In D. Reitter & F. E. Ritter (Eds.), Proceedings of the 14th International Conference on Cognitive Modeling (ICCM 2016). University Park, PA: Penn State.

JIVUI: JavaScript Interface for Visualization of User Interaction

Ignacio X. Domínguez (ignacioxd@ncsu.edu), **Jayant Dhawan (jdhawan2@ncsu.edu)**,
Robert St. Amant (stamant@csc.ncsu.edu), **David L. Roberts (robertsd@csc.ncsu.edu)**
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206 USA

Abstract

In this paper we describe the *JavaScript Interface for Visualization of User Interaction* (JIVUI): a modular, Web-based, and customizable visualization tool that shows an animation of the trace of a user interaction with a graphical interface, or of predictions made by cognitive models of user interaction. Any combination of gaze, mouse, and keyboard data can be reproduced within a user-provided interface. Although customizable, the tool includes a series of plug-ins to support common visualization tasks, including a timeline of input device events and perceptual and cognitive operators based on the Model Human Processor and TYPIST. We talk about our use of this tool to support hypothesis generation, assumption validation, and to guide our modeling efforts.

Keywords: Typing; Cognitive Modeling; Visualization; Replay; Tool; Data Visualization; Input Device; Cognition; Inter-keystroke Interval; Mouse Clicks; Mouse Motion; Eye-tracking; Gaze Data; Human-Computer Interaction; Human Information Processing; Human Behavior.

Introduction

Visually representing data is a common technique used to a) succinctly summarize and communicate information, and b) gain a better understanding of the process that generated said data. In its most common form—charts—data visualization is generally used for the former. For the latter, however, visualizations are generally more complex and are built ad-hoc according to the originating data's context. This results in an increased effort required to visualize similar, but not identical datasets.

Examples of data that are generally visualized to be better understood come from cognitive modeling and empirical user interaction with graphical user interfaces. In both cases, the data correspond to a particular context (*e.g.*, the user interface being used or modelled), to a particular set of input devices (*e.g.*, mice, keyboards, eye-trackers, trackpads, etc.), and has a particular set of spatial and temporal properties (*e.g.*, locations on the screen or relative to on-screen items, time elapsed between keyboard or mouse events, etc). A consequence of this need for specialization is a reduced number of available tools that are suitable to visualize a specific dataset. In cases where there exists a suitable tool, an added inconvenience is that it may not support the researcher's preferred operating system, or it may require the installation of dependencies that the researcher may prefer not to install.

In this paper we present the JavaScript Interface for Visualization of User Interaction (JIVUI)—a modular, Web-based, and customizable tool that addresses these problems, providing the ability to visualize and replay a user's interaction with a graphical user interface on any platform through any modern Web browser.

Data produced from cognitive models of input device usage is generally similar to that collected from a user, with the addition of annotations for the cognitive operators related to the user's observed interactions. Through a flexible data representation, JIVUI supports both. In addition to user and/or model data, JIVUI is capable of rendering a replay of the interaction over a customizable UI that can be crafted to represent the task from which the data was collected.

Related and Prior Work

Visualization tools are a staple in cognitive modeling research. Models can grow to be very complex, and understanding the subtleties of their behavior in carrying out specific tasks often requires more than textual descriptions or traces, tables of quantitative trace information, or summary statistics. A variety of tools are available for presenting models and user data in graphical form.

The ACT-R 6.0 Environment (Bothell, 2004) provides controls and information about a model in a graphical user interface. A stepper function is described as the most useful tool in the environment. It is comparable to a stepper in a conventional programming environment, supporting pauses before events and a range of choices for running a model until a specified condition becomes true (when a given duration is exceeded, a given production fires, or an event is generated by a given module). A graphical trace is provided, with time along one axis and events of different types in rows, filling the other axis. The environment can also display a state chart diagram, a directed graph showing the productions selected and fired in a model. Other, more specialized visualizations are also supported.

