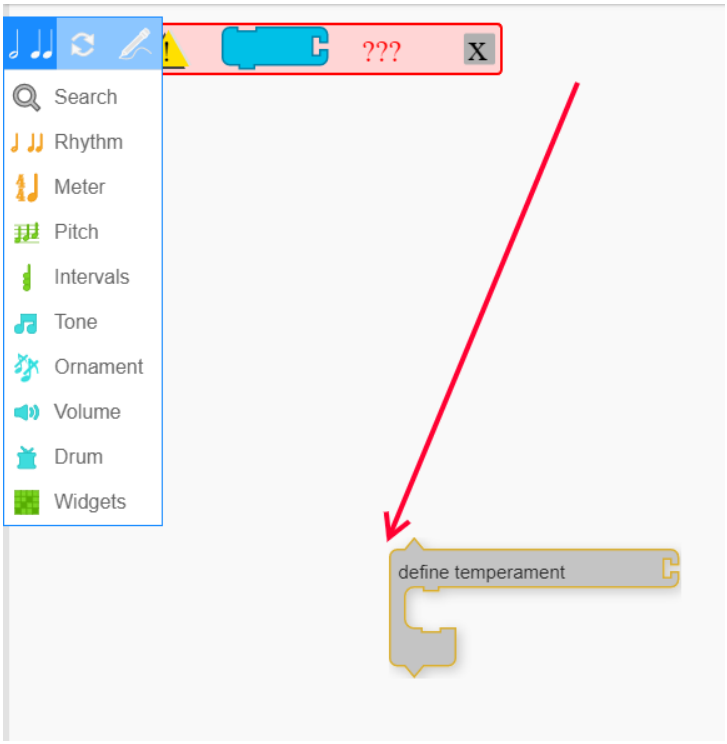
This utility can be improved by incorporating this into the UI of Music Blocks and not be used as a widget. We can take inspiration from developer tools of various browsers.

5. **Unresponsive Arrows:**

The arrows pointing to the source of warnings and errors and unresponsive to scrolling. Thus, if a user scrolls through the canvas, the arrows remain fixed and might create confusion, as these arrows may be pointing at some other block after scrolling.

6. **Arbitrary Error Boxes:**

   Currently, for any pre-recognized error, any error box with a string "???" is shown. We can improve its appearance, error message and its positioning, as it is currently positioned to be presented behind the palette menu bar. Also, we can maybe define some more errors to give more accurate information to a user. For example: The below shown image depicts this error generated due to an empty *define temperament* block. We can possibly classify such errors as "Empty Block not allowed" or something in line with this.

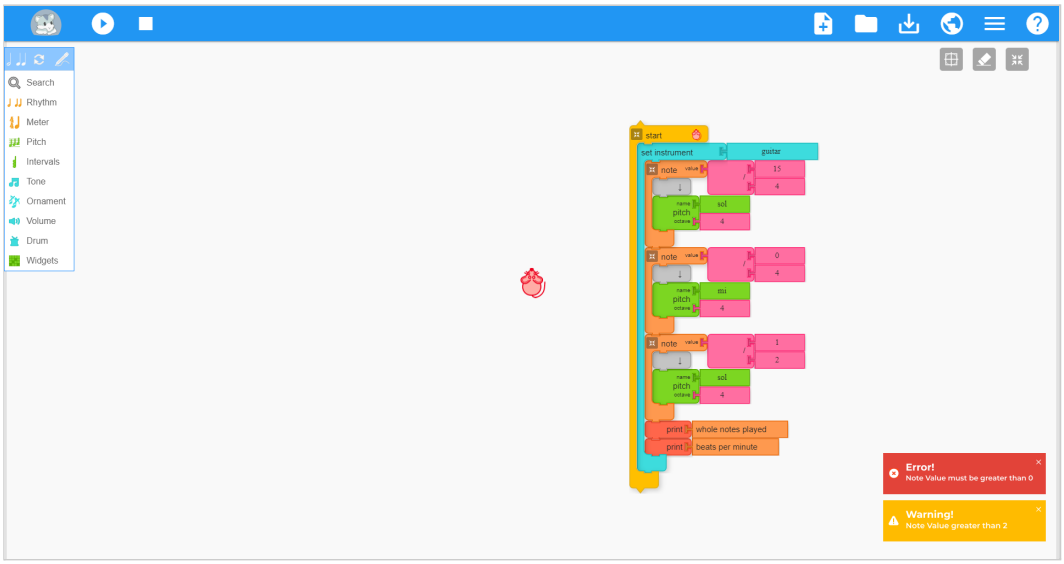


The Solution:

1. The Error/Warning messages need to be **more expressive** and be better positioned. Thus, I think they should be **positioned in the bottom right corner** of the window, where they can stack themselves in case of multiple errors/warnings. To indicate the error/warning producing blocks, one idea is to outline them by a red color. I think this will not be effective enough in large projects due to the presence of too many blocks and block-stacks.

