Bringing Blocks-based Programming into High School Computer Science Classrooms

David Weintrop & Uri Wilensky
Northwestern University

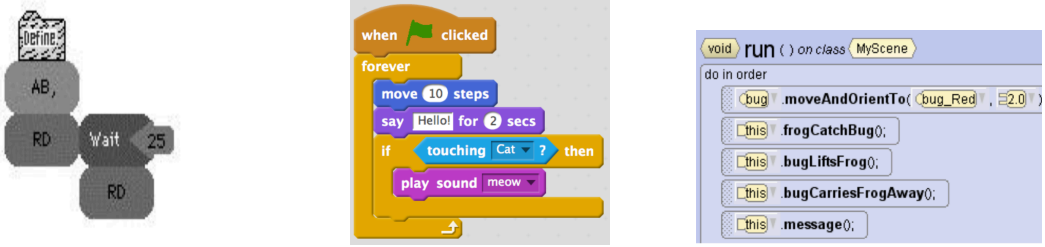
Abstract
Over the last decade, blocks-based programming has steadily been moving from the informal settings it was initially designed for, into more conventional educational contexts. Inspired in large part by the successes such tools have had at engaging novices in programming, these drag-and-drop graphical programming environments are now a central component of numerous curricula designed for high school computer science classrooms. In this paper, we explore some of the consequences, both positive and negative, associated with using blocks-based programming tools in introductory high school computer science courses, including issues relating to student perceptions of blocks-based tools, the shifting role of the teacher with differing tools, and open questions surrounding learning and transfer with blocks-based programming.

Introduction

Blocks-based programming is increasingly becoming the way that younger learners are being introduced to programming and computer science more broadly. Led by the popularity of tools like Scratch, Snap! and Code.org's suite of Hour of Code activities, millions of kids are engaging with programming through drag-and-drop graphical tools. Due in part to the success of such tools at engaging novices in programming, these environments are increasingly be incorporated into curricula designed for high school computer science classrooms. Notably, national curricular efforts including Exploring Computer Science (Goode, Chapman, & Margolis, 2012), the CS Principles project (Astrachan & Briggs, 2012), and Code.org's curricular offerings utilize blocks-based tools to introduce students to programming. This decision is not entirely unproblematic as the formal setting and older learners found in high school classrooms is quite distinct from the younger learners and informal contexts that most blocks-based programming environments were initially designed for. In this paper, we elucidate some consequences, both positive and negative, of the decision to bring this programming approach into high school classrooms. The goal of this work is to bring an analytical lens to the decision to incorporate blocks-based tools in high school curricula with the hope that by better understanding the strengths and drawbacks of the modality in formal contexts, we can ultimately improve learners' early programming experiences and better prepare them for future computer science learning opportunities.

Blocks-based Programming

Blocks-based programming environments are a variety of visual programming languages that leverage a primitives-as-puzzle-pieces metaphor (Figure 1). In such environments, learners can assemble functioning programs using only a mouse by snapping together instructions and receiving visual (and sometime audio) feedback informing the user if a given construction is valid. Each block provides visual cues to the user on how and where the block can be used through the block's shape, its color (which is associated with categories of similar blocks), and the use of natural language labels on the block to communicate its function. Along with the visual information depicted by each block, the construction space in which the blocks are used also provides various forms of scaffolding. Collectively, these features contributed to the perceived ease-of-use of blocks-based programming (Weintrop & Wilensky, 2015a).



(A) LogoBlocks (B) Scratch (C) Alice

Figure 1. Three examples of blocks-based programming tools.

Blocks-based programming is a relatively recent addition to the long line of programming languages and environments designed explicitly with learners in mind (for reviews of this work, see: Duncan, Bell, & Tanimoto, 2014; Guzdial, 2004; Kelleher & Pausch, 2005). The earliest language designed explicitly for children, and a direct influence on many blocks-based programming tools, is the Logo programming language (Feurzeig, Papert, Bloom, Grant, & Solomon, 1970). The Logo language introduced a number of characteristics that feature prominently in blocks-based programming environments, notably, the use of egocentric motion commands, the presence of onscreen avatars to carryout those commands (Logo had the turtle, while newer environments have sprites) and the emphasis on learner-directed construction and exploration, and the importance of learners creating public, sharable artifacts, often in the form of artwork, games, and interactive stories (Harel & Papert, 1991; Papert, 1980).