SANLab-CM (Patton & Gray, 2010) is a tool for activity network modeling, specifically models that include stochastic operators. SANLab-CM gives a modeler the ability to construct and edit a model in the form of a specialized directed graph that captures elementary cognitive, perceptual, and motor operators and the dependencies between them, as well as compositions called interactive routines. The modeler can visualize the model as a Gantt chart that shows the execution of operators over time; a histogram shows the distribution of model execution times. By selecting a specific critical path, the modeler can focus the visualization on the associated subset of operators.

NAV (Kriete, House, Bodenheimer, & Noelle, 2005) is a tool for generating animations of cognitive model execution, showing different visualizations to help non-expert users understand the dynamics of a model. The focus is on giving a

Figure 1: Example of replay and control interfaces illustrating the playback of typing and gaze data.

modeler the facilities needed to build a presentation-quality animation to convey the necessary information. For example, in a semantic network visually represented as nodes and arcs, the modeler can opt to show node activation levels over time by changing color or size. Annotations can be added to the visualization, which can also change over time.

SIMCog-JS (Simplified Interfacing for Modeling Cognition - JavaScript) (Halverson, Reynolds, & Blaha, 2015) addresses the continuing challenge of building cognitive models that can interact with external software, in this case the interaction between Java ACT-R and HTML/JavaScript. SIMCog-JS does not provide visualization facilities per se. Due to its integration with JavaScript, however, SIMCog-JS makes it very easy to instrument a Web interface to replay the behavior of a user interacting with the interface or to simulate a cognitive model using the interface—*e.g.* Dong and St. Amant (2016). One novel use of SIMCog-JS in this way has been toward the visualization of eye movements (Balint, Reynolds, Blaha, & Halverson, 2015). A Model Visualization Dashboard can play the trace of a model performing tasks in a given interface, along with other types of visualizations. Further, the environment contains an embodied virtual character that simulates eye and head movements predicted for a human being carrying out tasks in the interface.

An Overview of JIVUI

JIVUI is a Web framework designed to simplify the visualization of gaze, keyboard, and mouse data, and can visually represent perceptual and cognitive operators associated with the data on a timeline. It supports data generated by a cognitive model as well as data collected empirically. The data is provided to JIVUI in JSON (ECMA International, 2013) format either in the page's source or by loading it through the Web interface. The JSON structure and data flow are explained in more detail in the **JSON Data Description** section.

We designed JIVUI to run completely in a Web browser without the need for a back-end, making it immediately usable on any platform. A typical JIVUI instance will render a Web page with a replay interface, a playback control interface, and a timeline. The replay interface, illustrated at the top of Figure 1, renders the animated user interaction over a user interface that can be customized to look and behave like the experimental environment seen by participants. The playback control interface, illustrated at the bottom of Figure 1, is used to start, pause, and stop the replay animation, as well as to specify the playback speed and to move forward and backward in the animation. Finally, the timeline, shown in Figure 2, will display interaction events such as keystrokes, clicks, and gaze fixations, but can also display perceptual and cognitive operators associated with the interaction events based on a cognitive model.

JIVUI is designed to be extensible, meaning that all of the components described above are provided as plug-ins that can be modified or replaced without modifying JIVUI's core. This plug-in-based architecture allows the visualization experience to be highly customizable to support a wide range of user interfaces and data attributes. JIVUI's extensibility is made possible through its support of plug-ins for almost all its functionality, as described in the **JIVUI's Architecture** section.

JSON Data Description

JIVUI works with millisecond precision. It expects a JSON string containing a "`settings`" object and a "`data`" object (an example is shown in Listing 1). The "`settings`" object is required to contain a "`start`" numeric attribute indicating the smallest millisecond contained in the data, and an "`end`" numeric attribute indicating the largest millisecond contained in the data. It can also contain three optional attributes, "`title`", "`startOffset`" and "`endOffset`", providing a label to the data, time padding at the beginning, and time padding at the end of the visualization, respectively.

