Visual Programming Module for Music Blocks

C4GT DMP - Proposal Template

Name	Justin Charles
Email ID	charlesjustin2124@gmail.com
Phone Number	+91- 8539921966
GitHub ID	justin212407
Discord ID	justin.2124
Current occupation *(Working Professionals - add current organization & years of exp)*	Student
Education Details *(College Name - Degree Name and branch of engineering or other course/specialization)*	Manipal University Jaipur – Btech in CSE(spec. in data science)
Technical skills with level *(Mention tech skills/languages known/UI-UX and level - Novice/Intermediate/Expert)*	**Technical Stack:** Javascript, Typescript, Python, C, Java | **Frameworks & Technologies**: React, Vue, TailwindCSS, SASS, Redux, Recoil, Bootstrap, Django, Django REST framework, OpenAI GPT (NLP), Jest |**Database Management System**: MongoDB, Redis, PostgreSQL | **Development & Deployment Tools**: Vercel, Git, GIthub, VS Code, Postman, NGROK, CloudFlare

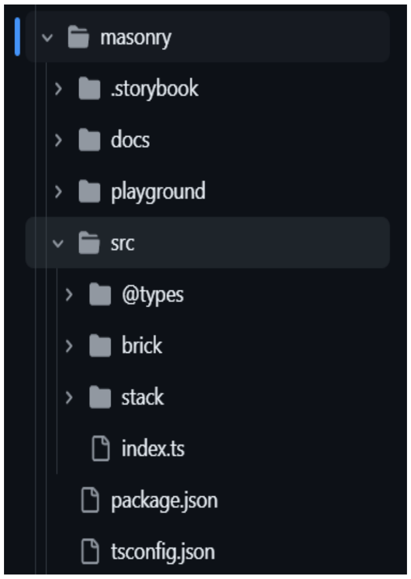
Title: The Masonry Module in MusicBlocks-v4

Summary

The Music Blocks v4 project is a reconstruction of the classic Music Blocks platform, a visual music programming environment, utilizing TypeScript, React, and contemporary tooling such as Vite, Zustand, and Jest. The proposal is centered around adding the missing "Masonry Module," a main feature that allows users to construct music programs by dragging, dropping, and linking interactive SVG-based blocks like Start, Rhythm, Note, Pitch, and Instrument. The module will feature configurable block rendering, a block palette with categories, a dynamic workspace, and utilities to visually and in-memory manage program structure. With performance, accessibility (through React-Aria), and modular design in mind, the Masonry Module will put v4 on par with the legacy version while greatly improving maintainability, user experience, and developer productivity. This project will simplify and make more intuitive the process of creating music by means of a rich, interactive web application.

The Masonry module is a streamlined module to previous years' Project-Builder Module. The work done last year creates the Masonry module which refactors the whole Project-Builder module. This module reorganized brick classes to align with their intended properties like inline editing and connection points. It also fixed bounding box rendering issues in the Masonry Module and integrated Storybook for visual testing.

The folder structure for Masonry Module ensures clarity and maintainability, separating concerns like core logic, testing, and interactive visualization. The screenshot below is of the folder breakdown for the Masonry Module:



```
masonry
  .storybook
  docs
  playground
  src
    @types
    brick
    stack
    index.ts
  package.json
  tsconfig.json
```

Project Detail

1.Project Overview:

Understanding of the Project:

1. /masonry/src - The src/ folder in the Masonry module leverages Object-Oriented Programming (OOP) concepts to structure and organize the code effectively. By utilizing classes and interfaces, it ensures that different brick types and their behaviors are abstracted into well-defined entities.

