

# Learning Syntax as Notational Expertise when using DrawBridge

Alistair Stead

*Computer Laboratory  
Cambridge University  
Alistair.Stead@cl.cam.ac.uk*

Alan F. Blackwell

*Computer Laboratory  
Cambridge University  
Alan.Blackwell@cl.cam.ac.uk*

Keywords: POP-II.A. Novices, POP-III.D. Editors, POP-VI.E. Computer Science Education Research

## Abstract

We report an experiment that explores the classroom application of DrawBridge, a system designed to support the acquisition of skill in using text-based programming language syntax. Graphical syntax is increasingly popular in educational programming languages such as Scratch, but anecdotal reports from classroom teachers suggest that students are now finding it difficult to make the transition to conventional text syntax. This research investigates text syntax use as a specific notational competence, and evaluates the use of DrawBridge to scaffold text syntax acquisition.

## 1. Introduction

Learning to code, like learning mathematics, involves skill in manipulating symbols. In mathematics, symbols are rather standardised - the conventions of symbolic algebra and so on. As a result, many learners see mathematics as being equivalent to the use of these particular symbols, and find it hard to imagine a different mathematics in which the same concepts might be associated with alternative symbol systems. However, learning to code involves not only familiarity with a particular symbol system (a programming language), but also the realisation that the learner may soon have to learn a completely different one while retaining some abstract meta-understanding of the underlying concepts. The "computational thinking" agenda, like the "new maths" of the 1960s, places an emphasis on teaching these underlying concepts rather than any particular language syntax (Wing, 2006).

In the psychology of programming field, it is understood that choices made with regard to language syntax and environment can have significant implications for usability of the language (Green, Petre, & Bellamy, 1991) (Green & Petre, 1996)). When designing languages for use in schools, it is certainly important that they be usable (McKay & Kölling, 2013) (Pane & Myers, 1996). Recent generations of instructional language and environment, such as Alice, Scratch and Greenfoot, have been extremely successful in improving usability by learners. However, the novel syntax used in some of these languages does not expose learners to some of the fundamental syntactic features of conventional languages: parsing, identifiers, delimiters and so on. Furthermore, these very features are well known (to teachers of programming) to pose significant obstacles to learners, just as the correct use of mathematical notation can be a significant obstacle to those learning algebra.

In programming language design, a distinction is often made between keywords, identifiers, and the other elements of concrete syntax that are used to delimit and relate that vocabulary. Typically, concrete syntax such as operators and delimiters are implemented using punctuation symbols and whitespace. The popularly described "visual languages" often include keywords and identifiers, but replace much of the detailed concrete syntax with colouring, region boundaries, and other graphic devices. We describe this as "graphical syntax", by comparison to the textual concrete syntax that is constructed from punctuation marks and whitespace.

## 1.1. Notational Expertise

The question addressed in this research is whether there might be better ways of acquiring conventional coding skills - the skills associated with conventional code syntax. In contrast to "computational thinking" that focuses on abstract concepts independent of any symbol system (Wing, 2008), we might regard this objective as teaching *notational expertise*. We are not necessarily advocating notational expertise as an opposing view, or as a new and competing curriculum priority. We simply observe that it brings a different perspective to the teaching of coding, and one that might usefully be explored through experiments. Furthermore, the concept of notational expertise offers a novel research question for the psychology of programming, which has well-established understanding with regard to the usability of notation, but not necessarily with regard to the learnability of notation. In mainstream HCI, it has long been understood that learnability and usability are alternative and complementary requirements.

We are conducting this research through the design and evaluation of a specific educational system, called DrawBridge. DrawBridge aims to provide a motivational and scaffolded (Wood, Bruner, & Ross, 1976) (Guzdial, 1994) approach to learning programming language syntax. There are two scaffolding strategies. One of these allows learners to construct a program initially by manipulating a block-syntax representation (as used in Scratch), and subsequently by viewing and modifying the same program in conventional text syntax. The other scaffolding strategy allows users to create example code with programming by demonstration - making animations by dragging objects on a canvas. The motivational elements of DrawBridge support broader creative experiences: the animated objects are captured from drawings made on paper by the students, and the resulting animation can be viewed directly as a web page running in a browser.