The "`data`" object is where user or model data will be provided. It is expected to contain an arbitrary number of entries where the keys are milliseconds. Each entry can contain any combination of event types that occurred at that millisecond, keyed by the event type as follows: "`click`" for mouse clicks, "`key`" for keystrokes, "`mouse`" for mouse position, and "`gaze`" for eye-tracking data. In turn, each of these event types will include a set of required and optional attributes. Because multiple clicks can occur at the same time (*e.g.* from different mouse buttons), a "`click`" event is an array where each element is expected to contain the `x` and `y` coordinates where the click happened, as well as the button that was pressed (*e.g.* "left", "right", or "middle"), and it can also contain an optional "`duration`" attribute indicating for how long was the mouse button pressed. If not specified, a mouse click is assumed to last 100ms for visualization purposes. Multiple keystrokes can also occur at the same time, so a "`key`" event is also an array where each element is expected to contain the key that was pressed, and can also contain optional attributes for the "`duration`" of the keystroke, whether

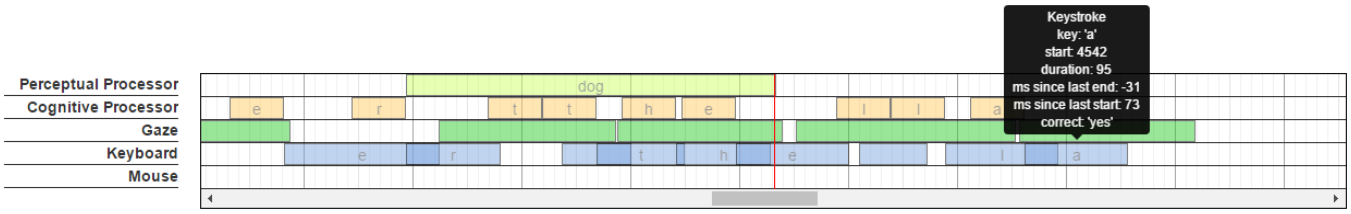

Figure 2: Timeline

```
{
  "settings": {
    "title": "Participant 31", // optional
    "start": 200,
    "end": 250,
    "startOffset": 100,        // optional
    "endOffset": 200           // optional
  },
  "data": {
    "200": {
      "click": [{
        "button": "left",
        "x": 200, "y": 150,
        "duration": 60         // optional
      }],
      "key": [{
        "key": "t",
        "duration": 88,        // optional
        "correct": true,       // optional
        "word": "the"          // optional
      }]
    },
    "250": {
      "gaze": {
        "x": 220, "y": 170,
        "fixated": false       // calculated
      },
      "mouse": {
        "x": 210, "y": 151,
        "drag": true           // calculated
      }
    }
  }
}
```

Listing 1: Example of simple input data in JSON format.

the keystroke correctly matched an expected keystroke (useful for transcription typing, or when the expected keystroke is otherwise known), and a word context where the keystroke occurred. If not specified, a keystroke is assumed to last 100ms for visualization purposes. The "`gaze`" and "`mouse`" events are expected to contain the `x` and `y` screen coordinates of the position of the eye-tracking and mouse pointer data, respectively.

Following the JSON standard, this data representation provides a few advantages. First, it guarantees that there is only one data entry per millisecond. Second, it guarantees that every entry contains at most one instance of each event type. Third, attributes and objects can be added to the data either before it is input to JIVUI, or dynamically by plug-ins, as discussed in the **PLUG-INS** section. More importantly, this representation is a string that can be easily produced from empirical data as well as from the output of cognitive modeling tools, and is supported by virtually all programming languages either natively or through popular libraries.

JIVUI's Architecture

At its core, JIVUI consists of a timing engine, a state manager, and a plug-in manager. The timing engine is used on playback to maintain a stable animation framerate regardless of any browser's constraints or performance limitations. Because JIVUI works with millisecond precision, the default framerate is 1000 frames per second (FPS) as an attempt to render the visualization in real-time. This value is configurable, even as an animation is being played. When the timing engine cannot render at the specified speed, it provides the number of frames that are lagging behind. This allows JIVUI to compensate for this lost time to ensure that the animation has accurate timing based on the set speed and the data's total duration.

