Dual Modality Instruction & Programming Environments: Student Usage & Perceptions

Jeremiah Blanchard
Dept. of Engineering Education
University of Florida
Gainesville, FL, USA
jjb@eng.ufl.edu

Christina Gardner-McCune
Department of CISE
University of Florida
Gainesville, FL, USA
gmccune@ufl.edu

Lisa Anthony
Department of CISE
University of Florida
Gainesville, FL, USA
lanthony@cise.ufl.edu

ABSTRACT

Dual-modality blocks-text programming environments have shown promise in helping students learn programming and computational thinking. These environments link blocks-based visualizations to text-based representations, which are more typical of production languages. Since prior work shows that some students who learn in dual-modality environments outperform those who learn in text on assessments, we sought to understand specifically how students use dual-modality environments and what support these environments provide to the learning process. We analyzed survey responses and tool logs collected during a study at a large public university in a CS1 course (N=425). We found that students from all prior programming experience backgrounds made use of the ability to visualize code structures by using blocks. Students with prior experience in blocks or no prior experience said they felt the dual-modality instruction helped them understand code structure and meaning. As students progressed through the class, we found that they made more use of the blocks mode's reference palettes than to its drag-and-drop facilities or mode-switching features. By identifying how students interact with dual-modality tools and how they impact student understanding, this work provides guidance for classroom instructors.

CCS CONCEPTS

•Social and professional topics~Professional topics~Computing education~Computing education programs~Computer science education~CS1•*Human-centered computing~Visualization*

KEYWORDS

Computer science education; CS1; blocks-based programming environments; programming languages; novice programmers; dual-modality programming environments.

SIGCSE '21, March 13–20, 2021, Virtual Event, USA.
 ACM ISBN 978-1-4503-8062-1/21/03…$15.00
https://doi.org/10.1145/3408877.3432434

ACM Reference format:

Jeremiah Blanchard, Christina Gardner-McCune, and Lisa Anthony. 2021. Dual Modality Instruction & Programming Environments: Student Usage & Perceptions. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE'21).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432434

1 Introduction

Students learning to program in text-based languages often face learning barriers related to syntax, which can cause them to perceive text-based programming negatively. Focus groups with women and girls suggest they sometimes find text programming intimidating [3], and this may impact future motivation to study computer science [59]. Studies also show that programming as a discipline suffers from association with "uninteresting" tasks [23], which has been implicated in limiting motivation especially among students in minority populations [8]. Blocks-based environments were developed to address many of these issues by making the environments more inviting and approachable while incorporating affordances and scaffolding to help students learn to program [18, 11, 17]. These environments have been shown to alleviate syntax challenges when initially learning programming concepts [20], but some students who move from blocks to text continue to struggle with negative perceptions after the move to text [13] and find the experience frustrating [4].

Building on work in blocks-based environments, **dual-modality programming environments** link blocks-based and text-based representations of code. Blocks allow students to learn programming concepts separate from syntax, while text offers authenticity more typical of production languages in advanced coursework, research, and industry. Prior work has shown that some students who use dual-modality environments to transition from blocks to text have more positive views of text programming compared to students who move directly from blocks to text languages [4]. Additionally, work has shown that some students working in dual-modality environments in an introductory computer science (CS1) course outperform those who worked solely in text on assessments [5]. By helping to alleviate negative perceptions, dual-modality programming instruction may contribute to improved motivation and confidence, which have both been shown to improve retention within the discipline [15].

In this work, we present an analysis of how students use dual-modality tools and materials, from data collected as part of a larger study [5]. We sought to answer the questions, **"How do CS1 students perceive dual-modality instruction?"** and **"How do CS1 students use dual-modality programming tools?"** The larger study was conducted at a public university in a high-enrollment CS1 course (N=425). In this course, students were provided with the open source Amphibian plugin for the IntelliJ integrated development environment (IDE) [6, 24]. Amphibian is a dual-modality plugin that allows students to move back and forth between blocks and text when developing Java programs. For this paper, we analyzed self-report survey responses about experiences using the environment, along with plugin usage and course material access logs. We investigated how students described their experience learning in a dual-modality environment, how they used tools provided to them, and how their usage and experiences changed over time. We hypothesized that the self-paced nature of the programming environment would provide students a level of control that is empowering and that would be reflected in student responses.