This section will breakdown the following important folders which encapsulates the essence of brick generation

a. @types: The brick.d.ts file defines core data types like TBrickType, TExtent, and TCoords to maintain consistency across bricks. It establishes interfaces for various brick types (e.g., IBrick, IBrickData, IBrickStatement) to ensure a modular design. These interfaces define brick properties, including connection points, rendering attributes, and dynamic behaviors. Bricks are classified by their TBrickKind (instruction or argument) and TBrickType (data, expression, statement, block), with geometry defined by TExtent and TCoords for size and position. The file structures how bricks are rendered, connected, and nested, with specific properties like SVG paths, labels, and scaling.

b. Brick: This folder expands into 3 more folders which are:

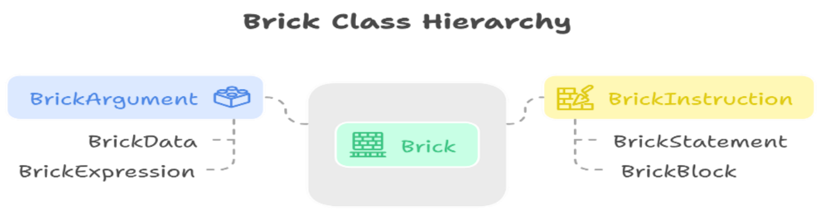
i. masonry/src/brick/design0/: The design0/ folder in the Masonry module defines the core class-based logic for various brick types in the MusicBlocks v4 system, including block bricks, data bricks, expression bricks, and statement bricks. Each brick type is modularly structured, inheriting from abstract base classes like BrickModelBlock, BrickModelData and BrickModelExpression ensuring consistency in behavior while providing specific functionalities. The BrickBlock.ts defines higher-level constructs like loops and functions, allowing nesting, while BrickData.ts stores values that other bricks use, with support for dynamic values. BrickExpression.ts handles mathematical and logical operations, connecting to other bricks for evaluation, and BrickStatement.ts defines action-triggering bricks like loops and conditionals that modify the system state. The BrickFactory.ts dynamically creates these brick types using a factory pattern, ensuring modularity, while managing each brick's lifecycle and guaranteeing unique IDs to prevent redundant creations. Together, these files enable the creation, manipulation, and rendering of bricks in a structured and interactive way, supporting the block-based programming interface of MusicBlocks-v4 . Zustand is also used for state management, specifically to manage and store the dynamic coordinates (x,y) of bricks in the workspace. Unlike more complex state management solutions like Redux or React Context, Zustand is lightweight, easy to integrate, and offers a simple API for managing global state with minimal boilerplate.

ii. masonry/src/brick/stories:The stories/components folder in the Brick module is integral to rendering and testing the various types of bricks in the MusicBlocks v4 system within the Storybook environment. This folder contains key React components like BrickBlock.tsx, BrickData.tsx, BrickExpression.tsx, BrickStatement.tsx and BrickWrapper.tsx, which define how each brick type block, data, expression, statement is displayed and interacted with during development. For example, BrickBlock.tsx handles the rendering of block bricks, such as functions and loops, while BrickData.tsx focuses on rendering data bricks that store variable values. BrickData.tsx and BrickStatement.tsx manage the rendering of expression and statement bricks,which deal with mathematical operations and actions like loops or function calls. BrickWrapper.tsx is a higher-order component that ensures the proper rendering of all these bricks. Alongside these components, the TypeScript files (brickBlock.ts, brickData.ts, brickExpression.ts, brickStatement.ts) provide the necessary logic and functionality for each brick type, defining their properties, methods, and interactions within the system.

iii. masonry/src/brick/utils: This folder includes two key files:

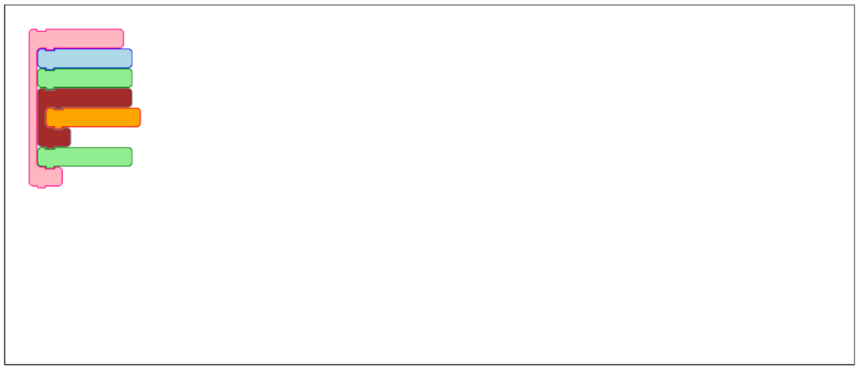
path.ts: defines utility functions to generate SVG paths with precise dimensions and positions, using private helpers and public path assembly functions.