As the name suggests, the state manager keeps track of JIVUI's state, which includes the currently loaded dataset (if any), the playback speed, the current frame, and whether the animation is playing, paused, or stopped. The state manager also provides an API to control an animation. Specifically, through the state manager a module can control the animation speed, advance and rewind to a specific frame, and can play, pause, and stop the animation.

The plug-in manager maintains a list of registered plug-ins, and provides an API to register and remove them.

Plug-ins

JIVUI supports two types of plug-ins: preprocessors and UI modules. We designed the plug-in API so that a single component could serve as a preprocessor and a UI module in the interest of maximizing JIVUI's applicability to varied use cases. Each plug-in is described by a JSON file that lists all the resources that it depends on, such as JavaScript files, HTML templates, and/or CSS files.

Preprocessors are exclusively invoked on data load and their main task is to augment the data by performing calculations on it. For example, a preprocessor can be used on gaze data to determine fixations, or work with mouse motion and click data to add attributes that indicate when the mouse motion is occurring within the context of a drag operation.

As shown in Figure 3, JIVUI invokes every preprocessor by calling their "`onDataLoaded(settings, data)`" method,

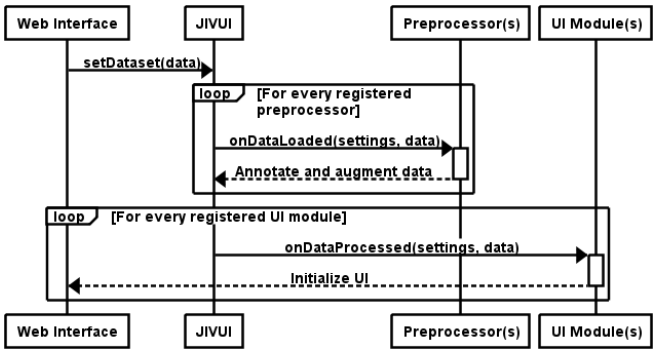

Figure 3: Sequence diagram of the data loading phase, where preprocessors annotate and augment the data, and UI modules initialize the Web interface.

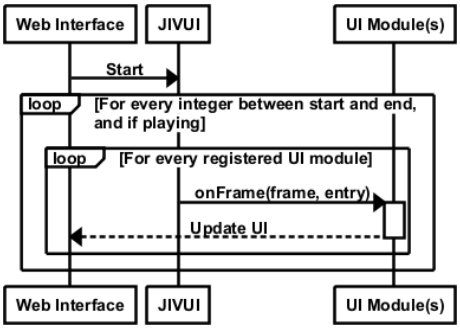

Figure 4: Sequence diagram of the playback phase, where UI modules are provided frame and event data to be rendered.

providing the dataset's "`settings`" and "`data`" objects as arguments. This tells every preprocessor that a new dataset has been loaded and allows them augment data entries.

UI modules are responsible for rendering elements on the Web page and providing functionality to the end user. They are initialized on data load, and are invoked by JIVUI during playback, following the process illustrated in Figure 4. Examples of UI modules are a replay area to visually reproduce keystrokes and mouse motion, playback controls, and timelines.

JIVUI's Included Plug-ins

In order to make JIVUI usable out of the box, and to showcase its potential, we include with it a set of plug-ins that are suitable for the most common user interaction cases. We include basic preprocessors for gaze, keyboard, and mouse data, as well as a cognitive preprocessor. For UI modules, we include a replay region, a set of animation controls, and a timeline.

Preprocessor Modules

Gaze Preprocessors: Two gaze preprocessors are included. One determines fixations using an implementation of the I-DT algorithm as described by Salvucci and Goldberg (2000). For every gaze event, it adds a boolean attribute "`fixated`". The second one uses an implementation of the 1€ filter (Casiez, Roussel, & Vogel, 2012) to stabilize noisy gaze data, adding numeric "`filteredX`" and "`filteredY`" attributes with smoothed eye position values.