2 Background

Ongoing work in developing tools and environments to help students learn programming requires consideration of not only providing scaffolding to help students learn, but also supporting their motivation. Here we review prior work on use of blocks, text, and dual-modality approaches to computing instruction.

2.1 Perceptions of Blocks and Text

Text-based languages, especially production languages, have the benefit of being perceived as more authentic than blocks-based languages by many learners [21]. However, interviews and surveys show that some students perceive text languages as hard and/or intimidating [11]. Thus, many early computing curricula, especially for younger learners, make use of blocks-based environments. These environments have been used successfully to help students learn. Studies have found that scaffolding these environments provide eliminates problems with syntax errors, allowing students to develop computational thinking, while also nurturing an appreciation of trial-and-error approaches necessary for programming and debugging [14,20,16].

However, even after working in and successfully learning in blocks, some students continue to struggle with negative perceptions when they move to text. In Weintrop's study with high school students [21], participants starting in blocks reported statistically significant increased levels of confidence from the start to just before changing to text (the midpoint of the study), but after switching from blocks to text they reported a statistically significant decrease in confidence in the second phase. The students who worked exclusively in text, however, did not show statistically significant changes in confidence over the course of the study. We also studied students in middle school who learned in blocks, text, and dual-modality environments [4]; we found that students who started in blocks perceive text more negatively after switching from blocks to text, unlike their counterparts who worked exclusively in text.

2.2 Dual-Modality Environments & Instruction

Dual-modality programming environments provide a means to translate between representations – often from blocks into text or vice versa – to help students understand the relationships between visual representations of constructs and their text-based counterparts [21]. Environments like Pencil Code and tools such as Amphibian (**Figure 1**) link blocks-based visualizations, which allow students to learn programming concepts separately from syntax, and text representations [2, 6]. By providing a practical and usable environment that can quickly transition between blocks and text representations of the same program semantics, dual-modality programming environments combine the affordances of blocks-based environments with the authenticity of text languages, while explicitly showing the equivalence of blocks and text variants of the same constructs.

Prior work shows that dual-modality instruction and environments can support positive perceptions of programming, as well as learning, in CS courses. In work with middle school students, we found that students who worked in dual-modality environments before moving to text held more positive views of text programming compared to students who move directly from blocks to text [4, 5]. In our work with CS1 students, assessments showed that students who learned via dual-modality instruction outperformed those who worked solely in text [5].

Prior research has further indicated that students in dual-modality programming environments may use blocks when learning new constructs, but they transition to text over time [2]. Bau et al. [2] found that, in a small group of middle school students (N=8), use of the text-based editor increased over time with experience, suggesting that, as students became accustomed to programming, they began using text mode more often. Students from a variety of prior experience backgrounds (i.e., no prior coding experiences, blocks-based experiences, and text-based experiences) can benefit from the ability to self-scaffold, transitioning when they are ready. Later, Weintrop & Holbert [22] showed that students most often switched from text back to blocks when adding new or unfamiliar constructs.

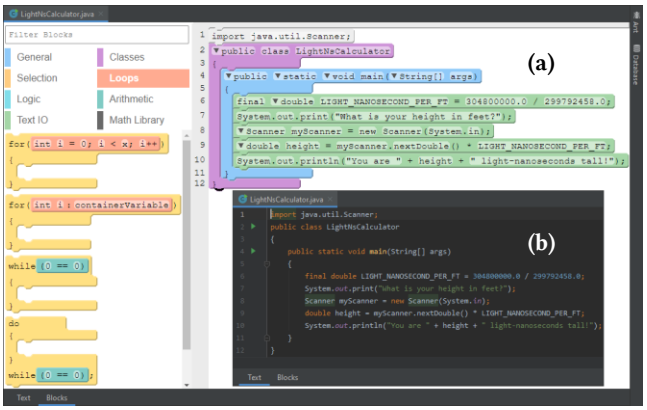

Fig. 1: Amphibian Plugin; blocks (a) with text overlaid (b).

3 Methods

For this paper, we analyzed surveys students filled out asking about their experiences using the environment for each course module during a CS1 course [5], along with Amphibian plugin usage logs and learning management system (LMS) download logs. All students met in a large lecture hall together, three times weekly, in addition to a smaller two-hour lab.