path.spec.ts: tests these utilities to ensure accuracy and handle edge cases reliably.



c. Stack - The data.ts file in the stack module defines the logic for managing and interacting with the stack structure used in MusicBlocks. It contains the interface, representing nodes in the stack, and the IStack interface, which defines the stack's operations, such as adding, removing, moving, and validating nodes. The stack structure allows bricks (such as BrickModelData, BrickModelExpression, BrickModelStatement and BrickModelBlock) to be organized hierarchically, supporting nested and dynamic structures. The file also includes the implementation of the StackNode and Stack classes, which encapsulate the behavior of nodes and stacks respectively. The Stack class provides methods for manipulating and validating the stack, handling tasks like collapsing and expanding block nodes, and checking connection validity between nodes.

2. /masonry/playground - The playground/ folder in the Masonry module serves as an interactive environment within MusicBlocks v4, designed to visualize and manipulate various types of bricks, such as blocks, data, expressions, and statements. It contains essential files like workspace/ which is the main page for rendering the interactive brick components, and houses components such as BrickBlock.tsx (for general-purpose blocks like loops and functions),
BrickData.tsx (for data storage), BrickExpression.tsx (for operations and expressions), BrickFactory.tsx (which dynamically selects and renders the correct brick type), and BrickStatement.tsx (for action-based bricks). Additionally, BrickStore.ts manages brick positions, while data.ts provides initialization data for rendering, and index.tsx serves as the entry point for initializing and coordinating the rendering process. The folder also includes utils.ts for helper functions that optimize rendering and state management, with index.html providing the base HTML structure and index.tsx bootstrapping the React components for dynamic user interaction. Together, these files facilitate a seamless interactive testing and experimentation platform for MusicBlocks v4.
This is how the playground looks right now when i run it locally:

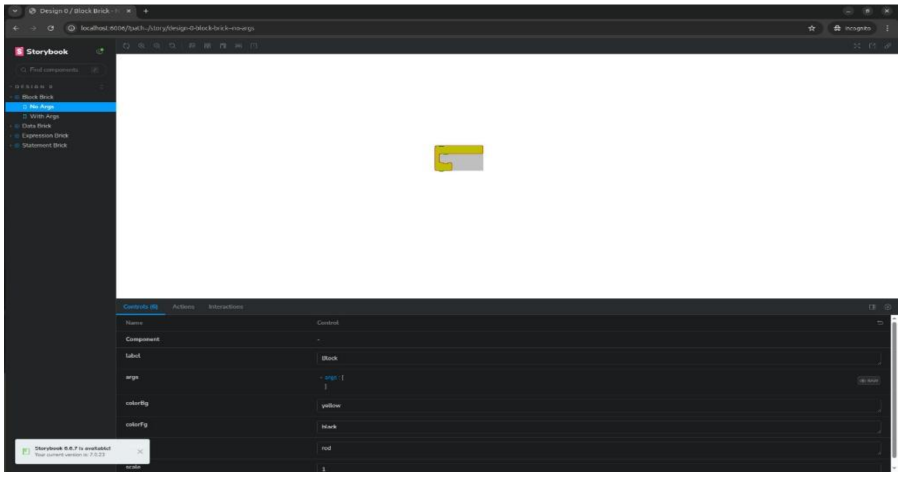


3. /masonry/.storybook - The .storybook folder in the Masonry module configures Storybook, a tool used to visualize and test UI components in isolation. It helps developers debug and test block-based

