Supplementary Material for ProtoGraph: A Non-Expert Toolkit for Creating Animated Graphs

Malchiel Rodrigues, Harvard University
Joel Dapello, Harvard University
Priyan Vaithilingam, Harvard University
Carolina Nobre[◦ *], University of Toronto
Johanna Beyer[◦ †], Harvard University

1 Introduction

Our evaluation approach focuses on evaluating the *learnability*, *usability*, and *expressiveness* of ProtoGraph in a user study. In this Supplementary Material we provide details on our user study (Sec. 2) and study results (Sec. 3).

2 Study Description

The rationale for our study design is given by the fact that a comparative study to current state-of-the-art systems would not be justified. No other tool focuses on the main feature of ProtoGraph, that is, the creation of animated graphs for people with little or no programming experience. The DOT language, for example, does not support animations and is also not well suited for the workflow of non-programmers. DOT's mental model of how a graph is defined is fundamentally different to ProtoGraph and requires users to have the complete graph in mind before they start programming. In ProtoGraph, users can adjust and refine a graph at any point in their program, similarly to instructing a listener who is drawing the graph bit by bit. Users can add extra nodes at any time, without having to go back and adjust their previously written code. The DOT language, its frontend Graphviz, and the JavaScript extension viz-js also do not offer integrated training and teaching for novices, making it difficult to learn for non-programmers.

On the other end of the spectrum of available tools are manual painting tools that do not require programming. However, these tools require a very manual and tedious process for any small change in the graph. For example, to change a graph's layout, a user would have to manually select and move all nodes, all edges, and all labels, making it less than ideal for iterative refinement.

Therefore, in our study, we focus on evaluating the core features of ProtoGraph. To that end, we conducted a crowdsourced user study with 41 participants to evaluate how ProtoGraph can be used by people of varying programming experience. We evaluate the tool's learnability by measuring users' ability to write and read ProtoGraph code after an initial training session. We further use participants' feedback to analyze the tool's usability. The expressiveness of ProtoGraph is evaluated implicitly by the ability to create animated graphs and the wide variety of reading and writing tasks in the study.

2.1 Procedure

We conducted our study with users at all levels of coding experience and asked participants to report their familiarity with coding and graph visualizations. We recruited 41 participants on Prolific, a research-focused crowdsourcing platform. All participants viewed and agreed to an IRB-approved consent form. The estimated duration was set by the completion time of pilot experiments. The study consisted of five phases: *Passive Training, Active Training, Trials, Study, and Demographics and Feedback*. The full study can be viewed at `http://protograph.projectalg.com/tool/study`. **Passive training** consisted of an introduction to graph visualization and a high-level overview of the ProtoGraph interface through a 2-minute video. Participants had to watch the video before being allowed to continue. **Active Training** introduced participants to each part of the interface and taught them basics of the ProtoGraph language in small activities. Participants also had to explore the internal documentation panel to learn parts of the language on their own. In the **Trial**, participants had to use ProtoGraph to recreate a visualization that we provided as an image accompanied with a task checklist. All the knowledge required to pass the trial was provided in the Active Training, this was to verify the participant was attentive. The **Study** itself, included three parts: writing with the ProtoGraph language, reading the ProtoGraph language, and a final free explore task. In the writing portion of the study, the participants were given the image of a visualization or frames of a short visualization animation and asked to recreate the visualization with the ProtoGraph language. In the reading portion, the participants were given an excerpt of code in the ProtoGraph language presented in a read-only version of the ProtoGraph internal code editor and asked to sketch the visualization with the provided digital whiteboard tool. In the final free explore tool, we asked participants to show off their learned knowledge of the ProtoGraph language and to create their own visualization. To conclude the study, participants filled out a **Demographic and Feedback** form.

2.2 Measures

We collected both quantitative and qualitative measures. We captured the start and end times for each task, section, as well as completion time for the entire study session. We also collected the code or sketch submission for each task. We recorded interactions with the internal documentation to assess how often participants were using the documentation to solve the tasks. Finally, we recorded participants' interaction with the tool and study through the open-source "record and replay" JavaScript library rrweb [1], which allowed us to replay each participant's actions. In the *Feedback* form, participants submitted feedback through 7-point Likert scales and free-response questions on the training material, the ProtoGraph language's *learnability*, the ProtoGraph language's *readability*, and the ProtoGraph interface's usability.