## 1.2. Outline of Paper

The remainder of this paper is arranged as follows: the next section briefly describes the functionality of DrawBridge. We then describe two specific research questions related to the development of notational expertise, and two studies - an interview-based pilot study and a larger classroom experiment - investigating these questions. Finally, we discuss the findings of these studies, and an agenda for further investigation of notational expertise in the context of the DrawBridge system.

## 2. Overview of DrawBridge

DrawBridge adopts a familiar strategy that is shared by mathematics education and user interface design, of introducing abstract concepts via concrete manipulation. This general strategy is influenced by the Piagetian model of childhood development, which observes that children develop abstract mathematical reasoning skills only after they become practiced at concrete physical action. Alan Kay famously adapted Piaget's model of childhood stages in his slogan for the Smalltalk language and WIMP GUI: "doing with images makes symbols" (Kay, 1990). This strategy is applied in DrawBridge by providing students with a tool that allows them to provide their own concrete drawings (made using pen and paper), which are then captured and segmented for use as digitally animated characters. DrawBridge literally presents users with a sequence of stages, displayed on the screen in pairs, in which concrete drawing becomes represented as a more abstract geometric object that can then be manipulated using mathematical operations, as in Figure 1. DrawBridge initialises in the leftmost position.

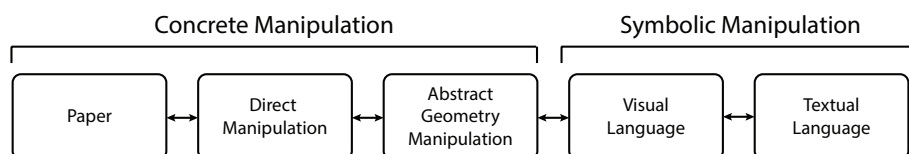


Figure 1 - DrawBridge Representations

Children find it highly motivating to draw their own characters. This has been noted as a limitation of Alice, where the complexity of creating 3D character models means that the system is restricted to providing (extensive) libraries of predefined characters. In Scratch, new sprites can easily be drawn

by children using a simple bitmap editor. DrawBridge simplifies the drawing process further, by supporting pen and paper with automated segmentation. However, as with Alice and Scratch, the introduction to programming occurs when the learner writes code to control the behaviour of the characters. Both Scratch and Alice use graphical syntax editors to support experimentation and minimise distracting syntax errors. However, as discussed already, this results in the trade-off that those languages do not explicitly teach the elements of conventional text language syntax that are central to much programming.

DrawBridge therefore adds a second scaffolding component, which allows learners to explore text syntax by use of an intermediary graphical syntax editor. As the user manipulates the hand-drawn characters to construct an animation program by demonstration, this program is initially displayed in a graphical block-syntax form that visually resembles Scratch or Alice (although with many small design differences, not discussed in detail in this paper). The user can experiment with behaviour modifications by manipulating the block syntax directly. However, users are also able to move on from block syntax to see and edit the same behaviour in conventional text syntax (here, valid JavaScript). The block syntax and JavaScript text syntax can be viewed side by side, with live, bidirectional interaction such that editing either of them also updates the other. The result is a scaffolded multi-representation editor that can be used to explicitly teach the principles of textual code syntax.

The remainder of this paper describes an initial study evaluating the design intentions of DrawBridge.

### **3. Research questions**

There are two primary research questions in this study:

1. To assess the educational benefit of moving from concrete drawing to symbolic mathematical operations
2. To assess the educational benefit of using graphical block syntax in order to scaffold learning of text syntax conventions.

### **4. Experimental design**

We explored these questions via a controlled experimental design, administered in a classroom setting. We manipulated two factors, via modifications to the DrawBridge system:

**Concrete drawing (C):** We created a version of the DrawBridge system in which there was no concrete drawing phase, so that participants started the task by manipulating geometric representation rather than their own drawings.

**Visual-first versus Text-first (VF/TF):** We created a version of the DrawBridge system in which participants used the conventional text representation before using the block-syntax representation.