An effective and slightly modified way of doing this is that, **on clicking these boxes the arrows are generated pointing them toward the fault producing block.** Thus, **the arrow is only generated on clicking the error/warning boxes and disappears on scroll.** This resolves the unresponsive issue of the arrows, which are a great indicator for the position error-producing boxes. The arbitrary error boxes would also be presented in the same way. **They are also presented in the stack in the bottom-right corner of the window.**

Wireframe depicting the above idea:

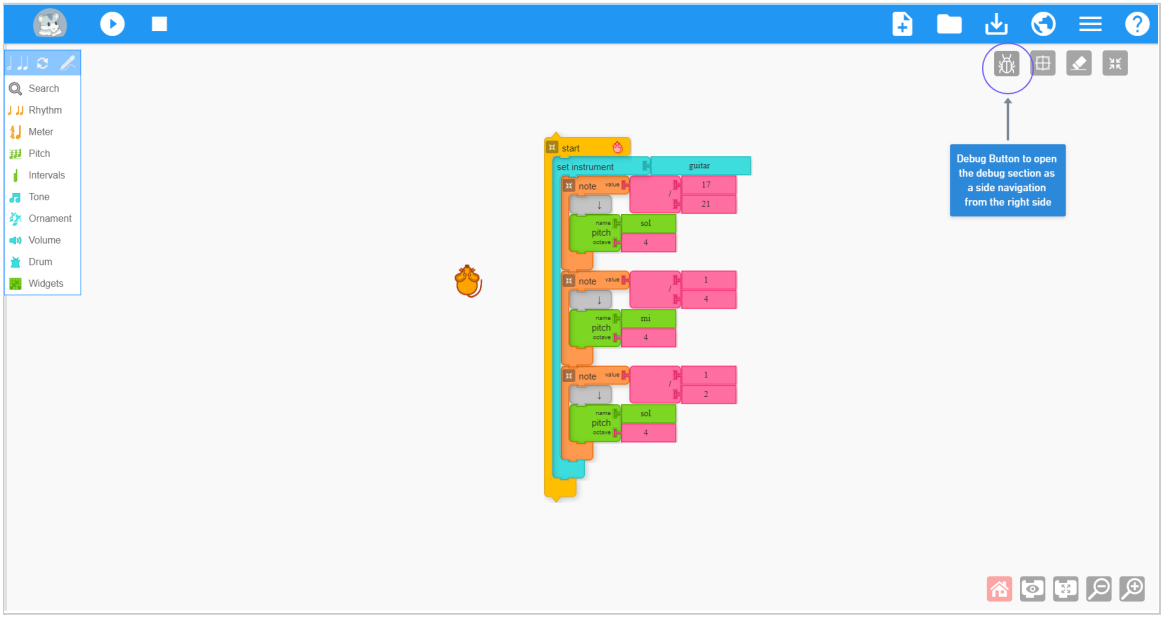


This is an example of a stack of errors and warnings. In this case, there is one error generated due to note value being 0 and a warning generated due to note value being greater than 2. However, **in case many such errors and warnings pile up, we can set a limit upto 4/5 such boxes,** such that on any new error/warning box after these upper-limit on boxes, the oldest one disappears and the latest one gets added to the stack. Also, if an arrow is already pointing towards a block, and another box is clicked, the arrow of this new box would be presented and the old arrow would disappear. In simpler words, **only one arrow will be present at any time on the window.** If we use Blockly in the new version, we can map the arrow from the error box to the block comfortably by getting the block's location relative to the predefined origin with the help of `getRelativeToSurfaceXY()` method of the Block class of Blockly. **We can optionally also add a highlight outline to the block towards which the arrow is pointing using Blockly.**

A prototype for this functionality, showing the arrow demonstration on clicking on the boxes can be found **here** in the Figma wireframe.