When calculating correctness, we segmented each task into the separate sub-tasks needed to perform it and assigned partial credit accordingly. For example, for Task 1 (see Fig. 1a) we graded each of the following components independently: topology, layout, color, and labels. This scoring approach allowed us to analyze which aspects of the language participants struggled with the most. After defining the grading rubric, we scored all participants by two separate graders to ensure the fairness of the scores.

2.3 Tasks

We designed the study tasks to evaluate participant's ability to (1) use the ProtoGraph interface efficiently, (2) write ProtoGraph code to generate static graphs, (3) write ProtoGraph to generate graph animations, and (4) understand the syntax of existing ProtoGraph code. We exemplify a few of these tasks below.

[*]e-mail: cnobre@cs.toronto.edu
[†]e-mail: jbeyer@g.harvard.edu
[◦] indicates equal contribution

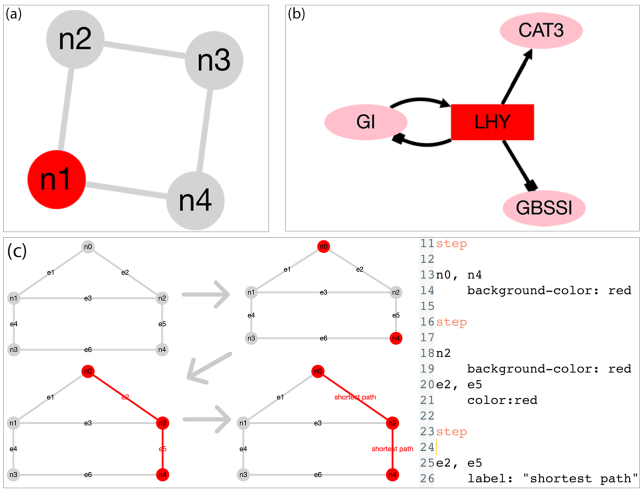


Figure 1: Tasks of Section I (Writing). Participants were asked to write code in the editor to best recreate a given graph visualization. (a) and (b) show the prompts for tasks 1 and 2, (c) shows the prompt as well as part of the solution for task 3 (animation).

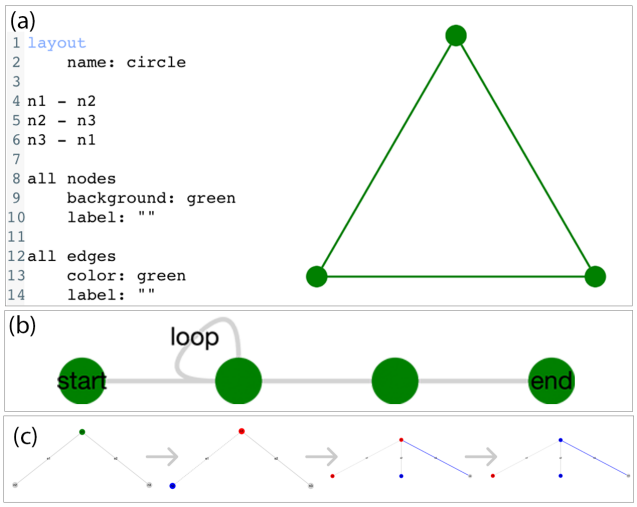


Figure 2: Tasks of Section II (Reading). Participants were asked to read the provided code and use the digital whiteboard to sketch out the corresponding graph. (a) Provided code for task 1 and correct answer. (b) Answer for task 2. In task 3, participants had to answer questions about code for the graph shown in (c).

Section I: Writing. Section I of the study contained three tasks asking participants to write ProtoGraph code to generate static (tasks 1 and 2) and animated (task 3) graphs (see Fig. 1).
Section II: Reading. The reading section of the study contained three tasks. The first two asked participants to read a snippet of ProtoGraph code and use the provided digital whiteboard to draw what they expected the graph to look like. In the third task, we provided code for an animated graph and tested participants' understanding of the animation commands such as the state of the graph at a given timestep (see Fig. 2).
Section III: Free Explore. The final section of the study allowed the participant to use their newly acquired skills to generate a graph or animation with ProtoGraph. This task was meant to capture the expressivity of ProtoGraph, and showcase the variety of graph types and animations that can be created once a user becomes familiar with the basics of the language and interface.

2.4 Pilots and Experimental Planning

We conducted several tests and pilots to evaluate tasks, system usability, data collection modalities, measures, and our procedure. Creating proper training material is a key component for the learnability of ProtoGraph. The pilot iterations were essential in identifying aspects of the training that were unclear, as well as small glitches in the system itself. Participant feedback, as described in Sec. 3.6, shows that participants found the training helpful and the ProtoGraph interface easy to use.