The experimental measures tested participants' understanding of syntax use before and after using DrawBridge, administered as a classroom pre-test and post-test.

The study was conducted over a period of two weeks, involving two sessions with the same class of students. In the first session, the class was divided into four groups, with each group using a different version of DrawBridge as determined by a Latin square design based on the two experimental factors:

<b>C-VF</b>	<b>C-TF</b>
Concrete drawing phase	Concrete drawing phase
Visual block syntax first	Conventional text syntax first
<b>VF</b>	<b>TF</b>
No drawing phase	No drawing phase
Visual block syntax first	Conventional text syntax first

*Table 1 – Experimental design.*

All groups completed the same pre-test and post-test at the start and end of the session, which consisted of a multiple-choice component and a translation component. DrawBridge was also instrumented to record data concerning time spent on each representation pair, error information, button clicks and navigation.

In the second session, all students used the same version of DrawBridge, corresponding to the C-VF condition. The reason for this was that a large difference in educational outcomes had been observed during the first session, as discussed in more detail below. The second session was therefore intended to provide all students with access to the version of DrawBridge expected to be most educationally beneficial, in order that no student would have been disadvantaged through participation in the study.

At the end of the second session, a return test was administered, in the same format as the pre-test and post-test.

## 5. Participants

A complete class of twenty-one school students was recruited from an independent school in Cambridgeshire. Participants were studying in Year 7, aged 11-12, and had a range of academic aptitudes. The four conditions were allocated to participants according to blocks of seating in the classroom, so that participants were not distracted by neighbours carrying out different activities. In session 2, three members of the class were absent, and the session was conducted with 18 participants.

## 6. Procedure

In session 1, the procedure was:

1. Questionnaire regarding previous computing and programming experience
2. Pre-test assessing prior knowledge of JavaScript syntax (see appendix)
3. Task using DrawBridge, allocated to participants according to experimental condition
4. Post-test assessing knowledge of JavaScript syntax after using DrawBridge

In session 2, the procedure was:

5. All participants complete DrawBridge task in the C-VF condition
6. Return-test assessing knowledge of JavaScript syntax
7. Questionnaire assessing participants' experiences and attitude to DrawBridge (see appendix)

The experimental task in the Concrete + Visual-First (C-VF) condition corresponded the ordered sequence of stages in DrawBridge as follows:

- a) Draw a character on paper, which is photographed and loaded into DrawBridge by the experimenter
- b) Adjust the size and location of the character on screen using the image direct manipulation screen

- c) Demonstrate and record an animation using the abstract geometry screen
- d) Add an extra step to the animation using the visual block syntax screen
- e) Modify and extend the animation using the text syntax screen

In the Concrete + Text-First (C-TF) condition, the same sequence is followed, but with the representation formats in steps d) and e) reversed.

In the two conditions without the concrete drawing phase (VF and TF), participants did not carry out any activity equivalent to stages a), b) and c). Instead, they were provided with a predefined program, which they were asked to enter into DrawBridge, using either the visual block syntax screen or the text syntax screen according to condition VF or TF respectively. After viewing the resulting conversion to text (from VF) or to visual blocks (from TF), they were then invited to recreate the program by translating it directly from the predefined program sheet.

## 7. Results

Most participants had previous experience of programming in Scratch, which is included in their school curriculum. Nevertheless, only 10/21 reported that they had programmed a computer before. Despite the fact that they did not consider this to be “programming”, 15/21 reported that they had created animations before, and that Scratch was the tool they had used. 18/21 reported either that they loved using computers, or found it OK. 11/21 reported a "good" level of computer skill, with 5 reporting skill below this level, and 5 reporting skill above.

An initial analysis was conducted to compare the four experimental groups, according to results obtained in the pre-test. Unexpectedly, participants allocated to the TF group performed significantly better in the syntax knowledge pre-test, with mean score 6.6/10, compared to means of 2.2, 3.2 and 3.8 in the VF, C-TF and C-VF groups ( $p < .01$ ), see Figure 2. This data indicates a lack of balance in the sample groups rather than a treatment effect, given that it was collected before any treatment was administered. It is possible that this resulted from the decision to allocate participants to conditions according to the location of their seats in the classroom, and that more able students may have been grouped in one part of the class.