components like BrickBlock, BrickData, and BrickExpression before integrating them into the broader MusicBlocks system. The folder contains two main files:
main.ts configures story loading, addons, and Vite integration for efficient builds and preview.ts sets global parameters, styling, and enables live updates during testing.
Together, these files streamline the development and testing of Masonry components in isolation.I am not making a detailed breakdown of this since this is something that I will be refactoring and working upon this summer.
This is the flow of execution that follows through in the storybook:

```
User opens Storybook →
        └── design0/stories/BrickBlock.stories.ts (story definition)
        └── calls stories/components/BrickBlock.tsx (renderer)
        └── instantiates design0/BrickBlock.ts (model)
        └── renders design0/components/BrickBlock.tsx (UI view)
```

This is how the storybook looks like right now when i run it locally:



Name	Control
Component	-
label	Block
args	args : [ ]
colorBg	yellow
colorFg	black
	red
scale	1

Goals to be fulfilled for Epics:

1. **Brick:**
    a. By supplying a configuration object that defines type, UI, inputs, and optional logic, the Masonry Framework should be able to render and initialize any valid brick dynamically for the user.
    b. Users should be able to connect and disconnect bricks using Visual Support Anchor.

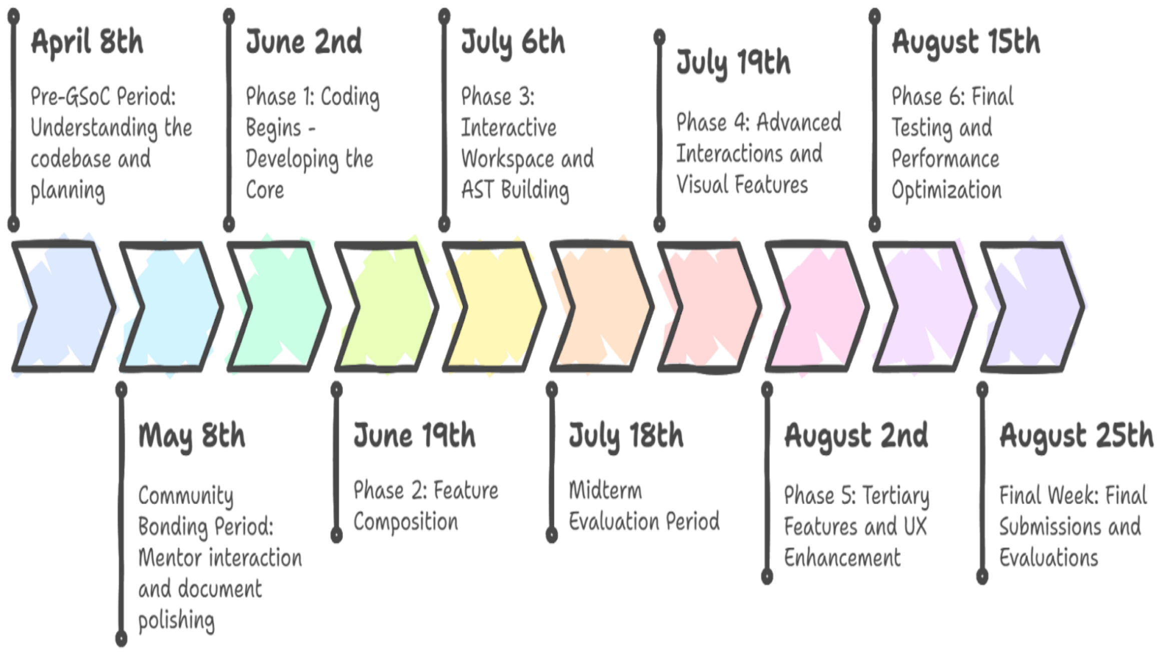
c. Bricks should be styled specifically according to their basic configurations.
d. Different Bricks can have context menus specific to the specific brick.
e. MusicBlocks-v4 should support a set of brick-level utility actions, accessible via right-click context menus.
f. MusicBlocks-v4 should allow users to switch between related brick types directly within the workspace.