2. Section for Debugging:

A new section is introduced for improving the user experience of debugging a project so as to efficiently find errors and bugs in a program. This section can be accessed by a "Debug" button, present in the canvas buttons. For implementation of this section, Blockly(a client-side library for block-based visual programming languages and editors), is incredibly essential. Note that all the wireframes shown below are just the demonstration of this Debug Section in the current UI of Music Blocks(v3).



Wireframe depicting the new Debug Button

For the implementation of this debug section, breakpoints need to be introduced. **Breakpoints can be described as intentional pausing/stopping locations in a program.** It helps in the debugging process by giving the user more information about the execution of the program. Firstly, these breakpoints need to be indicated in the frontend of Music Blocks. The indication for a block being a breakpoint will be provided by the circle at the top-left of the block. Every block can be a breakpoint. **To add a block as breakpoint, we can add the option to "Add Breakpoint" in the wheelnav, that appears on right-click on a block, or we can use the dropdown menu supported by Blockly**, which can be created by instantiating the DropDownDiv by `new DropDownDiv()`.

To disable the breakpoint of a block, the same button in the wheelnav/dropdown can be used as when a user sets a block as breakpoint, the state of that block is changed and this state can be used to manage the state of the button in its corresponding dropdown/wheelnav.

The breakpoint block will be demonstrated by three states:

(i) Breakpoint Enabled: (white circle)

(ii) Program execution is at the current breakpoint block: (yellow circle)

(iii) Breakpoint Disabled: (red circle)

Inside Debug Section:

The top-part of the Debug section will have 2 sections, namely, Debugger and Status and a user can navigate through these two sections using a horizontal navigation bar. These would have the following properties:

a. ***Debugger :***

It will show the current block in the process of execution/running of the program. It will also have a total of 6 buttons in it, namely, Play/Pause, Next Step, Stop, Step In, Step Out, and Step Over.

i. The **Play button** will be used to start running the program. As soon as the program starts running, this Play button will change to Pause button.

ii. The **Next button** is used to execute only the blocks which have breakpoints enabled.

iii. The **Stop button** will be used to stop the running program.
iv. The **Step In button** is used to stop at the first child(inner block) of a block.
v. The **Step Over button** is used to access/enter the next sibling block, i.e. the next block under the same parent block.
vi. The **Step Out button** is used to enter at the parent of the current block. It is the exact opposite of the Step In button.

All these functionalities can be implemented using the Block class of Blockly which can instantiated by:

```
new Block(workspace, prototypeName, opt_id)
```

Now, this Block class has several methods which we can use in order to implement these debug utilities, like:

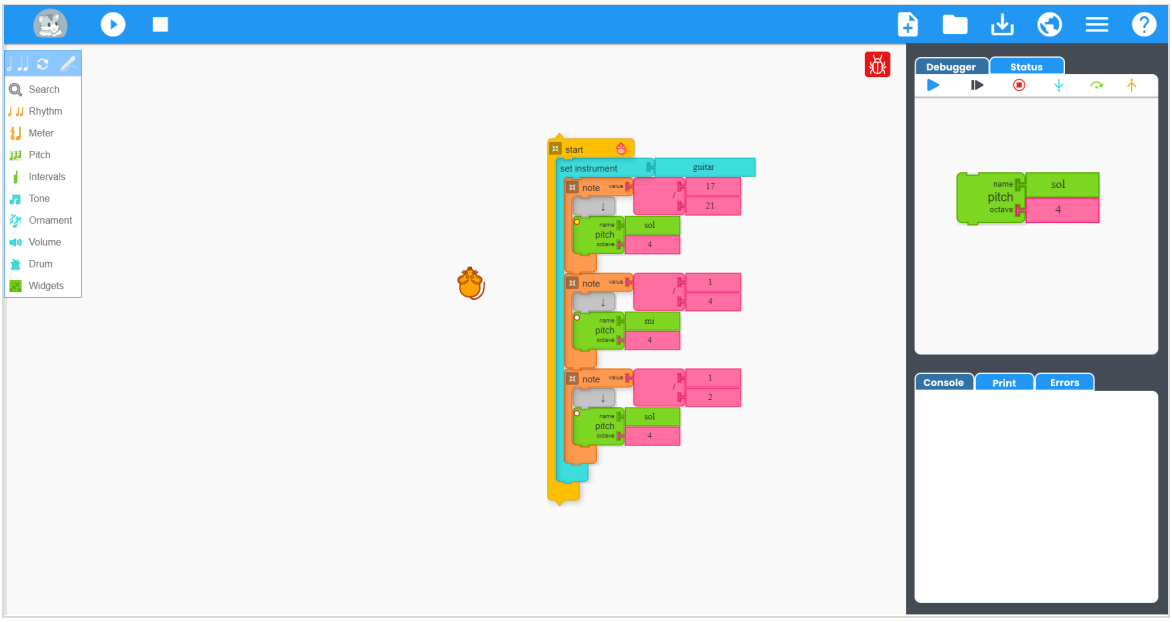
I. `getChildren()`:
This method returns the array of all the blocks nested directly inside the block of which this method is called.