3.1 Study Design

The analysis reported in this paper was collected in a larger study, conducted during a CS1 course taught using dual-modality instruction and the Amphibian IDE plugin. Key details of the study design are reproduced here for clarity; for full details, see Blanchard et al. [5]. One goal of the instruction was to move students to comfort in text by the end of the course. As such, the first two-thirds of the topics (8 out of 12) were covered in dual-modality instruction (**Table 1**). The remaining topics included advanced programming topics covered only in text (e.g., recursion and inheritance). The course included weekly surveys during each module about student perceptions of dual-modality instruction. There were two midterms and a final exam.

3.2 Dual-Modality Instruction

Course materials – including lecture slides and assignment descriptions – were adjusted by the instructor (the first author) to illustrate the plugin's blocks and text representations as well as the transition between blocks and text. In lecture slides, rather than individual lines of text, code was presented in blocks, with animated transitions to text to connect representations for students (**Figure 2a**). Assignment manuals also included text and blocks representations in the Amphibian visual style wherever sample or demonstration code was provided (**Figure 2b**).

3.3 Participants

In the study, there were more men than women, but students came from diverse ethno-racial groups (**Table 2**). Students also came from varied programming backgrounds: 40.2% of students (N=101) indicated that they had some prior programming experience. However, some students gave conflicting answers about this; for example, some students who said they *did not have* prior experience also marked experience in specific languages[1] – frequently blocks-based and scripting languages (e.g., Scratch or Python). From student responses to

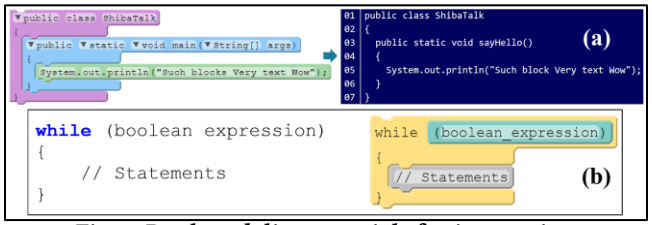

Fig. 2: Dual modality materials for instruction: (a) lecture transition example and (b) assignment sample

[1] Students may have been separating formal and informal programming experiences.

Table 1. Topics by instructional mode

Module	Course Topic	Instruction
0	*Introduction to CS*	Blocks & Text
1	*Variables / Arithmetic*	Blocks & Text
2	*Control (Selection, Loops)*	Blocks & Text
3	*Data Types / Arrays*	Blocks & Text
4	*Functions / Static Methods*	Blocks & Text
5	*Software Engineering*	Blocks & Text
6	*Classes & Objects*	Blocks & Text
7	*Algorithm Complexity*	Blocks & Text
8	*Inheritance / Interfaces*	Text Only
9	*Data Streams & Recursion*	Text Only
10	*Propositional Logic*	Text Only
11	*Memory & Paradigms*	Text Only

programming language and environment experience questions, we identified an additional 50 students with prior programming experience, for a total of 60.2% (N=151). 39.8% (N=100) of students did not report prior experience in any question responses. Of the students with prior programming experience, 68.9% (N=104) indicated that they had taken AP Computer Science, AP Computer Science Principles, or a college-level programming course, while 31.1% (N=47) indicated that they had not taken any of these.

3.4 Data Collection

We collected several types of data in this study, including assessments, survey responses, and use logs for the plugin and course resources. We previously reported assessment data from this study [5]. We base our analysis in this paper on the surveys (described below), plugin logs, and resource access logs.

3.4.2 Perception Surveys. In weekly labs, students completed a five-question survey about their experiences using the environment. The survey asked questions about student use of blocks and text ("Did you program in 'Blocks' mode since the end of your previous lab?", Yes/No), and student perceptions of the helpfulness of dual-modality instruction ("Does instruction in dual blocks-text modes help you learn better?", Likert) along with free response prompt ("Why do you feel this way?").

