Progsnap 2 CodeState Representation

CodeStates are used to represent student code in the Progsnap 2 format. Unlike the main table, CodeState representation can and should vary across different datasets, based on the type of work being stored. However, we recommend that developers represent their code using the following guidelines, to increase shareability.

CodeState Directory

The CodeState directory should hold the files containing the code data, where files can be referenced by ID in the main table. How that data is stored varies based on the type of data that has been logged. We consider varying types of data along the following dimensions:

- Program size (small or large)
- Number of files per CodeState (one or multiple)
- Code format (text or other)
- Size of dataset (above or below threshold)

The recommended representation format can then be decided based on the factors mentioned above:

- The **Git Format** might be most appropriate for large datasets and datasets with snapshots consisting of large numbers of files
- The **Directory** Format might be most appropriate for medium size datasets and datasets where the snapshots contain a small number of files (including single files)
- The **Table Format** might be most appropriate for datasets where each snapshot is a single file of a relatively small size and the source code representation is text-based

Producers of Progsnap 2 datasets must use one of the following CodeState representations. In general, it is left up to producers to decide which representation is most appropriate (considering the factors described above.) Producers should specify the CodeStateRepresentation property of the dataset's `DatasetMetadata.csv` to Git, Directory, or Table as appropriate.

Consumers of Progsnap 2 datasets should be prepared to work with any of the following CodeState representations.

All code data should be stored in the dataset's `CodeStates` directory.

Table Format

In the Table Format, all CodeStates are stored in a single CSV file for ease of access. This CSV should be named `CodeStates.csv` (in the `CodeStates` directory), and should contain two columns: ID and code. We strongly recommend that developers generate the CSV in RFC 4180 format using a standard library, to avoid parsing problems.

The ID column should hold each code state's ID, which should follow the ID datatype described by the main table. The code column should hold the complete program text referenced by the ID. The whole code text should be included, not just a diff.

Directory Format

In the Directory Format, each CodeState is stored in a different directory within the dataset's `CodeStates` directory, where the name of the directory is the CodeState ID. The directory then contains all files associated with the CodeState in their full (non-diff) form. Each CodeState directory may have an internal directory structure, and files may be located anywhere within the internal directory structure.

Care must be taken to ensure that CodeState ID values are legal directory names for common operating systems. For this reason, it is recommended that Directory Format CodeState ID values consist exclusively of the characters a-z, A-Z, 0-9, underscore ("_"), and hyphen ("-").

These directories should be **deduplicated**: that is, no two distinct CodeStates should be identical. If a unique program occurs across multiple events or students (for example, the starter code for an assignment), it should be referred to by the same ID in each event where it occurs.

Git Format

In the Git Format, a single Git repository contains all of the dataset's CodeStates, with each CodeState being represented as a commit.

In this representation, the CodeStates directory should be a bare Git repository (i.e., one created with the `git init --bare` command.) Each CodeState ID should name a commit in the Git repository. It is strongly recommended for dataset producers to use parent/child relationships between commits to represent code histories. For example, consider a sequence of edits made by one student working on one project. Assume there are CodeStates C1, C2, C3, etc., representing the edit sequence. Each CodeState is a commit in the Git repository. In order to model the connections between the CodeStates in the sequence, C1 should be C2's parent, C2 should be C3's parent, etc.

CodeStateSection Representation

[Note- this will be moved into the main protocol file later]

CodeStates are generally composed of multiple parts, which may be files, scripts, or problems. A CodeState, CodeStateSection, and SourceLocation, when combined, must allow the user to retrieve a specific subset of the CodeState from an external source. The representation of a CodeStateSection varies depending on the CodeStates representation:

- CodeStates represented in the Table Format do not require CodeStateSections.
- CodeStates represented in the Directory Format or the Git Format require a CodeStateSection that takes the form of a path from the top of the directory to the file that the event takes place in. For example, the path ‘tools/convert.py’ would refer to a python file (convert.py) in the tools directory of the main directory.

Developers should avoid events that require multiple files. Instead, these events should be broken into multiple events that each reference one file at the same timestamp.

Open Questions

- Should information about a CodeState’s language be included in the CodeState, or in the main table?
- Should more structured formats (like an AST) be stored in the CodeState, or generated by a script later on?