2. **Stack:**
    a. Generating a clean AST (Abstract Syntax Tree) from the bricks connected in the workspace
    b. The user should have a feature to fold/unfold block-type bricks.
    c. MusicBlocks-v4 should support automatic height adjustment of parent bricks based on the dimensions of their child bricks.
    d. The users should have real-time visual and contextual feedback during brick connections.
3. **Workspace:**
    a. The user shall be able to drag and drop brick into the workspace and move it around.
    b. MusicBlocks-v4 should have buttons around the start block to run only that thread, go step by step and go slowly.
    c. Bricks and blocks of bricks should not overlap when placed or moved within the workspace.
    d. Users should be able to delete bricks/blocks of bricks by dragging them to a corner of the workspace.
    e. Users should be able to scale blocks in the workspace.
    f. Users should be able to add inline comments and debug breakpoints within the workspace.
    g. Users should have access to buttons around the start block to run only that thread, go step by step and go slowly.
4. **Palette:**
    a. A list of Bricks shall be available (palette) shall be available for the users to choose specific bricks from
    b. The user should have a search functionality for bricks in the palette that allows users to instantly filter bricks by name and type.

Some other deliverables that I will be working upon are:

- Write Storybook stories to document and showcase UI components.
- Implement unit tests for functions and classes using Vitest.
- Export a minimal API for integration with other parts of the application.
- Produce developer documentation to describe main modules, data structures, and APIs.

Project Timeline:


April 8th
Pre-GSoC Period: Understanding the codebase and planning
May 8th
Community Bonding Period: Mentor interaction and document polishing
June 2nd
Phase 1: Coding Begins - Developing the Core
June 19th
Phase 2: Feature Composition
July 6th
Phase 3: Interactive Workspace and AST Building
July 18th
Midterm Evaluation Period
July 19th
Phase 4: Advanced Interactions and Visual Features
August 2nd
Phase 5: Tertiary Features and UX Enhancement
August 15th
Phase 6: Final Testing and Performance Optimization
August 25th
Final Week: Final Submissions and Evaluations

<u>Here is the detailed Breakdown of my timeline:</u>

Pre-GSoC Period (April 8th - May 8th) -

- Understanding the codebase → I have already understood a major chunk of the codebase and will be done with the whole module till the start of the community bonding.
- Refactor and Plan → I have already started drafting my approach for refactoring the Brick Class to add extensibility for custom types, nested bricks, and text to SVGs.

Community Bonding Period (May 8 - June 1) -

- Mentor Interaction → As I am already in contact with the project mentors regarding the flow and direction of the project this won't take much time.
- Polishing the existing documents → I have refined all of the existing documentation and got it reviewed by the project mentors beforehand, hence not much time will be invested in this and we can move forward with the coding section. The proposal already covers most of the high level technical approach as well as the design document. We shall focus on the low level technical approach and the specifics to development.

Phase 1: Coding Begins - Developing the Core (June 2 - June 18) -

- Create Utilities to Create SVG Paths for Bricks → Develop a dynamic utility that produces SVG path strings for bricks according to their configuration. Make sure it accommodates dynamic rendering requirements and bounding box calculations.
- Construct Utilities to Represent and Manipulate Programs In-Memory → Develop an in-memory structure that can represent Music Blocks programs including brick sequencing, nesting capabilities, and ways of querying and sorting bricks.

Phase 2: Feature Composition (June 19 - July 5) -

- Build the foundation for rendering any brick purely from configuration (including type, label, inputs, ports, and visual markers).
- Load and display the list of available bricks from external configuration files (defining palette externally).
- Establish Zustand state structure to support reactive workspace logic and stateful interactions across all submodules.