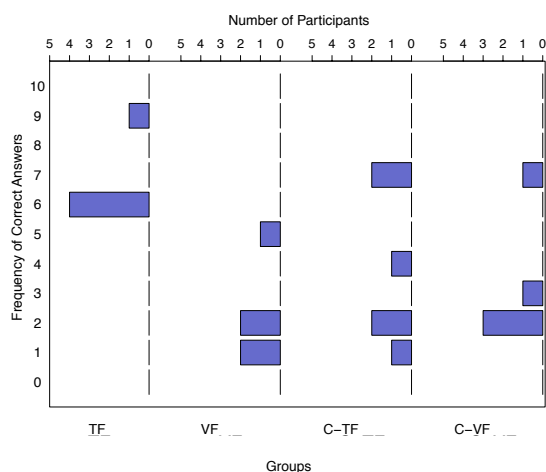


Figure 2 – Pre-Test Syntax Assessment

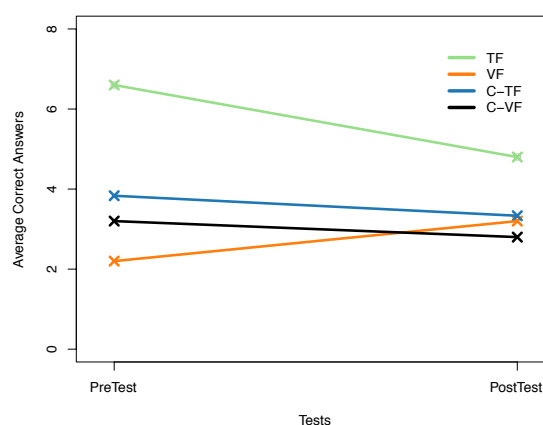


Figure 3 – Absolute Test Values

A comparison between pre-test and post-test results for the four experimental groups is shown in Table 1 and Figure 3. It is apparent that the high performance of the TF sample was not maintained in the post-test evaluation. On the contrary, performance on the post-test reduces substantially, although a pairwise t-test indicates the difference is not significant. Two possible explanations for this are either i) that the high pre-test scores were a coincidence, and that the lower post-test score represents reversion to the mean; or ii) that the TF condition impedes student's understanding of text syntax, such that otherwise capable students perform less well after carrying out this task. In order to explore

this further, we corrected for this pre-test sample difference by calculating change in test score between the pre-test and post-test, rather than absolute number of correct answers.

Group	Pre	Post	Return	Post-Pre	Return-Post	Return-Pre
TF	6.6 ± 1.67	4.8 ± 2.54	3.8 ± 1.62	-1.8 ± 2.96	-1.0 ± 2.22	-2.8 ± 5.01
VF	2.2 ± 2.04	3.2 ± 2.69	4.2 ± 3.09	1.0 ± 3.04	1.0 ± 3.51	2.0 ± 6.51
CTF	3.83 ± 2.77	3.33 ± 3.36	4.2 ± 2.83	-0.5 ± 3.74	0.87 ± 2.91	0.37 ± 5.00
CVF	3.2 ± 2.69	2.8 ± 2.04	3.67 ± 7.17	-0.4 ± 1.42	0.87 ± 5.74	0.47 ± 3.35

Table 1 – Assessment Marks and Change Scores (Green – above mean, Red – Below mean, Confidence Interval of 0.95)

After correcting for pre-test sample differences, we see that the visual-first treatment is associated with improved performance in the post-test, and that the concrete treatment is associated with slightly better performance than the conditions without the concrete task (see Figure 4). We also see that groups using concrete representations performed better on average in the post-test than groups not using concrete representations (4.0 vs. 3.1 mean respectively). However, the difference was not significant ( $p = 0.38$ ) and there was little difference in the change score between the pre-test and post-test.