3.4.3 Plugin and Resource Logs. The Amphibian plugin collected logs via a secure server, and the course learning management system (LMS) also logged accesses to course lecture materials. The plugin logs included events involved in switching modes (between blocks and text), palette viewing (showing code snippets in blocks), and blocks use (dragging and dropping constructs to build programs), along with time spent in each

Table 2. Student demographic groups

Group	Students	Group	Students
Women	25.1%, n=63	*Black / Afr.Amer.*	5.2%, n=13
Men	74.9%, n=188	*Hispanic/Latino*	24.7%, n=62
Asian	30.7%, n=77	*Native American*	0.4%, n=1
White	52.6%, n=132	*Nat.HI / Pac.Is.*	0.4%, n=1
Other	1.6%, n=4	*Multiple*	15.5%, n=39

mode. Course materials logs identified the time and file accessed, and each lecture slide deck was tied to a course topic / module.

3.5 Data Analysis

In our analysis, we attempted to identify how dual modality instruction shapes and affects the student experience, as well as the role prior experience plays – including any differences for students who have prior experience in blocks versus text-only programming. We examined student use of the dual-modality IDE plugin and materials to identify how students made use of them and to identify any patterns in their use. We classified students' prior experience based on answers to the demographic survey, which included questions about the specific languages and environments that students had used previously. We labeled each mentioned programming language or environment as "blocks" or "text" to identify the experience background for each student. Since most students with experience in blocks also had some experience in text, we classified student experience into three categories: "Text-only", "Blocks", and "None".

3.5.1 Qualitative Coding. We coded a subset of the consenting students' free responses using an inductive, multi-step qualitative coding approach based on the process described by Auerbach and Silverstein to develop a list of repeating ideas and refine them via iteration [1]. From the 252 students who completed the demographic survey, we selected a sample of 63 (25.0%) students. We selected these students to maximize coverage of ethno-racial, age, educational level, gender, and experience groups. We first included students who were the only representatives of a group; for example, there was only one Pacific Islander student. We then added individuals with a unique combination of demographic factors (age, education, gender, ethno-racial background, and experience). For remaining combinations, we randomly selected individuals from students who responded to all surveys. We coded responses from modules 1, 3, 4, 7, and 11 (Table 1). Modules 1 and 11 were the first and last surveys, respectively; 3 and 4 covered programming fundamentals (loops, data types, and functions); and module 7 covered the period in which the instructor changed from dual-mode to text-only instruction.

We completed coding in a four-step process to identify student reactions to dual-modality instruction and how they used materials. We focused our analysis on free response answers to questions about the helpfulness of dual-modality instruction. To start, the first author inductively developed a list of repeating reasons students gave for why dual-modality instruction was / was not helpful (e.g., ***Visualization*** or ***Practicality***) and iteratively refined them to establish 33 codes. Next, we independently coded responses from three students and discussed disagreements to refine the codebook. In discussion, we adjusted some codes and combined codes we determined had the same meaning (e.g., ***Familiarity*** and ***Accustomed***), leaving 29 codes. Two researchers coded responses from an additional 12.7% (n=8) of students, which we used to perform an inter-rater reliability analysis using Fleiss kappa [9]. Finally, the first author coded the remaining samples (n=55). The average agreement between coders was Fleiss kappa = 0.752, which is characterized as substantial agreement [19]. To identify patterns in the coded data, we did a frequency analysis. For each code, we counted the number of students whose responses in a given survey fit the code, and divided by the total number of students, to report the percentage of students whose responses fit the code.

3.5.2 Usage Patterns Over Time. To examine usage of lecture slides on Canvas, we identified a two-week timeframe for each set of slides that covered the introduction, conclusion, and first quiz or exam covering the topic. For each student, if access to the slide deck occurred within this coverage window, we marked the slides for that module as used by that student for the purposes of this analysis. For plugin usage, we identified a window covering the beginning to the end of the module, and for each student, if the plugin was used in the time window (e.g., using blocks or switching modes), the plugin was marked as being used by that student. Interactive plugin events were grouped into the categories of "Block Use", "Palette Viewing", and "Mode Switching" for the analysis. Palette Viewing actions were those in which the student selected a category of blocks to view (such as "Control", "Classes", or "Variables"); Mode Swapping was logged whenever a student switched from blocks to text mode or vice versa; and Block Use actions are those in which a student selected (dragged) a programming block and/or placed (dropped) a block within the program window. For each module, we calculated the percentage of students who used the lecture slides and plugin. We also calculated the average percentage of students who used the lecture slides and plugin over all module time windows. Finally, we plotted responses to survey questions about perceptions of dual-modality instruction and plugin logs by module in a time series so that they could be compared time-wise for triangulation [7].