In recent years, there has been a proliferation of programming environments that utilize a blocks-based approach. Well known block-based programming environments such as Scratch (Resnick et al., 2009) and Alice (Cooper, Dann, & Pausch, 2000) provide learners with open-ended, exploratory spaces designed to support creative activities like story telling and game making. With the rise in popularity of these and other similar tools, the number of activities a learner can engage with through blocks-based programming is growing increasingly diverse. For example, you can develop mobile applications with MIT App Inventor and Pocket Code (Slany, 2014), build and interact

with computational models with DeltaTick (Wilkerson-Jerde & Wilensky, 2010), Frog Pond (Horn et al., 2014) or StarLogo TNG (Begel & Klopfer, 2007), create artistic masterpieces with Turtle Art (Bontá, Papert, & Silverman, 2010) or Pencil Code (Bau et al., 2015), and play video games like RoboBuilder (Weintrop & Wilensky, 2012) and CodeSpells (Esper, Foster, & Griswold, 2013). Similarly, informal computer science education initiatives are increasingly relying on blocks-based programming, including the activities provided as part of Code.org's Hour of Code and Google's Made with Code initiative. Further, we expect this trend to continue as a growing number of libraries are making it easy to develop environments that incorporate a blocks-based programming interface (Fraser, 2013; Roque, 2007).

Methods and Participants

The data presented in this paper are part of a larger study comparing blocks-based, text-based, and hybrid blocks/text programming environments at a selective enrollment public high school in a Midwestern city. We followed students in three sections of an elective introductory programming course for the first 10 weeks of the school year. Each class spent the first five weeks of the course working in a form of blocks-based programming environment then transitioned to Java for the second five weeks of the study. Two teachers participated in this study (one teacher taught two of the classes), both of whom have over five years of teaching high school computer science and have previously taught the course. A variety of data were collected as part of the study including pre/mid/post attitudinal and content assessments, clinical interviews with the students and teachers.

A total of 90 students across three sections of a Programming I course participated in the study, which included 67 male students and 23 female students. The students participating in the study were 43% Hispanic, 29% White, 10% Asian, 6% African American, and 10% Multi-racial - a breakdown comparable to the larger student body. The classes included one student in eighth grade, three high school freshman, 43 sophomores, 18 juniors, and 25 high school seniors. Two-thirds of the students in these classes speak a language other than English in their homes.

Blocks-based Programming in High School Classrooms

In this section we discuss consequential aspects of bringing blocks-based programming into high-school computer science classrooms.

Perceived of Lack of Applicability Beyond the Classroom

One of the main differences between younger (elementary and middle-school aged learners) and high school aged students is their motivation for learning to program. Unlike younger learners, high school students who choose to enroll in a programming elective are often concerned with the direct applicability of what they are learning. As a result, some students take issue with a perceived inauthenticity of the blocks-based modality. This can be seen in student responses to an open-ended question from our post-

survey asking students to compare blocks-based program and Java. "*Java is actual code, while* [blocks-based programming] *is something nobody will let you code in,*" wrote one student, a second student echoed this sentiment saying: "*If we actually want to program something, we wouldn't have blocks.*" In both of these responses, students are thinking about programming beyond the classroom and recognizing how blocks-based tools are largely reserved for educational contexts. This position is consistent with other work done on older learners working in blocks-based tools where high school aged learners show a preference for text-based languages (DiSalvo, 2014).