Mouse Preprocessor: The included mouse preprocessor augments mouse motion data by adding a boolean "`dragged`" attribute to every mouse motion event. When a mouse motion event occurs within the duration window of a click, as specified by the click's "`duration`" attribute, this value is set to true; otherwise it is set to false.

Keyboard Preprocessor: The keyboard preprocessor augments keystroke events by adding the number of milliseconds elapsed between the start of the current keystroke and the end of the previous one, as well as between the start of the current keystroke and the start of the previous one.

Cognitive Preprocessor: This preprocessor annotates keystroke data with cognitive and perceptual operators based on the Model Human Processor (MHP) (Card, Moran, & Newell, 1986) and the TYPIST model (John, 1996). For every keystroke, this preprocessor creates a cognitive operator that precedes it, lasting 50ms. Similarly, before the first cognitive operator of every word, this preprocessor includes another cognitive operator that also lasts 50ms. Also for every word, a 340ms perceptual operator is created. The TYPIST model describes a working memory capacity of three words, thus perceptual operators for three words are created by this preprocessor at the beginning of the timeline; it waits for the typing of the first word to be completed before creating the perceptual operator for the next word, and so on.

UI Modules

Replay Region: This UI module instantiates a text area where keystrokes appear as key events are received. It also loads two additional UI modules: one to overlay gaze data and the other to overlay mouse motion and clicks. The gaze and mouse UI modules consist of a "`canvas`" element each, and render gaze and mouse motion as moving dots overlaid on the interface. Click events appear as two concentric circles. To improve visibility, the eye and mouse indicators are rendered in different colors that contrast with the background, and are configured to leave a "trail" that fades away as the animation progresses.

Animation Controls: This UI module is used to manipulate the animation. It provides controls to move the animation forward or backward to any particular frame, buttons to start, stop and pause the animation, and controls to set the animation's speed. As the animation is played, this plug-in updates the current millisecond being visualized to reflect the current frame in the animation. It also provides two keyboard shortcuts to control the animation: pressing the space bar will play and pause the animation, while pressing the backspace key will stop it.

Timeline: The included timeline plug-in is designed to look for the augmented data provided by the preprocessors described above and displays events and operators in five different tracks: gaze, mouse, keyboard, cognitive, and perceptual, as shown in Figure 2. Events on each track are displayed as boxes with their length based on the event's duration, and

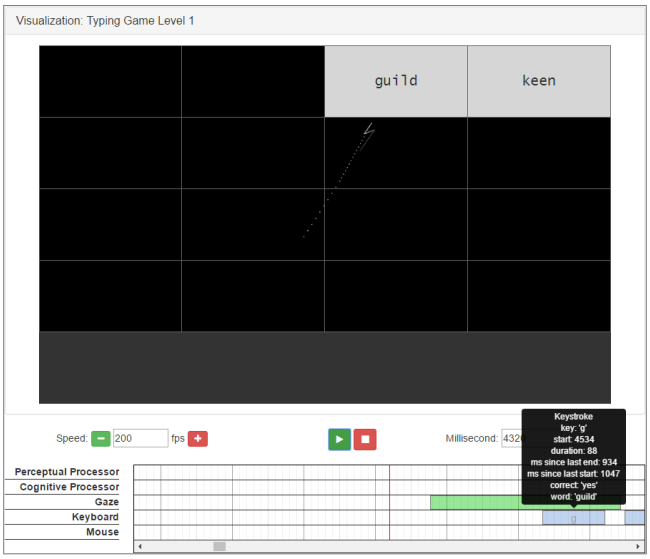

Figure 5: Example of JIVUI being used to visualize gaze and keystroke data collected from *The Typing Game*.

can sometimes overlap. The gaze track displays fixations, the mouse track displays clicks, and the keyboard track displays keystrokes. The cognitive and perceptual tracks display their corresponding operators. Each element on the timeline displays a tooltip on mouse hover providing more information about the event, such as the millisecond value when it occurred, its duration, and other corresponding data (*e.g.*, position of a click, key pressed on a keystroke, etc.), as well as annotations provided by preprocessors, such as the time elapsed after the previous event of the same type.

