Integrating Droplet into Applab – Improving The Usability of a Blocks-Based Text Editor

David Anthony Bau
Phillips Exeter Academy
Exeter, New Hampshire 03833
Email: dbau@exeter.edu

***Abstract*—Droplet is a new programming editor that allows dual-mode editing in blocks and text for any text program. This paper presents observations and improvements to Droplet based on integrating Droplet into Applab, Code.org's JavaScript sandbox learning environment. Droplet's unique interactions with both text and blocks create several unusual problems and opportunities for improvement.**

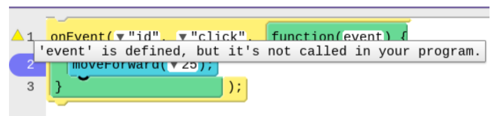

Fig. 1. An Example of Droplet Gutter Decorations in Applab

I. Introduction

Droplet [1] is a new programming editor that allows dual-mode editing in blocks and text for any text program. This paper presents observations and improvements to Droplet based on integrating Droplet into Applab, Code.org's JavaScript sandbox learning environment. While integrating Droplet into Applab the author encountered five problems with Droplet affecting its usability.

1) Pointers to blocks were lost upon Droplet's frequent reparses of blocks. A clean model for serializing and maintaining the locations of blocks in the document would resolve this problem and also allow for a full undo stack.
2) Droplet could not take advantage of the support for live linting errors and warnings and debug breakpoints that already existed in Applab's text mode.
3) Droplet did not have support for dropdowns for sockets, unlike other major block languages.
4) Droplet sockets often contained important work, and it was easy to lose that work when accidentally dropping a block into that socket.
5) Droplet could not support the two-dimensional side-by-side view of block that other major block languages do, because this view is difficult to relate to text code.

In this paper the author presents solutions to these five problems in the context of integrating Droplet into Applab, and comparing Droplet's approach to major block languages like Scratch [7] and Blockly [8].

II. A Clean Locations Model

A. Motivation

Droplet, in accordance with its text-first philosophy, reparses blocks whenever they are changed, which means that pointers to blocks were often lost. Droplet needed a clean model for serializing and maintaining the locations of blocks in the document. Creating this model would also allow Droplet to support a full undo stack, which other major block languages currently do not support. Undo stacks are important to usability, and are included in Neilsen's widely-recognized user interface heuristics [6].

B. Related Work

Droplet's issue with losing pointers to blocks is unique to its need to reparse. Neither Scratch nor Blockly implements a full undo stack. Blockly has an undo stack in progress based on the serialized operations it will make for real-time collaboration. Scratch does not have an undo stack, but has support for ”undeleting” the last block that was deleted.

C. Solution

Droplet now supports two different locations models: a token-based locations model identifying blocks by their token offset from the beginning of the document, and a text-based locations model identifying blocks by their character offset, type, and string length. The text-based locations model is more human-readable but is slightly ambiguous. Editing entities like the cursor, focused socket for text input, and selection are stored as token-based locations.

Droplet now also confines mutations to three fundamental operations: insert, delete, and replace (which is used for reparsing). When an insert or delete occurs, locations are preserved by temporarily getting a pointer to the block at the location, doing the mutation, and then reserializing the new location of the block after the mutation. When a replace occurs, locations are preserved by temporarily converting locations inside the replaced area to text locations, then converting them back afterward.

Each of these mutation operations now also generates an undo operation based on token-based locations.

III. Breakpoints and Line Annotations

A. Motivation

Applab had existing support for live errors and warnings and debugging breakpoints in text mode. Because Droplet

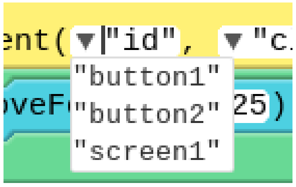

Fig. 2. An Example of Droplet Dropdowns in Applab

blocks have a one-to-one relationship with text code, adding breakpoint and live line-annotation support to Droplet could easily take advantage of Applab's existing debugging infrastructure. Line breakpoints and live annotations are a part of most major professional development environments, including Applab's text mode and Eclipse [2]. A study by Murphy [5] found that over 70% of Eclipse users use breakpoints. In 1986 Baecker [4] proposed "Metatext" or annotations as one of the five main principles of program visualization.

B. Related Work

This is a unique opportunity for Droplet, because it has a one-to-one relationship to text code. Neither Scratch nor Blockly have support for line annotations or line breakpoints.

C. Solution

Droplet now supports breakpoints and annotations in the gutter the same way major text editors do (fig 1). Droplet mimics Ace editor's API to allow Applab and other embedders to easily convert their existing debugging infrastructure from Ace editor to Droplet.

IV. Dropdowns in Text-Generated Blocks

A. Motivation

Because all Droplet blocks are generated from text, Droplet did not have good support for dropdowns from sockets, which other major block languages do. Dropdowns, like autocomplete in text code, help students remember what parameters are valid, in accordinance with Neilsen's heuristic of recognition vs. recall [6].

B. Related Work

Both Scratch and Blockly implement dropdowns for their text inputs. Both have special selectors for colors, allowing users to use a color picker or to "eyedrop" existing pixels on the screen.

C. Solution

Droplet added new configuration to allow the embedding application layer to specify dropdowns. Embedders may specify dropdowns by function name and argument position in JavaScript and CoffeeScript mode – for instance, in Figure 2, the "fd" function has a dropdown specified at argument 0. Dropdowns can be dynamically generated – in Figure 2, this list is generated from elements that the user has placed on the screen in Applab's WYSIWYG Design Mode.