One potential way to address this perceived lack of authenticity is to make clear the relationship between blocks-based and text-based modalities. There are a number of ways to do this, including isomorphic tools that allow learners to move back and forth between the two representations (Bau et al., 2015; Matsuzawa, Ohata, Sugiura, & Sakai, 2015) or providing ways to view text-based versions of programs authored in a blocks-based interfaces (Weintrop, Wilensky, Roscoe, & Law, 2015). In making explicit this link, teachers can directly confront the perception of the limited scope of blocks-based tools by showcasing the isomorphic features of graphical introductory tools and the text-based languages they will encounter in the future.

It is also worth noting that there are some upsides to this perceived inauthenticity. For example, the more inviting, playful feel of the blocks-based interface can contribute to a productive classroom culture; as one teacher commented: "[Blocks-based programming] *creates a different feel to the room...blocks take away the foreign feel, it looks friendly, and it's something you can do right away, and because of that, the culture in the room is different, kids are more prone to talk to their neighbors, more prone to feel OK about joking around.*" This characteristic of blocks-based tools helps them excel in informal spaces and can productively change the culture of formal classrooms.

Open Questions Around Transfer to Text-based Languages

A second potential drawback to the inclusion of blocks-based programming in high school classrooms stems from the open question of if and how understandings and practices developed in blocks-based tools transfer to more conventional text-based languages. Studies have reported both successful transfer between modalities (Armoni, Meerbaum-Salant, & Ben-Ari, 2015; Dann, Cosgrove, Slater, Culyba, & Cooper, 2012) and found difficulties with students transferring concepts and practices from blocks-based to text-based tools (Chetty & Barlow-Jones, 2012; Cliburn, 2008; Mullins, Whitfield, & Conlon, 2009; Powers, Ecott, & Hirshfield, 2007). Direct comparison between conceptual understanding in blocks-based and text-based languages found there to be some concepts that are more accessible in the blocks-based modality, but that this benefit is far from universal across programming concepts (Lewis, 2010; Weintrop & Wilensky, 2015b). These conflicting findings underscore the importance of better understanding the inclusion of blocks-based tools in formal high-school contexts as the assumed transfer of

learned concepts and developed programming practices underpins the motivation for using blocks-based tools in formal computer science contexts.

In our study, teacher encountered this reported lack of conceptual transfer. When asked about how concepts carried over from the blocks-based introduction to Java, one teacher said the transition was "*rough, I think* [the students] *lost what they were doing* [in the blocks-based tool] *with what they were doing in Java.*" The extended gap between when students first encounter concepts at the outset of the year and when they are reintroduced in Java is problematic. One way to address this challenge is to have tools that let you move back and forth more fluidly, like Pencil Code (Bau et al., 2015), or to consistently move back and forth between blocks and text over the course of the year, which is another approach that has been used in classes (Matsuzawa et al., 2015).

One feature that is shared by the studies that found successful transfer of concepts and practices was an explicit focus on preparing students for their eventual move from blocks-based tools to text-based languages. Throughout students' use of the introductory graphical environments, the teachers made it a point to link the blocks-based representation and activities with the text-based languages learners were going to encounter in the future. This suggests that, at least in part, there are pedagogical strategies that can be used to effectively bridge blocks-based and text-based programming.

Shifting Role of the Teacher

One major difference between formal and informal learning spaces is the presence of an expert who can provide guidance and support. Teachers can not only provide instruction, they also design the curriculum, providing a carefully crafted sequence of activities to smoothly move learners from accessible to more challenging concepts, alleviating the need for the environment itself to play these supportive roles. As such, the scaffolds that blocks-based tools provide change the role of the teacher in the classroom. For example, when teaching text-based languages, early classroom time is spent discussing unintuitive, yet necessary, syntax features and walking students through program compilation and execution procedures, while early blocks-based classes allow students to dive right into the code. While this beneficial with respect to student confidence and engagement, there are potential drawbacks to the intuitive nature as one of our teachers commented: *"the point of the environment is that it shouldn't generate a whole lot of questions, like 'how do I do this?' - it's more intuitive.*" The teacher goes on to explain that while this is empowering for the learner, it gives him fewer opportunities to engage in productive discussions on different aspects of programming. Additionally, in classrooms using blocks-based tools, the teacher can spend less time standing in front of the class lecturing and instead, focus on working one-on-one with students who get stuck.