4 Findings

In this section we discuss how students used the materials and their perceptions of the dual-modality instruction. We illustrate student perceptions through their words. We randomly generated pseudonyms for each student made up of adjective-animal pairs produced by the PetName module in Python [12].

4.1 Use of Dual-Modality Materials

On average across all weeks, 58.9% of the students accessed the lecture slides for the modules while they were being covered in class (Median=58.3%; σ=27.3%). The highest percentage of accesses was in the week before Exam 1, when 70.1% of students accessed the slides (Module 4), and the weeks before and following Exam 2 (Module 5, 70.7%; Module 6, 71.8%).

88.0% of the students in the class (n=374) installed and registered the plugin and used it in at least one session, with a total of 98,913 unique logged events. Logs of plugin events showed that most of the students used the plugin during each module (**Figure 3**). Viewing multiple palettes sequentially within a 60-second window was considered a single event in our

analysis. Aggregated across all weeks and students, the majority of students' interactions with the plugin were Palette Viewing (59.8%, n=58,656) followed by Mode Swapping (36.2%, n=35,503), with Block Use being the smallest category (4.4%, n=4,279). During the first module, Palette Viewing (36.3%, n=1,799) and Mode Swapping (52.0%, n=2,577) events were most common, with fewer Block Use events (11.7%, n=582). Over time, the percentage of Mode Swapping and Block Use events decreased while Palette Viewing events increased. By the last week, most events were Palette Viewing (68.9%, n=9,335), with a smaller percentage of Mode Swapping (30.9%, n=4,182) and few Block Use events (0.3%, n=46).

The surveys provide further insight into how students used the dual-modality materials (lecture slides and plugin) during the course. For example, one student with prior text experience who used the plugin in 10 of the 11 modules mentioned that palettes were an effective reference for constructs: "it's helpful to refer to the blocks for quick reference" (unique-unicorn, Code: Helpful, Module 4 [M4]). Students also said swapping modes helped them visualize their code and understand its structure. A student who used the plugin during every module wrote, "It allows me to see two ways of coding the same program, and sometimes blocks are more structured" (sharp-stingray: Helpful, M7).

4.2 Perceptions of Helpfulness

Students from all experience backgrounds indicated in survey responses that dual-modality instruction was helpful throughout the course. More than half of students (54.6%, n=137) perceived the dual-modality instruction as helpful in learning to program at the beginning of the course. Over time, the percentage of students that perceived dual-modality as helpful decreased. However, even at the end of the course, after students had learned some advanced programming topics, 34.8% of students with prior experience exclusively in text said dual-modality instruction was helpful, along with 37.1% of blocks-experience students and 43.8% of students with no prior experience.

When we analyzed free response answers, students' most commonly cited reasons were: (a) ***Visualization***: related to visualization of the code (cited by 41.3%, n=26), (b) ***Structure***: structuring code or understanding structure (22.2%, n=14), (c) ***Understanding***: understanding code / concepts (20.6%, n=13), and (d) ***Introduction***: introducing constructs to beginners (19.0%, n=12). Visualization was consistently reported as a reason by students from all programming experience backgrounds.

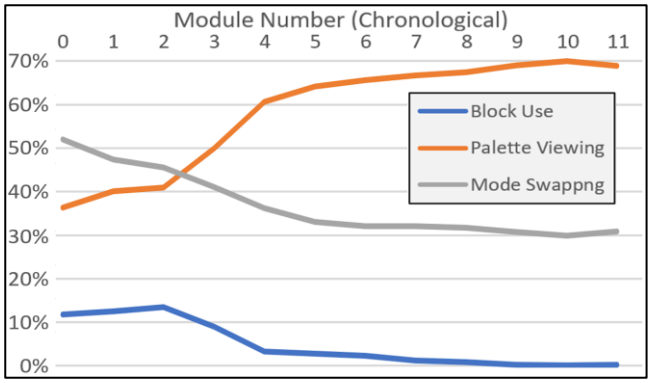

Fig. 3: Amphibian Plugin Events, Pct. vs Time

4.2.1 Prior Experience only in Text. Among students whose only prior experience was in text, 30.0% to 45.1% said that dual-modality instruction was helpful to them throughout the semester. Among the coded student samples, 35.3% (n=6) of text-experience students cited help in ***Visualization*** on at least one survey. One student said, "it helps me to learn the simpler way (blocks) before having to put concepts into practice (text)" (neutral-narwhal: Helpful, M3).

