Final task-Open Ended Question

Description

For a project you've worked on (e.g., a research project or class project), please share or describe the organization of data/materials and associated documentation for the project. You can pretend like you are passing the project off to us, so you should give us everything you have to help us make sense of the project materials to date. You can include any info, descriptions, links, or files you think would be helpful. Note, actually sharing data files is not necessary.

Response (link to Google Docs)

Task Overview

As Research Software Engineer at jsPsych, I am adapting the tasks in the NIH Toolbox Cognition Battery (NIHTB-CB) into jsPsych experiment packages. Each of all 12 tasks in the battery is to be implemented as a separate package under the `packages` folder in the `jspsych-timelines` repository, which is jsPsych's community repository for sharing experiments as "timelines" that can then be easily imported, modified and run online interoperably by other researchers.

Repository Structure

Here is a walkthrough of the structure of jspsych-timelines, the most relevant files commented:

```
jspsych-timelines/
├── packages/                    # All available timeline packages
│   ├── arrow-flanker/
│   ├── false-memory/
│   ├── spatial-cueing/
│   └── list-sorting-working-memory/  # Example NIHTB-CB task implementation
├── .github/             # GitHub workflows for automated build and release
├──.changeset/                   # Config files for version management
├──.gitignore
├── gulpfile.js
├── LICENSE
├── package-lock.json
├── package.json
├── README.md
└──update-readme.js # Updates root README to reflect package changes
```

Implementing a New NIHTB-CB Task

You should mostly be focusing on adding your implementations of the NIHTB-CB tasks as folders under the `packages` folder, where all the actual experiment timeline packages are. Here is a high-level walkthrough of how you would start implementing a task in NIHTB-CB as a jsPsych timeline package in jspsych-timelines:

Step One: Set Up Local Development Environment

1. Clone `jspsych-timelines`

```
Unset
git clone https://github.com/jspsych/jspsych-timelines.git

cd jspsych-timelines
```

2. Create a new branch and pull request for your new timeline package (e.g., `'nihtb-cb_oral-symbol-digit-test'`) and check the branch out locally. As an example, I have created a separate branch and a pull request for the implementation of the List Sorting Working Memory Task in NIHTB-CB

```
Unset
git checkout -b nihtb-cb_<task-name>
```

3. Install dependencies from your root directory (`jspych-timelines`)

```
Unset
npm install
```

Step Two: Generate a New Timeline Package

1. Use our built-in tool for creating new timeline packages in a standardized format:

```
Unset
npx @jspsych/new-timeline
```

2. Answer the prompts generated by the tool in the command line to create a new timeline template in the packages directory. These prompts will ask for information like the package's author's name, a brief description of the package, a link to the author's GitHub, etc.

Step Three: Understanding the Generated Boilerplate Package

After these steps, a boilerplate package for your new timeline package should be created under the `packages` folder based on the timeline template specified in `@jspsych/new-timeline`. This boilerplate package has the following structure:

```
boilerplate-timeline-package/
├── examples/
│   └── index.html          # Example .html file for running the timeline on a browser
├── src/
│   └── index.ts            # Main source file containing timeline logic
├── jest.config.js          # Configuration files for testing
├── package.json            # Package metadata and dependencies
├── README.md               # Documentation for your package
├── tsconfig.json           # TypeScript configuration
└── update-readme.js# Script to auto-generate API documentation in README.md
```

Now, you should edit the files in the new boilerplate-timeline-package to implement your timeline. `src/index.ts` is the main file that will be loaded when the timeline is run, and where you will be writing your main body of code that implements your task as a jsPsych timeline.

Step Four: Implement Your Timeline

The most important thing to know about developing a timeline package for `jspsych-timelines` is that you are highly encouraged to use our `new-timeline` command line tool and follow the standardized source script export structure it imposes. You can find more detailed information about how to use this tool, the standardized format it enforces on the created timeline package source scripts, and the reasoning behind it in the `jspsych-dev/packages/new-timeline/README.md` file. We also have a working document describing this standardization and how it encourages open, collaborative development in the scientific community.

In brief, your `src/index.ts` file should export three things:

1. The main `createTimeline()` function that serves as your default export and runs the entire experiment timeline
2. The `timelineUnits` array that exports meaningful partitions of your experiment timeline that you think would be useful and reusable for others, e.g. demos, test blocks, procedures, individual trials, etc.
3. The `utils` array that holds any useful functions for the smooth running of your experiment

The trickiest challenge in this process is determining how to demarcate your code in a way that produces useful, meaningful partitions that you can then export for others to use. A good strategy is to think about your conceptual understanding of your experiment _ what are the different logical components that build up to it?

See footnote[1] if you need help getting started with writing a jsPsych experiment

Step Five: Testing Your Timeline

When you are done editing `src/index.ts`, it is time to make sure your experiment actually runs!

Here are the steps:

1. Navigate to your package directory to build the timeline. This will create a `dist` directory with the compiled timeline.

```
Unset
cd pacakges/<task-name>

npm run build
```

2. Verify that the timeline works by opening `examples/index.html` in your browser. `examples/index.html` is a basic jsPsych experiment template that you can modify to illustrate how your timeline works.

3. Fix any issues in your code.

[1] You can follow a thorough tutorial demoing the creation of an experiment timeline on the jsPsych webpage, or read through the other packages in `jspsych-timelines`.

For an example timeline closer to your implementation of NIHTB-CB tasks, you should check out my recent, complete implementation of the Hearts and Flowers Task, a classic experiment paradigm we implemented in preparation of a meeting with Levante, a friendly lab focused on developing assessment suites for investigating how variability in development affects learning outcomes.

Step Six: Document Your Timeline Package

When your timeline runs successfully, it is time to publish it to be publicly accessible by officially adding it to `jspsych-timelines`! The repository has tools that automate this process:

1. Auto-generate documentation for your timeline's `README.md` and modify to make sure there are clear descriptions of the task, usage instructions, and fleshed out documentation on the parameters and customization options for each function

```
Unset
node run update-readme
```

Step Seven: Publish Your Package for Others to Use

1. Add a changeset in the main directory of the package repository. This will prompt you for a description of the changes you made and create a new changeset file in the `changesets` directory.

```
Unset
    npm run changeset
```

2. Our team, which you are part of now, will review your code and if all is well, merge your pull request into the main branch of `jspsych-timelines`. The `release.yml` file under `./github/workflows` in `jspsych-timelines` will detect the changeset in your pull request and automatically release your timeline package to the npm registry, a popular registry for sharing JavaScript packages. Your timeline package can now be easily found under the `@jspsych` namespace on `npm`!

Open Science Statement

By following this structure of writing jsPsych experiments, your implementation will help open science practices in cognitives science research by making classic experiment paradigms more readily adopted and run by researchers worldwide. Our standardized format for timeline packages promotes:

- Code reuse and interoperability
- Transparent scientific methods
- Collaboration within the research community
- Reproducibility of experiments