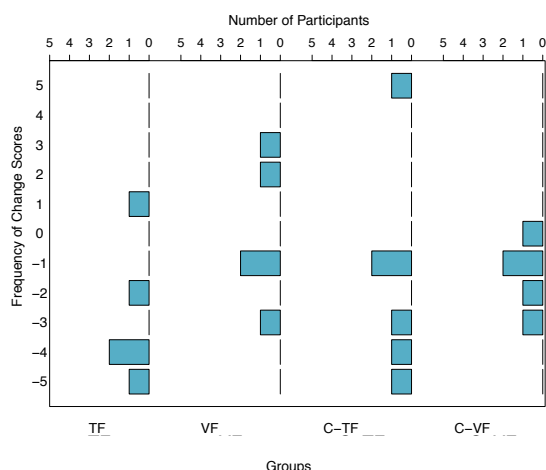


Figure 4 – Pre-Test to Post-Test Change Scores

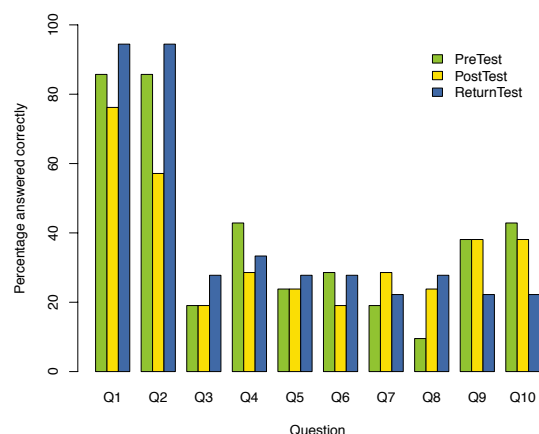


Figure 5 – Correct Answers by Question

In response to the overall reduction in performance that was observed between the pre-test and post-test, a further lesson was arranged, in which all participants carried out the C-VF task. The result of the return-test administered after this task is shown in Table 1 and Figure 6. On average, performance improved in the syntax component of the return-test, with the exception of the group originally assigned to the TF condition, which decreased even further. This suggests that, of the two explanations for the original fall in performance, reversion to the mean seems the most likely.

The percentage of correct answers for each question in the multiple-choice component is shown in Figure 5. In order to improve participant self-efficacy, we included two “easy” questions relating to the validity of a web address and email address. Participants’ answers for these questions were significantly better than other questions ( $p < .05$ ) as expected. However, despite this and other attempts to reduce anxiety and increase participation, test results suffer from poor reliability - many participants provided no answer to a substantial number of the translation questions, or gave the same answer to every multiple-choice question.

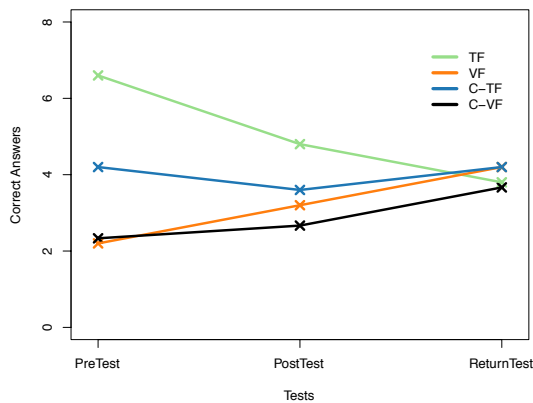


Figure 6 – Absolute Test Values

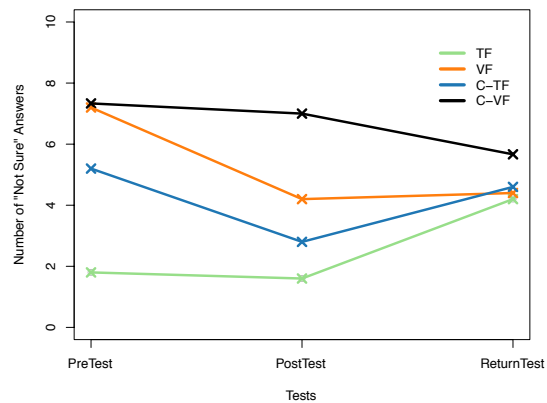


Figure 7 – “Not Sure” responses

The number of “not sure” responses in the multiple-choice component decreased for VF and C-TF between the pre-test and post-test, suggesting that self-efficacy improved for these groups (see Figure 7). Responses stayed reasonably constant for TF and C-VF in the post-test, before all groups converged towards a mean value of 4.5 in the return session.