3 Study Results

We recruited 41 participants for this study. After reviewing all submissions, we excluded the responses of 2 participants due to incomplete submissions. This left us with 39 valid submissions (19 identified as female, 19 as male, and 1 chose other). Most participants had little or no coding experience. Here we present an overview of task accuracy and time to completion. We present the tasks according to the aspect of ProtoGraph they were intended to test: reading, writing, usability/free explore. The study results, along with the screen recordings of the study are available at `https://protograph.io/analysis/`.

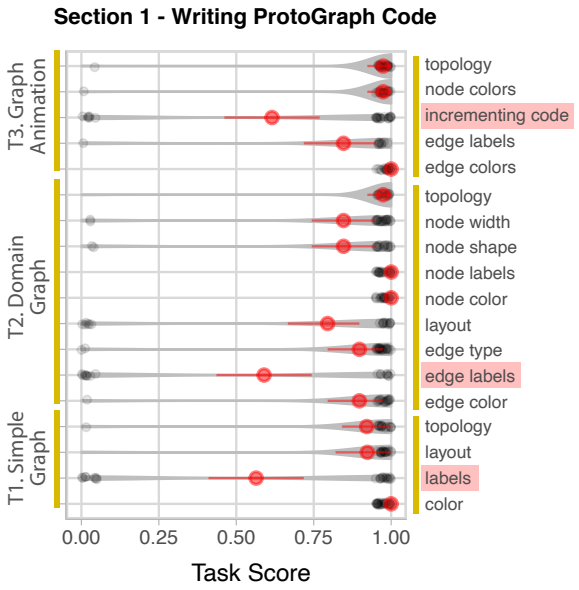
Figure 3a and Figure 3b show the distribution of participant performance for the study tasks in sections 1 (writing) and 2 (reading), respectively. On average, participants scored over 75% on all tasks. Participants scored highest on tasks in the code writing portion, with averages between 80% and 90% accuracy. The lowest average accuracy was on Task 2 of the code reading portion, which required participants to read code and draw the corresponding graph for a more complex network. Figure 4 shows that there is no significant impact of prior coding experience or experience with graph visualization on participants' accuracy, indicating that ProtoGraph is easy to adopt even by non-programmers and novices.

3.1 Section I: Writing

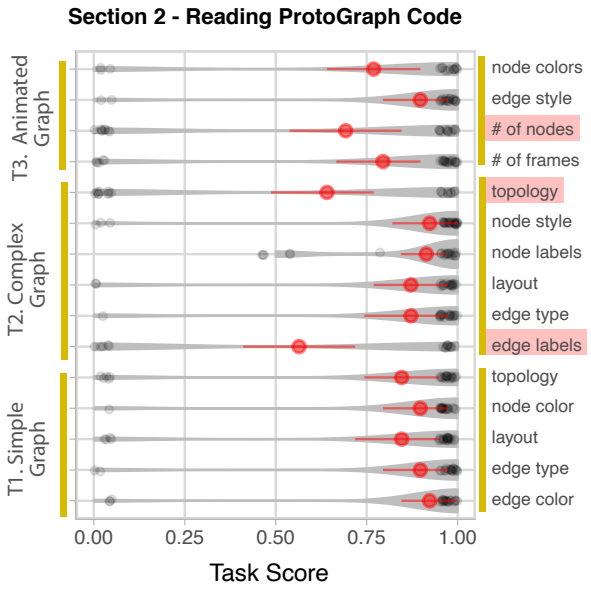
The writing section (Figure 3a) revealed that even with brief training and a short acquaintance period, participants were able to use both the ProtoGraph language and system, validating **R4** the learnability of ProtoGraph. Task 1 showed a high success rate when participants were asked to establish the topology of the graph visualization; this validated that ProtoGraph fulfilled **R1** (specifying graph structure). This requirement was further explored in tasks 1, 2, and 3 where participants were asked to provide visual styling for the graph. We saw that even when the participant was asked to establish specific layouts, they were over 75% successful. Task 2 even required the use of a language extension, which participants were able to achieve with an average success rate over 75%; this validated the **extensibility** of ProtoGraph; we implemented a useful extension that was subsequently used by participants. Though we had initial reservations about the choice of indented colon-separated data and style attributes, the high success rate in specifying color, shape, width, and other visual properties showed that participants, even those with little to no programming experience, were able to successfully use this syntax. Furthermore, considering these properties, specifically labels, we see that as tasks progress the participants were able to achieve the desired target with increasing accuracy; this shows that participants were able to quickly learn parts of the syntax and specific names/keywords. Task 3 showed that participants struggled a little with animations, however, even in this task, the success rate was over 50%. The growth evident in prior steps indicate that participants may have been able to achieve higher success rates if given multiple animation tasks (**R2**).