Phase 3: Interactive Workspace and AST Building (July 6 - July 17) -

- Support drag-and-drop of bricks from the palette into the workspace, and enable freeform dragging and repositioning within the workspace area.
- Provide connection and disconnection between bricks, and offload connection validation logic to an external engine or validator module.
- Provide deletion of bricks or nested blocks by dragging them into a trash area on the canvas.
- Develop serialization of the workspace into an abstract syntax tree (AST) format, with preservation of brick structure and hierarchy.
- Begin performance optimization to ensure drag, drop, and connect operations continue to be smooth with respect to user interaction.

Midterm Evaluation Period (July 18) -

- Submit Midterm Evaluation → Midterm evaluation form submitted with progress report and updates on all implemented features.
- Ensure that all major components of the system are working, modular, and test-covered internally.

Phase 4: Advanced Interactions and visual features (July 19 - August 1) -

- Provide rich styling support for bricks (colors, hover, active, etc.) via their external configuration.
- Provide contextual menus and inline editors for each brick, enabling toggling, dropdowns, or advanced field configuration.
- Provide brick utility actions such as duplicating a brick, extracting from a block, deleting through context menu, and disconnecting inputs.
- Provide fold/unfold behavior for block-type bricks like start, repeat, and set instrument.
- Implement brick and block scaling in the workspace, with zoom controls or responsive resize.
- Make sure parent block heights automatically size to accommodate their child bricks, addressing the layout problem found in MB3.
- Add visual feedback (warnings, tooltips) upon invalid connections, such as missing inputs or incompatible types of bricks.

Phase 6: Tertiary Features and UX Enhancement (August 2 - August 15) -

- Support dynamic switching between analogous bricks (e.g., if ↔ if-else, pitch in MIDI ↔ octave ↔ hertz).
- Include a search input in the palette that filters bricks by name, type, or tags in real-time.
- Avoid overlapping of bricks and blocks by using layout constraints or repositioning logic on drag/drop.
- Introduce control buttons for execution near start bricks to execute a particular thread, step through it, or execute in slowly.
- Provide ability to add comments in the workspace, either as floating comments or to particular bricks.
- Implement support for turning on/off debug breakpoints on individual bricks, with interaction with the debugger and execution engine.

Phase 6: Final Testing and Performance Optimization (August 15 - August 24) -

- Create unit tests for the core logic with Vitest → such as brick config management, workspace updates, AST generation, and interaction helpers.
- Prioritize performance optimization for all interactions especially brick rendering, AST serialization, and drag/drop mechanics.
- Refine visual details, resolve edge case bugs, and improve the UI/UX to provide a consistent and accessible experience.

Final Week: Final Submissions and Evaluations (August 25 - September 1) -

- Fixing any last moment bugs and ensuring a smooth integration and performance of the project.
- Submitting Final Work Product that is the code and the documentation.
- Final Mentor Evaluation.

Availability

Number of hours available to dedicate to this project per week	40 hours
Do you have any other engagements that will require your time? (projects/internships)	None

Personal Information

About Me:

My initial introduction to the concept of open-source software dates back to 2021, when our teacher illuminated the fascinating world of publicly accessible source code. The notion that companies willingly unveil their proprietary code for collective improvement and innovation piqued my curiosity. Now, driven by an eagerness to contribute, I find myself delving into MusicBlocks,going through each intricate detail present.

I hold a high school diploma with a specialization in Information Technology. My academic journey encompassed learning the fundamental principles of programming, with a particular emphasis on Python’s syntax and structural paradigms. Additionally, we explored the intricacies of database management, gaining proficiency in relational database languages such as MySQL, which allowed us to grasp the fundamentals of data organization, manipulation, and retrieval.

My journey into Web Development commenced in the summer of 2024. Although I was not entirely a novice, my prior knowledge was limited to the foundational building blocks of the web HTML and CSS. As I progressed through this journey, I not only expanded my technical expertise but also became an **active member** of the **developer community** of SugarLabs.