A comparison of the time spent viewing symbolic representations is shown in Figure 8. Participants in without-concrete groups spent significantly longer viewing symbolic representations than with-concrete groups ( $p < .001$ , 3m 23s on average vs. 38m 35s). With-concrete groups spent most time (19m 51s on average) viewing the first pair of representations, which included direct manipulation of automatically segmented characters.

All but two participants present on the second visit reported that they loved, liked or did not mind using DrawBridge, with all but three happy to use DrawBridge again. When participants were asked to report the three best things about DrawBridge, they agreed that drawing their own characters and being able to animate was a clear strength - “see your own animations. it’s fun! it’s different!” and “You can do some cool animation, it’s fun to do, being able to control the characters.” When asked about the three worst things, reliability was noted as the main problem - “sometimes it froze”, and “It’s a bit slow sometimes”.

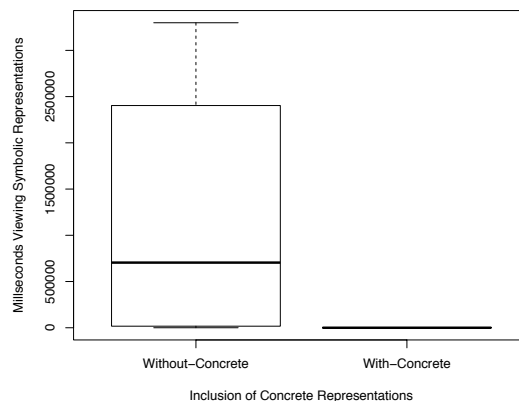


Figure 8 – Instrumentation Viewing Time

## 8. Discussion

Our aim in this study was i) to assess the educational benefit of moving from concrete drawing to symbolic mathematical operations and ii) to assess the educational benefits of using graphical block syntax in order to scaffold learning of text syntax conventions.

Our pre-test results suffered from imbalance, apparently caused by an association between participants' seating arrangements and ability. We therefore focused on change scores between tests, in accordance with best practice for non-equivalent group design analysis (May, 2012), and previous research (Levy, Ben-Ari, & Uronen, 2003) However, change-score analysis can be prone to regression artefacts (Reichardt, 2005), which occur when groups with extreme pre-test values regress towards the mean in the post-test. Given the extreme pre-test results of group TF, it is likely that the decline observed in post-test results is due to regression to the mean rather than exposure to DrawBridge.

The change scores relating to the benefits of the inclusion of concrete drawing representations suggest that there was no significant educational benefit to including concrete drawing representations. However, instrumentation results show that participants in without-concrete groups spent 11.43 times longer viewing symbolic representations than with-concrete groups, which could explain the lack of difference. Furthermore, results from the questionnaire given on the second visit showed that drawing on paper and manipulating characters for animation provided a major motivation for participants.

Results relating to the order of visual and text representations show that groups who encountered the visual representation first showed the highest change scores between the pre-test and post-test (see Figure 4), suggesting that exposure to and scaffolding from the visual representation before the text representation was beneficial to users. Return-test results show that visual-first groups showed the highest improvements, suggesting that longer exposure to visual-first groups also proved to be beneficial.

The difference between the number of answers in which participants responded "Not Sure" between the pre-test and return-test was significantly ( $p < .05$ ) affected by order: visual-first groups VF and C-VF had a reduced number of "Not Sure" answers (-2.375 on average), while text-first groups had an increased number (0.9 on average), suggesting that the visual-first order may increase self-efficacy. However, it is possible the difference could be due to faster maturity of participants in the visual-first groups, who had longer exposure to the visual-first version of DrawBridge used in the return test.