3.2 Section II: Reading

The reading task evaluated the readability of the ProtoGraph language. While in most cases, a ProtoGraph user will be writing the language, the readability of the language impacts the time it takes for



(a) Participant accuracy scores in code writing.



(b) Participant accuracy scores in code reading.

Figure 3: Participants scored on average above 75% on all but 3 subtasks in writing (a) and 3 subtasks in reading (b) ProtoGraph code (highlighted in red).

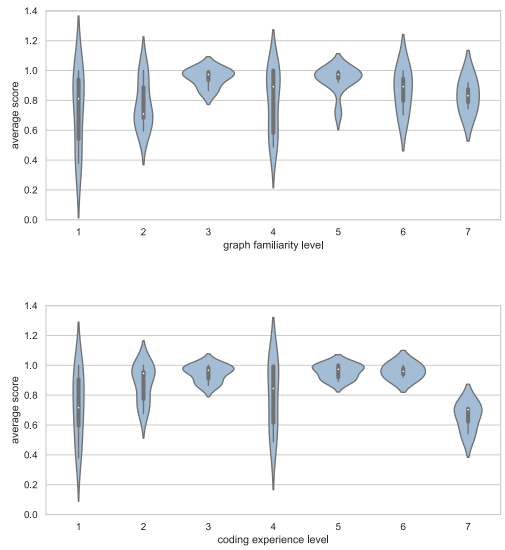


Figure 4: Participant accuracy based on experience with graph visualization (top) and coding (bottom). There was no significant trend in participant accuracy as a function of their previous graph visualization or coding experience.

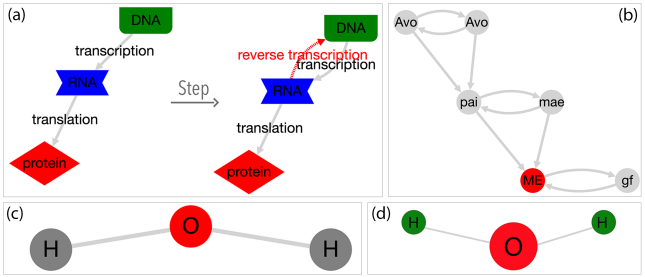


Figure 5: Results from four participants on section III of the study, where we invited them to create their own visualization using the ProtoGraph system. (a) Protein creation process from a participant with visualization experience of 5 out of 7 and coding experience of 5 out 7. (b) Family tree written in Portuguese (participant experience: vis 4/7, coding 1/7). (c) H_2O water molecule (participant experience: vis 3/7, coding 3/7). (d) Water molecule from another participant (participant experience: vis 5/7, coding 3/7).

a user to reacquaint with old code and a more readable language is easier to discuss outside of the ProtoGraph system. The readability is also related to how "natural" the language feels for English speakers. Figure 3b shows detailed results for the reading sub-tasks, where we see that the ProtoGraph language has a syntax that resulted in a high success rate for reading and determining topology, layout, and visual style as shown by tasks 1 and 2. Labels did not seem to have a high success rate in this section of the study. This may have to do with how ProtoGraph was optimized to make writing names easier by allowing them to be specified in the constructor or as a style attribute. The rendering has no issue resolving this and the visual feedback means that users do not struggle with this when writing the ProtoGraph language; however, it may make reading the ProtoGraph language slightly more difficult. On the other hand, the low success rate may also be the result of participants thinking that labels were not as important and other visual properties or a slight increase in effort required to add labels in the sketching system used in the study. Ultimately, this could benefit from further evaluation in the future, but we are satisfied that participants were able to specify labels as shown in Section I. In task 3, we see that participants did not answer perfectly regarding the animation questions; however, they were still able to achieve an average success rate over 75%. This also requires future evaluation as it is not clear if this is an indication of difficulty with the animation syntax or if it is just reflective of an increased working memory strain required to track the changes of an animation across frames with no visual aid (due to our task set-up).