V. Remembering Old Socket Values

A. Motivation

Unlike in a text editor, it is easy to accidentally drop Droplet blocks into the wrong socket. However, because Droplet is text-based, sockets often contain important information like variable names or long strings. Other major block languages do not have this problem because sockets with information like variable names or strings are not usually drop targets for other blocks.

B. Related Work

Blockly does not have this problem, because most constants that could be long, sensitive work are not also drop targets for other blocks. In Blockly, typing socket, which is round, is different from a droppable socket, which looks like a puzzle piece, so work is not lost when an accidental drop occurs.

Scratch has this problem, but it is less pronounced, since sockets that contain strings are not drop targets. Scratch repopulates sockets with hand-chosen default values when blocks are dragged out of them.

C. Solution

Droplet now remembers the old values of a socket when a block is dropped into it, so that when the block is dragged back out the socket is repopulated with the old value (fig 3). This prevents students from losing work by accidentally dropping blocks into the wrong socket.

Because Droplet frequently reparses blocks, attaching the remembered value data directly to the socket is not possible. Instead, Droplet maintains a map from socket locations to remembered values. By preserving the location of the socket (using the same infrastructure as the new undo stack), the Droplet editor can track the socket and preserve the remembered value this until the block is deleted. The map from sockets to remembered values is also tracked in the undo stack, so the remembered values are preserved across undo and redo.

VI. Relating Floating Blocks to Text Code

A. Motivation

A study by Weintrop [3] found that being able to see blocks side-by-side anywhere on the screen was beneficial to students using Scratch. It allowed students to try out different ways of performing the same task, and also allowed them to compose programs in a "non-linear" way. Maloney etl al. [9] term "tinkerability" in their initial presentation of the Scratch programming environment, and say that it supports "a bottom-up approach to writing scripts where small chunks of code are assembled and tested, then combined into larger units." in This kind of view, however, is unfriendly to the linear layout of text code. Droplet needed a way to support assembling and viewing code side-by-side while retaining a one-to-one relationship to text code.

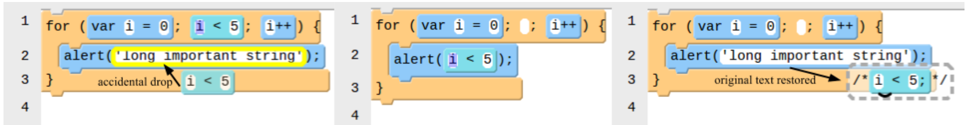

Fig. 3. An Example of Droplet Restoring Old Socket Values

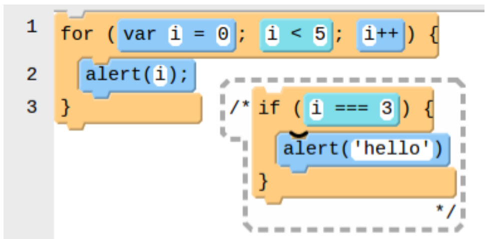

Fig. 4. An Example of Droplet's Floating Block Graphics

B. Related Work

Both Scratch and Blockly support floating blocks. Blockly runs all floating code in top-left to bottom-right order, while each Scratch block stack is associated with an event handler and runs whenever the attached event is fired. Scratch also runs a stack when it is double-clicked.

Droplet originally allowed floating blocks only provisionally, making them non-interactive and graying them out to indicate that they were not going to be run, and that the floating block area was just a place to store blocks while editing was happening in the main document.

C. Solution

Droplet now allows floating blocks to be assembled, but surrounds them with a dotted line and the language's block comment symbol to indicate that they are not going to be run (fig 4). The surrounding dotted line can be grabbed to move the entire block or replace it into the main document to "uncomment" it.

D. Future Work

In the future, Droplet may persist these blocks in the code by inserting them as comments and animating their text during the Droplet animation appropriately. Dropet may also add a "play" button to each floating block stack to allow stack to be run independently, the way that Scratch supports.

References

[1] Bau, D. A. Droplet, A Blocks-Based Editor for Text Code. Journal of Computer Science in Colleges. 30, 6 (June 2015).

[2] Mars Eclipse. http://eclipse.org

[3] Weintrop, D. and Wilensky, U. To Block or Not To Block, That is the Question: Students' Perceptions of Block-based Programming. IDC '15 proceedings (June 2015).

[4] Baecker, R. and Marcus, A. Design Principles for the Enhanced Presentation of Computer Program Source Text. CHI '86 proceedings (April 1986).

[5] Murphy, G. Kersten, M. and Findlater, L. How Are Java Software Developers Using the Eclipse IDE? IEEE Software (July/August 2006) 72-82.

[6] Nielsen, J. (1994). Heuristic evaluation. In Nielsen, J., and Mack, R.L. (Eds.), Usability Inspection Methods, John Wiley & Sons, New York, NY

[7] Scratch. https://scratch.mit.edu/

[8] Blockly. https://blockly-games.appspot.com/

[9] Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. 2010. The scratch programming language and environment. ACM Trans. Comput. Educ. 10, 4, Article 16 (November 2010), 15 pages. DOI = 10.1145/1868358.1868363. http://doi.acm.org/10.1145/1868358.1868363.