JIVUI in Use

We used JIVUI to visualize the data we collected from a study involving a computer game, which consisted of participants' typing and gaze data. We were able to replay players' experiences from trace data as a simulation of their participation. A screenshot of one of the play traces is shown in Figure 5. As our study only collected participants' typing and gaze data, we used every plug-in included with JIVUI with the exception of the mouse preprocessor. In this case, we used a modified version of the replay region to mimic the game's user interface, and to respond appropriately to the simulated key presses as if it were the real game.

The use of JIVUI facilitated hypothesis generation for our study. The visualization helped us quickly identify touch typists among our participants, and to generally assess their typing skill, by looking at how often a participant's gaze suddenly dropped off the bottom of the screen, indicating that the participant was looking at the keyboard while typing.

One slightly unexpected discovery in our use of JIVUI came in the examination of typing patterns in the typing game. Participants were asked to type the sentence, "The quick brown fox jumps over the lazy dog," so that the game could be adjusted to their approximate typing speed. Our initial assumption was that this sentence would be typed in a similar way to transcription typing. Following the TYPIST model of transcription typing, we would expect a visual attention shift to a word to occur several keystrokes before the first letter in the word. Possibly because of the way the interface is set up, with the text to be typed visible on the screen, overwritten by gray characters as each is correctly typed, a different pattern typically emerged: the visualization showed that their gaze tended to stay on a word until the word was almost completely typed. We judge that without the animated replay provided by the visualization, it would have taken much longer for us to realize that one of our basic assumptions (the appropriateness of the model) was incorrect.

As a result, we have a new ACT-R model (still under development) with tighter constraints between perceptual and motor processing. In the model, a visual attention shift (including eye movements, using the EMMA (Salvucci, 2000) extension to ACT-R) occurs at the end of each word. The model under-predicts the elapsed time between the fixation on a word and the first keystroke for that word, at around 150 ms, where the elapsed time for participants is closer to 400 ms. Despite this, the model does preserve a general pattern, a relatively small variance in the distribution of elapsed times between fixations and the first keystrokes of words. Our modeling effort in this area continues.

JIVUI also allowed us to assess the quality of our gaze data, helping us identify eye tracker calibration biases, and poor gaze data in general. Being able to visualize calibration biases is incredibly helpful as a guide to correct a participant's gaze data, making it usable, whereas it would have otherwise been discarded.

As a Web tool, we deployed JIVUI on a centralized server allowing multiple researchers to use it simultaneously and on the same dataset. Some of our ongoing efforts using JIVUI to visualize our game data include identifying word segmentation. One approach for doing so is to allow multiple people to visualize our data and manually label segmentation boundaries, resulting in a dataset from which agreement metrics could be computed to determine word segments.

To showcase JIVUI's versatility, we also show in Figure 6 a visualization of data we collected from another game. In this case we visualize mouse motion (overlaid on the game screen) and mouse clicks (as timeline events.

Discussion and Conclusion

The main contribution of this paper is JIVUI itself, which is being made available online[1]. In addition to the source code, we are also providing basic documentation, example data files, and a working demo.

The most salient advantages of JIVUI are its flexible data format, its extensibility, and its portability. With minimal expectations on data structure, JIVUI can be used with both simple and complex datasets, where extended attributes could be easily added to configure the visualization (through the

[1] http://go.ncsu.edu/jivui

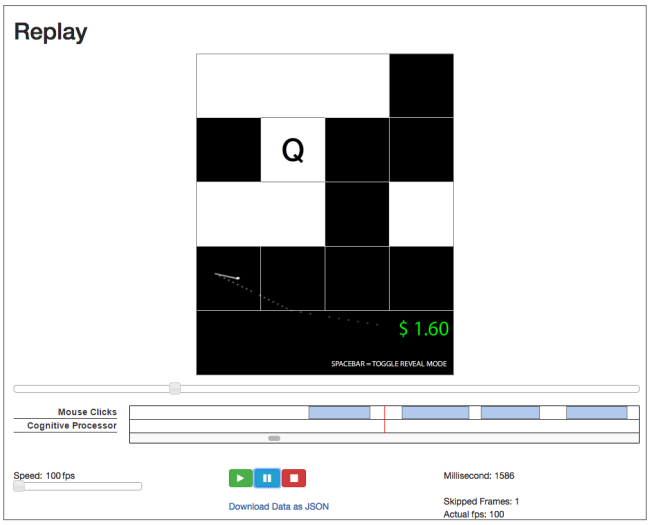

Figure 6: Example of JIVUI being used to visualize mouse motion and click data collected from *The Concentration Game*.