3.3 Section III: Free Explore

The Free Explore section gave participants an opportunity to demonstrate what they had learned in their brief time with the ProtoGraph

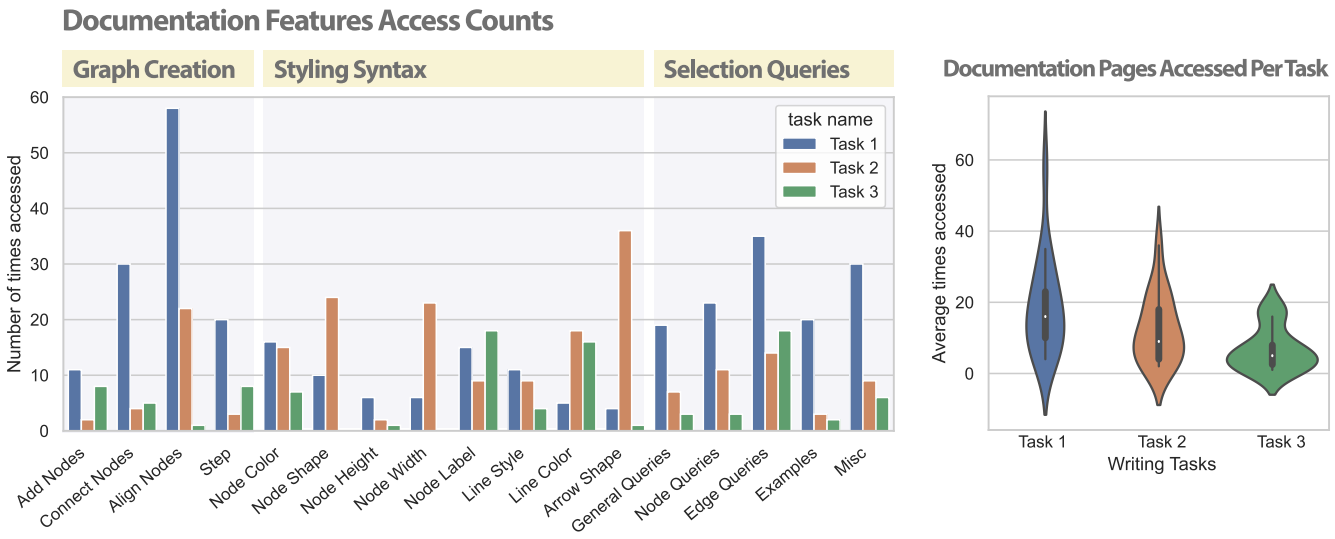


Figure 6: Documentation access count per task. Left: Documentation usage for each feature. Tasks that test specific features correlate with increased documentation accesses for that specific feature (e.g., task 2 uses shape, node width, and arrow shape). Right: Aggregated documentation usage per task. Documentation is accessed less as the participants progress through tasks, suggesting that participants were able to quickly learn the ProtoGraph language. This validates the **readability** requirement of the language.

system. The section allowed participants to create any graph visualization or animation and provided validation that the ProtoGraph system can be used to solve real-world scenarios. Though this section was optional; we saw that almost all of the participants' challenged themselves to use most of what they had learned or try features they had not yet learned. In Fig. 5 we see a few notable examples that show that participants were able to successfully create visualizations that are very similar to the anticipated use cases of the ProtoGraph system. All examples were chosen from participants' submissions in Section III. We hope that the ProtoGraph system can be used as a pedagogical tool and examples (a) through (d) in Fig. 5 show pedagogical use cases. Fig. 5a shows the protein transcription process and fits the scenario where a biology professor or teacher prepares a graphic for a slideshow, worksheet, or textbook. This participant also made use of node shapes that were not introduced in the training showing that they were able to successfully navigate the documentation or autocomplete to find specific styles that they desired; this participant even leveraged the animation syntax of the ProtoGraph language to show the multi-step protein creation process as an animation. As this participant did, an instructor could use the ProtoGraph system to create animations to explain concepts in steps and break down complex processes. Fig. 5b shows a family tree written in Portuguese; this reflects the use case where a student might construct a graph visualization for a homework assignment or presents a visual representation of a historical lineage. Notably, this family tree visualization was made by a participant who rated their prior coding experience as a 1 on a 1 to 7 Likert scale; this highlights the ease of adoption and ease of use of the ProtoGraph system in creating a real-world visualization for someone with little to no programming experience. Visualizations (c) and (d) show that two participants coincidentally had the idea of creating a graph visualization for representing a H_2O molecule; we even see that the second participant (d) used the width and height style properties to resize the "O" (oxygen) node to better represent the size of the atom. This represents the pedagogical use in another field: chemistry; here an instructor or student may use this to prepare chemistry slides or notes.