Several students with text experience empathized with new learners, pointing out how dual-modality instruction could help students as an ***Introduction*** to programming (23.5%, n=4). For example, one student reflected on the experiences of new programmers, stating, "Blocks is good for new programmers" (fond-falcon: Helpful, M7). Another pointed out that the dual-modality instruction was helpful in lecture when introducing new concepts: "Dual block text in class is helpful for highlighting general structured [sic] when they [constructs] are introduced" (current-chipmunk: Helpful, M3).

Not all students with prior experience (17.6%, n=3) perceived dual-modality instruction as helpful. Some pointed to uses of authenticity and dependency, referring to blocks scaffolding as a crutch: "blocks creates a programmer who is reliant on pre-set syntax" (big-buzzard: Not Helpful, M3). Another 11.8% (n=2) said they found blocks distracting, with one student noting, "colors in block mode distracts me" (fond-firefly: Not Helpful, M1).

4.2.2 Prior Experience in Blocks. 36.2% to 56.9% of students with prior blocks programming experience indicated that dual-modality instruction helped them learn. When asked why, students with blocks experience indicated ***Visualization*** most often (47.4%, n=9), with one student stating, "It helps to visualize more the regions of the code with the uses of colors shaped areas around the areas of the code" (ideal-ibex: Helpful, M1). Students with blocks experience also mentioned usefulness as an ***Introduction*** to programming (31.6%, n=6), with one saying, "It helps me when learning the basics of text, as I can look to blocks for help" (patient-panda, Helpful, M1).

In later surveys, some students with prior experience in blocks indicated that they felt they had progressed in their skills over time and used the blocks constructs less often as a result; their early survey responses (e.g., M1) described their use of blocks constructs, and responses to later surveys (e.g., M4, M7) indicated that they no longer needed or used blocks. On the first survey, for example, one student said, "When learning a new programming language, the syntax and structure of the language can be confusing. Block code is a little easier to read" (equipped-emu: Helpful, M1). In the third survey, the same student noted: "I primarily code in text mode, but sometimes it helps to jump into block mode to see the color code looping" (equipped-emu: Helpful, M3). By the fourth survey, the same student noted that they no longer depended on blocks: "I didn't really use blocks this time around. I am just more comfortable using text" (equipped-emu: Helpful, M4).

4.2.3 No Prior Experience. Most students with no prior programming experience indicated they found dual-modality instruction helpful during every module that dual-modality instruction was used in the class, from the first survey through the seventh, ranging from 50.5% to 61.7%. At the end of the course (M11), nearly half (43.8%, n=39) still found dual-modality instruction helpful. Responses from students with no prior experience most frequently cited ***Visualization*** (40.7%, n=11) and ***Structure*** (22.2%, n=6). Some of them also showed growing independence from the scaffolding over time. On the first survey, one student said, "[it] allows for better visualization of Java language with blocks, but also allows for necessary learning of Java language through text" (sure-shrimp: Helpful, M1); by the final survey, the same student responded differently: "now I feel as though I do not necessarily need it to understand the text code" (sure-shrimp: Helpful, M1).

5 Discussion

While previous literature studying use of dual-modality tools noted that students swap between modes less often as they progress in their programming skills [9], our findings extend the community's knowledge to include the types of actions students engage in and how those actions change over time. We were surprised that students in our study primarily used blocks mode as a code reference, rather than to construct programs. Further, while students differed in their perceptions based on their prior experience, many students from all experience backgrounds found the dual modality instruction helpful throughout the class.

Before our analysis, we expected that students would use the plugin for a variety of tasks, including using blocks mode to write code by selecting, dragging, and dropping blocks. However, the most common event in the logs was Palette Viewing, in which students view sections of blocks-based code snippets. Student free responses suggested they refer to palettes as a quick-lookup mechanism for sections of code. By comparison, students only infrequently used blocks to program, suggesting the scaffolding of blocks mode to write and construct programs was not a highly utilized feature. Instead, students said they used the tool to remember how to use certain constructs they had learned in text.