The shift in the role of the teacher is often accompanied by new, or at least different, pedagogical strategies and classroom orchestration techniques. Being able to support these different types of classrooms requires a level of comfort and confidence with the material that not all teachers possess. This shift is compounded in classrooms

where students will eventually transition to text-based tools, as that will potentially necessitate the teacher changing his or her pedagogical strategy over the course of the school year.

Similarities between High School Classrooms and Computer Clubhouses

In some cases, the challenges faced by younger learners in informal spaces mirror those of older learners in classrooms. For example, in both cases, learners struggle with issues of syntax and benefit from the easy of composition provided by the blocks-based interface. Similarly, the visual outcomes and easy of browsing available commands play the same role for younger and older learners. When asked to compare the blocks-based introductory class with a text-based alternative, one teacher responded, with blocks "*students feel like they can do more right away*."

A second important strength shared by computer education opportunities in both formal and informal context is their difficulty in recruiting female learners and minority students. Blocks-based tools like Alice and Scratch have been found to be effective at attract students who are underrepresented in computing fields (Kelleher, Pausch, & Kiesler, 2007; Maloney, Peppler, Kafai, Resnick, & Rusk, 2008). The engaging, accessible, and convivial aspects of blocks-based programming tools are productive in both formal and informal settings. Similarly, the ease with which programs can be personalized, presented, and shared, all aid students in creating personally meaningful projects that they are eager and proud to share, which promotes deep, meaningful learning (Papert, 1980).

There are also some features of blocks-based tools that lend themselves well to formal spaces. For example, the ease of browsing available commands makes it possible for learners to discover and tinker with new programming constructs with little (or no) formal introduction (Weintrop & Wilensky, 2013). This makes it possible for learners who are more experienced, or more adventurous, to go beyond what has been covered in class. This discoverability can keep learners of various level engaged.

Conclusions

In the last five years, the blocks-based programming approach that has changed how younger learners are being introduced to programming in informal spaces has arrived in high school classrooms. The inclusion of graphical drag-and-drop programming tools in formal contexts with older learners is not as straightforward as simply changing the program learners open up at the start of class, but instead includes a number of differences that can effect the learning experience. These differences include students' perception of the programming tool and its utility, issues of transfer between programming environments, and shifting roles of the teacher in classrooms using blocks-based tools. As more and more curricula incorporate drag-and-drop programming tools, it is important that we understand the effects of using such tools in formal classrooms with older learners. Our hope is that in gaining a better understanding of how blocks-based

tools fit into formal spaces we can take full advantage of what the modality provides and also better equip teachers to effectively incorporate them into their classrooms.

References

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "Real" Programming. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 25:1–15.

Astrachan, O., & Briggs, A. (2012). The CS principles project. *ACM Inroads*, *3*(2), 38–42.

Bau, D., Bau, D. A., Dawson, M., & Pickens, C. S. (2015). Pencil Code: Block Code for a Text World. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 445–448). New York, NY, USA: ACM.

Begel, A., & Klopfer, E. (2007). Starlogo TNG: An introduction to game development. *Journal of E-Learning*.

Bontá, P., Papert, A., & Silverman, B. (2010). Turtle, Art, TurtleArt. In *Proceedings of Constructionism 2010 Conference*. Paris, France.

Chetty, J., & Barlow-Jones, G. (2012). Bridging the Gap: the Role of Mediated Transfer for Computer Programming. *International Proceedings of Computer Science & Information Technology*, *43*.

Cliburn, D. C. (2008). Student opinions of Alice in CS1. In *Frontiers in Education Conference, 2008. FIE 2008. 38th Annual* (p. T3B–1). IEEE.

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, *15*(5), 107–116.

Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated transfer: Alice 3 to Java. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 141–146). ACM.