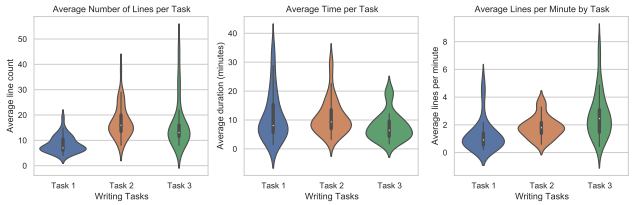


Figure 7: From left to right: Average number of written code lines, task completion times and coding speed per task in Section I (Writing). The average number of lines written per minute (right) increased as participants progressed through tasks, which indicates good learnability (**R4**) and a corresponding increase in efficiency (**R5**).

3.4 General Analysis

The study was a revealing and validating process. While participants did not perform perfectly, the high success rate showed that even in a short amount of time, participants were able to effectively learn and use the ProtoGraph system. In Sec. 3.6, we discuss the end-of-study survey which includes a 7-point Likert scale asking about familiarity with graph visualizations and prior experience with programming. The study included participants with varying levels of programming experience and experience with graph visualizations. Notably, most of our participants had little programming experience and knowledge of graph visualizations. This distribution, allied with the high accuracy rates observed, gives us confidence in the learnability of the ProtoGraph system. Participants were able to complete tasks with high accuracy and increasingly fewer documentation accesses as the study progressed (see Sec. 3.5). They also were able to complete tasks with increasing efficiency as shown by lines written divided by duration. Fig. 7 shows that after an initial learning period which is shown in the slower completion of task 1, users were able to complete tasks 2 and 3 significantly faster. This jump in writing speed was so drastic that between tasks 1 and 2 participants were on average able to write double the lines of code in the same time or less, and their efficiency further increased from tasks 2 to 3. This reveals that participants were able to quickly get acquainted with the ProtoGraph system, indicating learnability **R4** and that participants

were able to greatly increase their speed as they spent time with the system, validating our efficiency requirement **R5**.

3.5 Documentation Usage

The documentation usage data we collected during the study provided us great insight into the experience of the participants. This usage showed that participants were able to approach a new syntax and with very brief and concise documentation, they were able to learn and implement the syntax; this further validates the learnability and approachability of the ProtoGraph language. We saw that when participants were asked to use visual properties that they were not explicitly introduced to in the active training, they were able to successfully use the documentation to learn and use the specific style. As a tool, the documentation paired with the autocomplete reduced errors and reduced the need for perfect recall in the participant. Fig. 6 shows the documentation usage per location/page separated by task in the writing section. Documentation usage for language fragments that are present in each task (e.g., "connect", "node color", and queries) declined as tasks progressed. This suggests that participants were able to quickly learn core aspects of the ProtoGraph language, thus reducing the need to check the documentation (**R4**). This is even more evident with "Document pages accessed per task" in Fig. 6 which shows a decrease in overall documentation access per task.

3.6 Participant Feedback

The ProtoGraph system is a user-first system designed to help users quickly and effectively make graph visualizations and animations. Therefore, we asked each participant to complete a short survey at the end of the study. We asked participants about their prior familiarity with graph visualizations and programming experience. We also asked them to rate and give feedback on the ProtoGraph system about usefulness, learnability, readability, and usability.

We collected answers on a 7-point Likert scale and a free-response field for any comments. The vast majority of users rated the ProtoGraph system as highly useful, learnable, readable, and usable. Comments from participants with self-reported coding experiences of 1 and 2 included: "The protoGraph language was easy to read because it uses day to day words". "For someone that has little experience with code it is simple to read it", "After learning to use it it is very easy to remember the commands and use them." This feedback highlights the approachability and ease of use for non-programmers. On the other end of the spectrum, comments among participants who reported coding experiences of 6 or higher included "Found it very similar to other languages so it was easy to understand", "It made creating graphs very simple yet fun. Very accessible language rules and easy to understand documentation.", "The language is well-structured into "blocks" that are easy and pleasant to the eye. The names of the commands and attributes are intuitive, so the user can easily understand and remember them." , "The interface is user friendly, everything is where you can see it and easy to find. Also, when an error has been made, the user is notified immediately which is efficient." Ultimately, the feedback shows that ProtoGraph was well received by participants at all levels of coding experience.

REFERENCES

[1] Rrweb. https://github.com/rrweb-io/rrweb.