## 9. Conclusion

We presented an experiment that explored the classroom application of DrawBridge, a novice-programming environment that transitions users from concrete representations to abstract, symbolic representations in an effort to scaffold syntax acquisition. Results show that users who were able to use the visual representation first improved more than those encountering the text representation first. We also found that the use of a pen and paper as a starting point for programming provided major motivation to participants. Unfortunately, the experiment suffered from an imbalance of pre-test results and low response rates in some questions, resulting in weakened internal validity. However, steps were taken to address the initial imbalance by using a return test in addition to the pre-post comparison. Future work will explore students' experience of DrawBridge through more in-depth qualitative interviews, leading to refinement of the DrawBridge prototype and the evaluation tests used. Based on the outcome of that study, we expect to carry out further classroom evaluation.

## 10. References

Green, T. R. G., & Petre, M. (1996). Usability Analysis of Visual Programming Environments: A "Cognitive Dimensions" Framework. *Journal of Visual Languages and Computing* (pp. 1–51).



- Green, T. R. G., Petre, M., & Bellamy, R. K. E. (1991). Comprehensibility of Visual and Textual Programs: A Test of Superlativism Against the “Match-Mismatch” Conjecture. *ESP*, *91*(743), 121–146.
- Guzdial, M. (1994). Software-Realized Scaffolding to Facilitate Programming for Science Learning. *Interactive Learning Environments*, *4*(1), 37–41.
- Kay, A. (1990). User Interface: A Personal View. *The Art of Human-Computer Interface Design*.
- Levy, R. B.-B., Ben-Ari, M., & Uronen, P. (2003). The Jeliot 2000 Program Animation System. *Computers & Education*, *40*(1), 1–15.
- May, H. (2012). Nonequivalent Comparison Group Designs. *APA Handbook of Research Methods in Psychology, Vol 2: Research designs: Quantitative, Qualitative, Neuropsychological, and Biological* (Vol. 2, pp. 489–509). Washington, DC, US: American Psychological Association.
- McKay, F., & Kölling, M. (2013). Predictive Modelling for HCI Problems in Novice Program Editors. *27th International British Computer Society Human Computer Interaction Conference*. London, UK.
- Pane, J. F., & Myers, B. A. (1996). Usability Issues in the Design of Novice Programming Systems, (August).
- Reichardt, C. (2005). Nonequivalent group design. *Encyclopedia of Statistics in Behavioral Science*.
- Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, *49*(3), 33.
- Wing, J. M. (2008). Computational Thinking and Thinking about Computing. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, *366*(1881), 3717–3725.
- Wood, D., Bruner, J. S., & Ross, G. (1976). The Role of Tutoring is Problem Solving. *Journal of Child Psychology and Psychiatry*, *17*(2), 89–100.

## 11. Appendix

### 11.1 Pre-Test



Name: \_\_\_\_\_

School: \_\_\_\_\_

Date: \_\_\_\_\_

#### 1. Do these pieces of text look correct?

www.google.com

Yes

No

Not Sure




user@gmail.com

Yes

No

Not Sure




variable x = 2;

Yes

No

Not Sure




var x == 2;

Yes

No

Not Sure




varx = 2;

Yes

No

Not Sure




var x = 2

Yes

No

Not Sure




var x <= 2;

Yes

No

Not Sure




var x = 2;

Yes

No

Not Sure

## Pre-Test Page 2

```
setImagePosition 100;
```

Yes

No

Not Sure



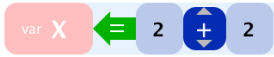

```
setImagePosition(100);
```

Yes

No

Not Sure

## 2. Can you fill in the missing pieces?

Blocks	Code
	
	<pre>setTimeToDestination(500);</pre>
	
	<pre>loadImage(2, 541, 173, 238, 168);</pre>
	
	<pre>document.getElementById("myCanvas");</pre>
	
	<pre>var X = 2 / 2;</pre>

## 11.2 Return Questionnaire



Name: \_\_\_\_\_

School: \_\_\_\_\_

Date: \_\_\_\_\_

**1. Overall, how much have you enjoyed using DrawBridge?**

Hated it



Disliked it



Didn't mind it



Liked it



Loved it

**2. Would you like to use DrawBridge again?**

Never



Not really



Don't mind



OK



Definitely

**3. What are the three best things about DrawBridge?**

**4. What are the three worst things about DrawBridge?**

**5. How can we improve DrawBridge?**