DiSalvo, B. (2014). Graphical Qualities of Educational Technology: Using Drag-and-Drop and Text-Based Programs for Introductory Computer Science. *IEEE Computer Graphics and Applications*, (6), 12–15.

Duncan, C., Bell, T., & Tanimoto, S. (2014). Should Your 8-year-old Learn Coding? In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 60–69). New York, NY, USA: ACM.

Esper, S., Foster, S. R., & Griswold, W. G. (2013). CodeSpells: embodying the metaphor of wizardry for programming. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (pp. 249–254). ACM.

Feurzeig, W., Papert, S., Bloom, M., Grant, R., & Solomon, C. (1970). Programming-languages as a conceptual framework for teaching mathematics. *SIGCUE Outlook*, *4*(2), 13–17.

Fraser, N. (2013). *Blockly*. https://code.google.com/p/blockly: Google.

Goode, J., Chapman, G., & Margolis, J. (2012). Beyond curriculum: the exploring computer science program. *ACM Inroads*, *3*(2), 47–53.

Guzdial, M. (2004). Programming environments for novices. *Computer Science Education Research*, *2004*, 127–154.

Harel, I., & Papert, S. (Eds.). (1991). *Constructionism*. Norwood N.J.: Ablex Publishing.

Horn, M. S., Brady, C., Hjorth, A., Wagh, A., & Wilensky, U. (2014). Frog pond: a codefirst learning environment on evolution and natural selection. In *Proceedings of the 2014 conference on Interaction design and children* (pp. 357–360). ACM.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, *37*(2), 83–137.

Kelleher, C., Pausch, R., & Kiesler, S. (2007). Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 1455–1464).

Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: Logo vs. Scratch. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 346–350). New York, NY.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with Scratch. *ACM SIGCSE Bulletin*, *40*(1), 367–371.

Matsuzawa, Y., Ohata, T., Sugiura, M., & Sakai, S. (2015). Language Migration in non-CS Introductory Programming through Mutual Language Translation Environment. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 185–190). ACM Press.

Mullins, P., Whitfield, D., & Conlon, M. (2009). Using Alice 2.0 as a first language. *Journal of Computing Sciences in Colleges*, *24*(3), 136–143.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic books.

Powers, K., Ecott, S., & Hirshfield, L. M. (2007). Through the looking glass: teaching CS0 with Alice. *ACM SIGCSE Bulletin*, *39*(1), 213–217.

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., … Silver, J. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60.

Roque, R. V. (2007). *OpenBlocks: An extendable framework for graphical block programming systems* (Master's Thesis). Massachusetts Institute of Technology.

Slany, W. (2014). Tinkering with Pocket Code, a Scratch-like programming app for your smartphone. In *Proceedings of Constructionism 2014*. Vienna, Austria.

Weintrop, D., & Wilensky, U. (2012). RoboBuilder: A program-to-play constructionist video game. In C. Kynigos, J. Clayson, & N. Yiannoutsou (Eds.), *Proceedings of the Constructionism 2012 Conference*. Athens, Greece.

Weintrop, D., & Wilensky, U. (2013). Supporting computational expression: How novices use programming primitives in achieving a computational goal. Presented at the American Education Researchers Association, San Francisco, CA, USA.

Weintrop, D., & Wilensky, U. (2015a). To Block or Not to Block, That is the Question: Students' Perceptions of Blocks-based Programming. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 199–208). New York, NY, USA: ACM.

Weintrop, D., & Wilensky, U. (2015b). Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. In *Proceedings of the Twelth Annual International Conference on International Computing Education Research*. Omaha, NE.

Weintrop, D., Wilensky, U., Roscoe, J., & Law, D. (2015). Teaching Text-based Programming in a Blocks-based World. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 678–678). New York, NY, USA: ACM.

Wilkerson-Jerde, M. H., & Wilensky, U. (2010). Restructuring Change, Interpreting Changes: The DeltaTick Modeling and Analysis Toolkit. In J. Clayson & I. Kalas (Eds.), *Proceedings of the Constructionism 2010 Conference*. Paris, France.