II. `getParent()`:
This method returns the parent block of the block of which this method is called. If this is the top-level block itself, this method returns null.

III. `getNextBlock()`:
This method returns the sibling of the block of which this method has been called.

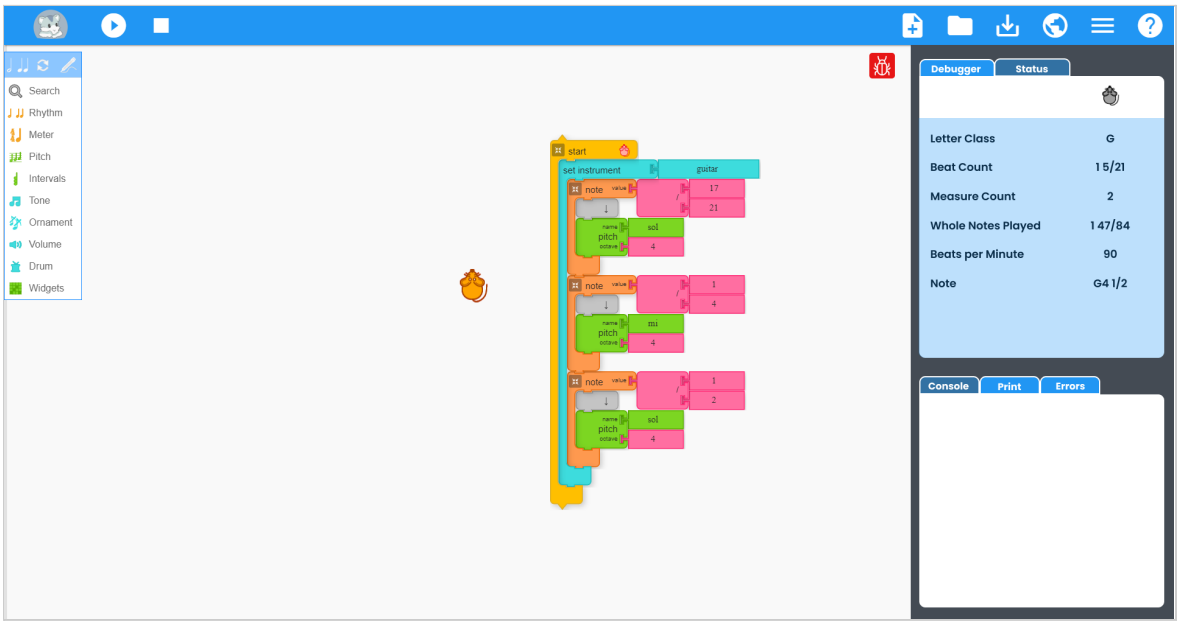
IV. `setColor()`:
This method is used to change the color of the block of which this method has been called.

Now, the Step In functionality can use the `getChildren()` method, the Step Over functionality can use the `getNextBlock()` method, and the Step Out functionality can use the `getParent()` method. The `setColor()` method can be used to highlight the current block in the process of execution. The wireframe for Debugger tab opened inside the Debug section is shown below:



b. ***Status :***

This section displays various music factors just as in the Status Widget in the current implementation. If the status widget block is present in the project, the Status tab will show the usual music factors and their values. However, if the status widget block is not present in the project and the user clicks on the Status tab, it will automatically add the status widget block in the project with the default factors(as in current implementation). The wireframe showing the active Status tab in the Debug Section is shown below:

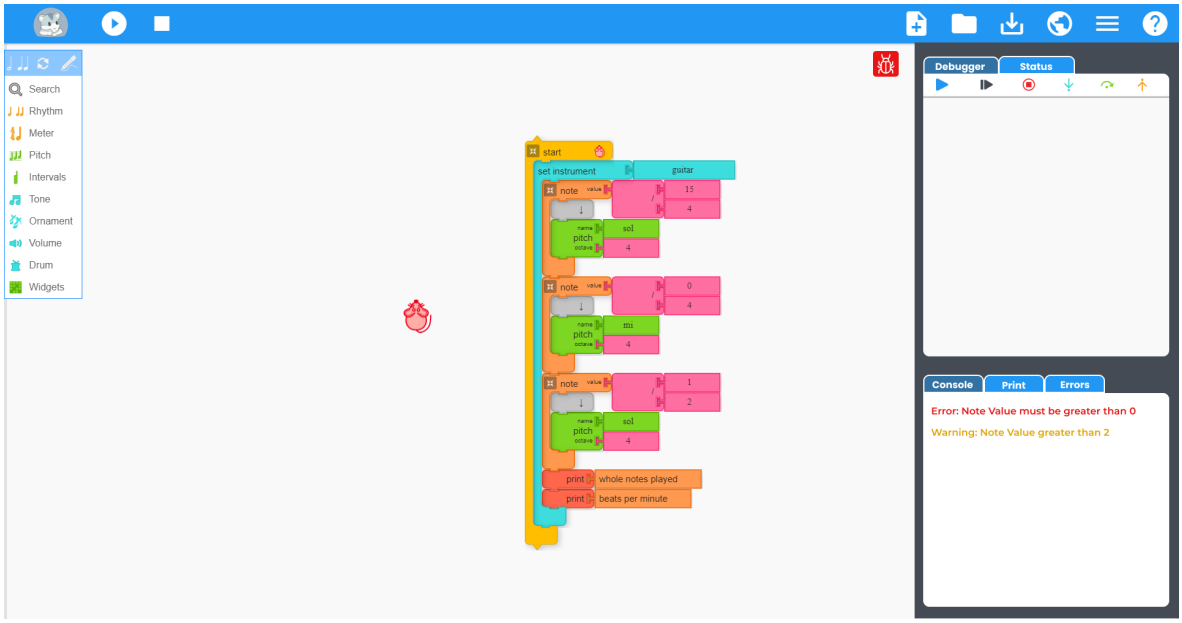


This wireframe can be found **here** and its working prototype can be found **here**.

c. ***Console Section :***

The lower half of the Debug Section is allocated to three tabs: Console, Print and Errors. **If the Debug Section is opened, then all the print messages and error/warning messages will get logged in this section and not be displayed in the usual boxes**. The console tab will show all of the Error/Warning logs and Print/Comment logs in sequential order, while the Print tab will show just the Print/Comment messages and the Errors tab will show just the Error/Warning messages.

On clicking the message an arrow will be generated from the Debug Section(location of origin of arrow would be fixed) to the block where the message is being generated. A wireframe showing console logs in an erroneous project is shown below:



Here also, only one arrow can be present at any time. A working prototype of this console section can be found **here** in this Figma Prototype. The related Figma wireframes can be found **here**.

Useful Resources and Links mentioned:

1. Blockly Visual Debugger
2. Blockly Documentation
3. Figma Wireframes for Error/Warning Boxes **(clickable boxes)**
4. Figma Wireframes for the Debug Section Upper-Half
   (clickable debugger and status tabs)
5. Figma Wireframes for Debug Section Lower-Half **(clickable arrows)**

Timeline:

Pre GSoC	• Familiarize myself with the current implementation and all the music terms. • 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 functionalities to be implemented in this project. • Work out the final framework, workflow and the structure for the debug section and the error boxes.
Week 2 (14th June - 21st June)	• Setup the display of print, comment, error, and warning boxes. • Implement the render of arrows from message boxes to blocks.
Week 3 - 4 (21st June - 5th July)	• Work on the utility to add or remove a breakpoint to a block. • Setup the UI of the debug section as a side navigation. • Work on the display of all the messages among the console, print and error tabs in the debug section. • Implement the render of arrows from console section to blocks.
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)
	• Implement the functionalities of the debugger toolbar:
	  ○ Play/Pause
	  ○ Next
	  ○ Stop
	  ○ Step In
	  ○ Step Over
	  ○ Step Out
	• Add support for various languages for messages/logs to be displayed.
	• Start working on the Status utility.

Week 8 (30th July - 6th August)	• Work on the Status tab demonstrating various music factors.
Week 9 (6th August - 9th August)	• Testing of the debug section and error handling. • Add necessary guide and 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 period, 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.