We also found that students with different experience levels reacted differently to dual-modality presentation materials (e.g., lecture slides) than to the dual-modality plugin that allowed programming in different modes. Students with no prior experience said the plugin helped them understand code structure. Thus, employing the dual-modality plugin in earlier curricula – such as the AP CS Principles course – may help students learn even before reaching CS1 courses by helping them to link blocks and text representations. Based on responses from students with prior experience, the benefits of dual-modality presentation materials may extend beyond early programming to later computer science courses (e.g., CS2 and Data Structures courses). Further, the benefit to experienced students from dual-modality representations suggests that there may be value even to experts in such a visual blocking mechanism. Students in the course used the blocks mode of the plugin to check their understanding of code structure and ensure its correctness; this could be explored within professional IDEs, for example, by graphically delineating boundaries of constructs in text modes.

6 Limitations & Future Work

There are important limitations of our work that could be a source for exploration in future work. In the larger study, it is possible that a selection bias played a role in student differences, as the data were collected in different semesters, and we did not use a random-control model. Additionally, the study was conducted at a single university with a single instructor. In the larger study, dual-modality instruction ended at basic object-oriented structures, and notably did not include such concepts as inheritance, interfaces, and abstract classes. Integration of representations of such concepts into dual-modality tools – such as incorporating the visual inheritance modeling in BlueJ and GreenFoot [12, 10] – would allow investigation of the effectiveness of dual-modality instruction and tools in helping students learn more complex computer science concepts. We could also investigate whether there are meaningful trends in performance, tool usage, or perceptions along ethno-racial, age, or educational backgrounds. Investigation of these data could further help us craft instruction and curricula that better serve a diverse population in CS education. Finally, the course used in this study sought to prepare students for future work in text languages. The context of the study lends it high external validity, but it also prevents isolation of the effects of dual-modality instruction on student perception. Future work could explore whether students in courses without a transition to text would perceive dual-modality instruction the same way and use dual-modality tools similarly.

7 Conclusion

We analyzed student perceptions of dual-modality instruction based on a sample of surveys and usage logs in the context of a larger study in a college-level CS1 programming course [5]. More than half said dual-modality instruction was helpful at the beginning of the course, with the highest percentages among those with no experience or experience in blocks. Several students expressed growing comfort in text as the course progressed. We also found that, even at the end of the class, more than one-third of the students still found dual-modality instruction helpful; this was true for all prior experience groups, suggesting that all students can benefit to some degree from dual-modality instruction. Additionally, contrary to our expectations, we found that students used the dual-modality plugin primarily as a visualization tool and as a way to verify structure of code and how constructs connect, rather than as a means to develop the code through drag-and-drop program development. The results of our work suggest dual-modality instruction helps students understand the structure and meaning of code by helping them visualize constructs, their connections, and their functions, regardless of prior experience.

ACKNOWLEDGMENTS

We extend our thanks to Amanpreet Kapoor and Cheryl Resch for their help in analyzing data for this paper. We also thank Aishat Aloba and Shaghayegh Esmaeili for their support in conducting the larger study.

REFERENCES

[1] Auerbach, C. and Silverstein, L.B. 2003. Qualitative data: An introduction to coding and analysis. *Qualitative data: An introduction to coding and analysis*. NYU press. 31–87.

[2] Bau, D., Bau, D.A., Pickens, C.S. and Dawson, M. 2015. Pencil Code: Block Code for a Text World. *Proceedings of the 14th International Conference on Interaction Design and Children* (2015), 445–448.

[3] Begel, A. and Klopfer, E. 2007. StarLogo TNG: An Introduction to Game Development. *Journal of E-Learning*. 53, (2007), 146.

[4] Blanchard, J., Gardner-McCune, C. and Anthony, L. 2019. Effects of Code Representation on Student Perceptions and Attitudes Toward Programming. *2019 IEEE Symposium on Visual Languages and Human-Centric Computing* (2019), 127–131.

[5] Blanchard, J., Gardner-McCune, C. and Anthony, L. 2020. Dual-Modality Instruction and Learning: A Case Study in CS1. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (2020), 818–824.

