**Design Document for Scratch .sb3 Grading Library**

Team Members: Devan Griffin, Jake Houghton, Jasper Nisly

Stakeholders: Russel Feldhausen, Safia Malallah

**1. Introduction**

A library designed to assist instructors with automated grading of Scratch .sb3 projects. This library will extract, parse, grade, and provide feedback from Scratch projects based on predefined criteria and will offer feedback mechanisms for students.

**2. Components**

**2.1. File Extraction**

* Responsible for extracting contents of the .sb3 file, which is essentially a .zip file.
* Key Libraries & Technologies:
* [JSZip](https://stuk.github.io/jszip/)

**2.2. JSON Parsing**

* Converts the project.json data within the .sb3 file into structured JavaScript objects.
* Creates a preliminary structure, making data traversal more straightforward for the next stages.

**2.3. Abstract Syntax Tree (AST) Building**

* Convert parsed JSON into a more queryable AST structure.
* Captures:
* Blocks and their types
* Inputs to the blocks
* Child blocks
* Parent block references
* Contained blocks (for C-shaped blocks)
* This AST will serve as the foundation for the grading operations.

**2.4. Grading Engine**

* Contains grading rules and logic.
* Capabilities:
* Query AST to find specific blocks by type or input values.
* Analyze music sequences (considering inline math and ignoring extraneous blocks).
* Ascertain sprite positions based on sequences of blocks.
* Other criteria as defined by instructors.

**2.5. Feedback Mechanisms**

* Provides feedback to students based on the grading results.
* Handles the feedback through various outputs:
* Writing to stdout or stderr.
* Managing the exit codes.
* Sending partial points through GET requests to the CODIO\_PARTIAL\_POINTS\_URL.

**2.6. Error Reporting**

* Handles unexpected errors or issues.
* Reports issues to the ALT+CS lab.

**3. Non-Functional Requirements**

* Efficient in performance.
* Robust error handling.
* Modular and extensible design for future updates or grading requirements changes.

**High-Level Steps to Create the Application**

**1. Setup & Initialization**

* Initialize a new JavaScript project.
* Install necessary libraries (e.g., JSZip for file extraction).
* Set up version control, preferably Git, to track code changes and enable collaboration.

**2. File Extraction**

* Create a module dedicated to extracting .sb3 files.
* Use JSZip to unpack the .sb3 (which is a zip file) and retrieve project.json.

**3. JSON Parsing**

* Develop a module to parse project.json into JavaScript objects.
* This will create a preliminary nested structure of the Scratch project.

**4. AST Building**

* Design the AST structure based on Scratch's blocks and the requirements.
* Convert the parsed JSON into this AST structure, making sure to capture all necessary relationships and data points.
* Implement utility functions to easily traverse the AST (e.g., get all child blocks, get parent block, etc.).

**5. Grading Engine**

* Establish a set of base rules for grading.
* Implement functions to query the AST and grade based on the established rules.
* For complex grading tasks (e.g., analyzing music sequences), develop specialized functions to handle specific grading criteria.

**6. Feedback Mechanisms**

* Implement functions to provide grading feedback to students.
* Handle different types of feedback outputs, from writing to standard streams to sending GET requests for partial points.

**7. Error Reporting**

* Implement robust error handling throughout the library.
* Create a dedicated module for error reporting to the ALT+CS lab.

**8. Testing & Validation**

* Use libraries like [Jest](https://jestjs.io/) for unit testing.
* Regularly test the library against a variety of Scratch projects to ensure robustness and accuracy.

**9. Documentation & Collaboration**

* Document each module and its functions.

**10. Deployment & Distribution**

* Package the library for easy integration into Codio or other platforms.
* Provide clear setup and usage instructions for instructors.