"`settings`" object), and to provide a richer dataset (by adding elements to the data entries). This allows JIVUI to support input data generated from different cognitive modeling methods, as well as data collected from a real user.

JIVUI's heavy reliance on plug-ins allows it to be highly customizable not just to support a wide range of data attributes to visualize, but also to perform calculations on the data and to completely customize how the data is finally played back in the Web interface. Additionally, JIVUI's included plug-ins are designed to accommodate the most common user interaction visualization use cases, making it approachable for newcomers. The included plug-ins also serve as examples for end users to build upon when creating more complex preprocessors and UI modules.

Because JIVUI is based entirely on Web standards, it requires only a modern Web browser to run. This allows JIVUI to be platform-agnostic. It can be hosted on a local computer or on a server, allowing a single instance to be used by multiple people simultaneously. JIVUI does not require any specific dependencies and can be hosted on any Web server. With appropriate UI modules, JIVUI can be used to visualize data on mobile devices as well.

In addition to presenting the tool itself, we have also provided a few examples of how JIVUI has helped us visualize and obtain insight into data collected from our experiments. It can also be applied to visually compare predicted user behavior as cognitive models with actual user data.

Acknowledgments

This work was funded in part by the National Security Agency under grant number H98230-14-C-0139 and in part by the National Science Foundation under grant number IIS-1451172.

References

Balint, J. T., Reynolds, B., Blaha, L. M., & Halverson, T. (2015). Visualizing eye movements in formal cognitive models. In *Workshop on eye tracking and visualization.*

Bothell, D. (2004). Act-r environment manual (working draft) [Computer software manual]. (Downloaded March 2016.)

Card, S. K., Moran, T. P., & Newell, A. (1986). The model human processor: An engineering model of human performance. *Handbook of Perception and Human Performance*, *2*, 1–35.

Casiez, G., Roussel, N., & Vogel, D. (2012). 1€ filter: A simple speed-based low-pass filter for noisy input in interactive systems. In *Proceedings of the 2012 acm conference on human factors in computing systems (chi 2012)* (pp. 2527–2530).

Dong, L., & St. Amant, R. (2016). Answering questions with line and bar graphs. In *International conference on social computing, behavioral-cultural modeling, & prediction and behavior representation in modeling and simulation.* (To appear.)

ECMA International. (2013). *Standard ECMA–404: The JSON Data Interchange Format* (1st ed.).

Halverson, T., Reynolds, B., & Blaha, L. (2015). Simcog-js: Simplified interfacing for modeling cognition - javascript. In *Proceedings of the international conference on cognitive modeling* (pp. 39–44).

John, B. E. (1996). Typist: A theory of performance in skilled typing. *Human-Computer Interaction*, *11*(4), 321–355.

Kriete, T., House, M., Bodenheimer, B., & Noelle, D. C. (2005). Nav: A tool for producing presentation quality animations of graphical cognitive model dynamics. *Behavior research methods*, *37*(2), 335–339.

Patton, E. W., & Gray, W. D. (2010). Sanlab-cm: A tool for incorporating stochastic operations into activity network modeling. *Behavior Research Methods*, *42*(3), 877–883.

Salvucci, D. D. (2000). A model of eye movements and visual attention. In *Proceedings of the 2000 international conference on cognitive modeling* (pp. 252–259).

Salvucci, D. D., & Goldberg, J. H. (2000). Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 symposium on eye tracking research & applications* (pp. 71–78). doi: 10.1145/355017.355028