[6] Blanchard, J., Gardner-McCune, C. and Anthony, L. 2019. Amphibian: Dual-Modality Representation in Integrated Development Environments. *Proceedings of the 2019 IEEE Blocks and Beyond Workshop (Blocks and Beyond)* (2019), 83–85.

[7] Brockwell, P.J. and Davis, R.A. 2002. *Introduction to Time Series and Forecasting*. Springer.

[8] DiSalvo, B., Guzdial, M., Bruckman, A. and McKlin, T. 2014. Saving face while geeking out: Video game testing as a justification for learning computer science. *Journal of the Learning Sciences*. 23, 3 (2014), 272–315. DOI:https://doi.org/10.1080/10508406.2014.893434.

[9] Gwet, K.L. 2008. Computing inter-rater reliability and its variance in the presence of high agreement. *British Journal of Mathematical and Statistical Psychology*. 61, 1 (2008), 29–48. DOI:https://doi.org/10.1348/000711006X126600.

[10] Henriksen, P. and Kolling, M. 2004. Greenfoot: Combining Object Visualization with Interaction. *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications - OOPSLA '04*. (2004), 73. DOI:https://doi.org/10.1145/1028664.1028701.

[11] Hsu, K.C. 1996. *Developing Programming Environments for Programmable Bricks*. Ph.D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA.

[12] Python PetName. Retrieved August 29, 2020: *https://github.com/dustinkirkland/python-petname*.

[13] Malan, D.J. and Leitner, H.H. 2007. Scratch for budding computer scientists. *ACM SIGCSE Bulletin*. 39, 1 (2007), 223–227. DOI:https://doi.org/10.1145/1227504.1227388.

[14] Maloney, J., Resnick, M. and Rusk, N. 2010. The Scratch programming language and environment. *ACM Transactions on Computing Education*. 10, 4 (2010), 1–15. DOI:https://doi.org/10.1145/1868358.1868363.http.

[15] Margolis, J. and Fisher, A. 2002. *Unlocking the clubhouse: women in computing*. MIT Press.

[16] Nikiforos, S., Kontomaris, C. and Chorianopoulos, K. 2013. MIT Scratch : A Powerful Tool for Improving Teaching of Programming. *Conferrence on Informatics in Education* (2013), 11–12.

[17] Pausch, R., T. Burnette, A. C. Capehart, M. Conway, D. Cosgrove, R. DeLine, J. Durbin, R. Gossweiler, S. Koga, and J.W. 1995. A brief architectural overview of Alice, a rapid prototyping system for virtual reality. *IEEE Computer Graphics and Applications (1995), 195-203.*

[18] Repenning, A. 1993. Agentsheets: A Tool for Building Domain-Oriented Visual Programming Environments. *CHI '93: Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. (1993), 142–143. DOI:https://doi.org/10.1145/169059.169119.

[19] Viera, A.J. and Garrett, J.M. 2005. Understanding Interobserver Agreement: The Kappa Statistic. *Family Medicine*. May (2005), 360–363. DOI:https://doi.org/Vol. 37, No. 5.

[20] Ward, B., Marghitu, D., Bell, T. and Lambert, L. 2010. Teaching computer science concepts in Scratch and Alice. *Journal of computing Sciences in Colleges*. 26, 2 (2010), 173–180.

[21] Weintrop, D. 2016. *Modality Matters: Understanding the Effects of Programming Language Representation in High School Computer Science Classrooms*. Ph.D. Dissertation. Northwestern University, Evanston, IL.

[22] Weintrop, D. and Holbert, N. 2017. From Blocks to Text and Back: Programming Patterns in a Dual-modality Environment. *Proceedings of the 48th ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. (2017). DOI:https://doi.org/10.1145/3017680.3017707.

[23] Yardi, S. and Bruckman, A. 2007. What Is Computing? Bridging the Gap Between Teenagers ' Perceptions and Graduate Students ' Experiences Categories and Subject Descriptors. *Proceedings of the third international workshop on Computing education research*. (2007), 39–49. DOI:https://doi.org/10.1145/1288580.1288586.

[24] IntelliJ IDEA. Retrieved August 29, 2020: *https://www.jetbrains.com/idea/*.