My introduction to the open-source ecosystem came when I participated in **Hacktoberfest 2025**, an experience that provided me with invaluable hands-on exposure. Through this initiative, I learned the essential workflows of open-source contributions i.e. creating issues, opening pull requests, and adhering to best practices in version control. This proved to be a pivotal learning experience, deepening my understanding of collaborative software development. Over the course of the event, I successfully contributed **8 pull requests across various codebases**, with **4 of them being merged in my very first week**. This milestone solidified my enthusiasm for open-source development and further fueled my passion for contributing to impactful projects.Here is a LinkedIn post and tweet regarding the same.

I also recently participated in my first Hackathon which I won. Here is a tweet about the same.We created 5 projects in the same hackathon while creating an ecosystem connecting all of them together.

What is your motivation to apply for this project? Answer briefly in 5-10 lines.

I first came across Sugar Labs in 2023 while searching for a platform capable of generating guitar sounds with diverse tonal qualities. My curiosity led me to MusicBlocks, where I was fascinated by how it processed music through the Tone.js library while simultaneously providing a visual representation of sound. Having spent six years in formal guitar education, I was naturally drawn to the project and saw an opportunity to contribute meaningfully. My active involvement in MusicBlocks-v3 not only deepened my understanding of its inner workings but also led to an invitation to officially join the repository as a member on February 15, 2025. This milestone, which I documented in a LinkedIn post and Tweet, marked a significant step in my journey with the project.

As my engagement with MusicBlocks and Sugar Labs grew, I began exploring their broader ecosystem and discovered MusicBlocks-v4—a complete architectural overhaul of the platform. Given that it is built with React.js and TypeScript, technologies I have extensively worked with, I was immediately intrigued by its potential. By leveraging the capabilities of a robust front-end framework, MusicBlocks-v4 enables seamless integration with web-based workflows, ensuring it stays relevant and adaptable to contemporary technological advancements.
What excites me most is its potential to redefine generative music education and creative coding, shaping a tool that adapts to the evolving digital landscape. As a committed member of the MusicBlocks community, I aim to refine its capabilities, optimize performance, and push its creative boundaries.

Please mention if you have solved any issues/tickets for this or other C4GT projects:

Currently I have a total of 11 PRs merged and 2 open. Here's a concise table with a categorisation on them. Each file has a list of PRs with their brief description

S.No.	Resolution description in short	Link to pull request
1.	Play-Only Mode	1. [Feature] Adding Play-Only-Mode to MusicBlocks(merged) - #4320 2. [Enhancement] Adding Meta tag for ensuring transition of Play-Only-Mode(merged) - #4494 3. [Enhancement]Added the necessary features for Play-Only-Mode(open) - #4516
2.	Dark Mode	1. [Feature]Added Dark Mode for Planet Page(merged) - #4297 2. [Bug]Fixed Dark Mode Issues(merged) - #4463
3.	Test Files	1. #4291 2. #4209 3. #4234
4.	Deployment Workflow	[Enhancement] Deployment pipeline(open) - #4290

Previous experience/open source projects:

Project Name	Project Description	Links (if any)
Arcadia SLCM	• Web3 Domain Name Service (DNS) for students. • This innovative platform will equip students with Web3-	Deployed Website Github Repository

	university's domain compatible domains under the name, empowering them to securely receive, manage, and verify their NFT credentials	
Arcadia Meme	• Arcadia Meme simplifies memecoin creation with seamless minting on the blockchain. • This platform gamifies the trading experience, making it fun, engaging, and low-stakes for users who want to explore crypto markets without the pressure of high-risk investments.	Deployed Website Github Repository
Arcadia	• Arcadia is an innovative NFT maker platform that empowers users to effortlessly create, customize, and surprise themselves with unique digital collectibles, blending creativity with Aptos.	Deployed Website Github Repository
Zenith Wallet	• A Mini App designed to provide a seamless, user-friendly cryptocurrency wallet experience. • It allows users to create, manage, and interact with their wallets, send and receive EDU tokens, and enjoy real-time balance updates.	Deployed Website Github Repository