Position: Building Blocks for Agent-based Modeling Can Scaffold Computational Thinking Engagement in STEM Classrooms

Connor Bain, Gabriella Anton, Michael Horn, and Uri Wilensky
Department of Computer Science & Department of Learning Sciences
Northwestern University
Evanston, IL, USA
{connorbain, gabriella.anton}@u.northwestern.edu
{michael-horn, uri}@northwestern.edu

***Abstract*—Computational models and simulations can be powerful tools to help learners understand a wide variety of natural phenomena. However, understanding and learning from computational models requires learners to comprehend the rules agents follow that lead to emergent outcomes. Blocks-based programming is useful in scaffolding learners in the early stage of programming tasks. We posit that students can authentically interact with agent-based modeling via a blocks-based dialect of the popular ABM language NetLogo, dubbed NetTango, across many difficulty levels. Specifically, we discuss three different activities in which we have integrated blocks-based programming into STEM classrooms that show blocks can be used for activities of increasing computational engagement and difficulty.**

***Index Terms*—agent-based modeling, NetLogo, STEM education, blocks-based programming**

I. Introduction

Blocks-based programming has emerged as a popular approach to introducing younger learners to programming. With accessible tools like Scratch [1], Snap! [2] and Code.org's suite of Hour of Code activities [3], millions of young learners 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 being incorporated into curricula designed for high school computer science classrooms, like Exploring Computer Science [4], the CS Principles project [5], and Code.org's curricular offerings [6].

However this focus on computer science learning environments has resulted in many blocks-based environments being viewed by students [7], teachers [8], and even researchers [9] as transitory environments—temporary scaffolds that students can rely on while they progress toward true text-based programming [9], [7]. In a traditional computer science classroom, this attitude aligns with the student progression towards mastery of a particular language, which is often text-based in high school and college courses. However, in a classroom focused on learning some other discipline, a blocks-based environment can provide a multitude of differing learning opportunities to engage in computational and content knowledge and practice in addition to acting as digital scaffold toward more advanced computational activities. This project focuses on three different ways of integrating blocks-based environments in agent-based modeling tasks in high school STEM classrooms.

Agent-based modeling (ABM) is a form of computational modeling whereby a phenomenon is modeled in terms of agents and their interactions. By looking at scientific phenomenon using ABMs, students can learn to understand the micro interactions that are responsible for so many of the seemingly complex systems in the world around them [10], [11], [12], [13], [14]. NetTango [15], is a new blocks-based dialect of the popular NetLogo [16] modeling environment, in which authors identify and create the block primitives from which students later can leverage for coding models. These technological tools provide the opportunity to enrich STEM classes with computational thinking (CT) and computer science practices that are aligned or fully integrated with the content knowledge being studied. In order to best leverage the advantages of blocks-based languages [17], [18], [19], we must carefully consider how we might integrate a blocks-based language into STEM classrooms such that students engage in authentic CT-STEM practices [20], [21] without backgrounding the science content that STEM classrooms are centered around. Here, we present three different approaches to integrating NetTango into existing science curricula, each designed for students to gain differing experiences with computational thinking practices in the STEM classroom.

II. Using blocks as an introductory computational activity

Block-based activities can be used as an introductory activity that scaffolds student thinking and practice with computational tools [22], [23], [24]. For many students in STEM classes, computational thinking or computational practices are not a focal learning outcome or experience. The practices and ways of thinking with and about computation can be

This work was made possible through generous support from the National Science Foundation (grants CNS-1138461, CNS-1441041 and DRL-1020101), the National Science Foundation Graduate Research Fellowship Program, and the Spencer Foundation (Award # 201600069).

978-1-7281-4849-6/19/$31.00 ©2019 IEEE

challenging to learn, especially for those students without prior exposure or those who perceive themselves explicitly as a non-computer science person. Thus, block-based activities can be used to help students begin to think about how to interact with these challenging and possibly intimidating ideas and tools while in the context of STEM content knowledge.

This approach to using blocks as an introductory computational activity was leveraged in the design of a Chemistry unit on the ideal gas laws. In this unit, students use agent-based models to explore the phenomenon of air pressure, discover that it results from air particles hitting the walls of containers, and that it is impacted by number of particles, temperature and volume. Rather than having students immediately interact with full agent-based models, the curriculum was designed to scaffold learners into increasingly complex computational practices to explore increasingly complex science content. In this unit, students' first exposure to computational practice is to code a simple agent-based model to make two particles move "correctly". Students expressed their prior knowledge about particle behavior by coding models that reflected their personal beliefs about how particles should behave (see Figure 1). The blocks themselves are based on intuitive physical understanding in the world, more like using p-prims [25], like including the description of motion in terms of "straight" or "zigzag" and bouncing in relation to types of balls like balloons or billiards. As students code their models, the blocks provided prompted students to be explicit in describing certain interactions or behaviors related to the phenomenon while scaffolding their intuitions through the primitives provided. For example, in one episode of student talk, Yolanda and Karen (pseudonyms) discuss why Yolanda's particles were flying away in her model. They discover that Yolanda hasn't written any rules that define the action of the particles when they hit the wall. When they discover that mistake, they then immediately move to discuss the differences between their models as Yolanda has chosen for particles to bounce like basketballs while Karen has decided particles will bounce like billiard balls.

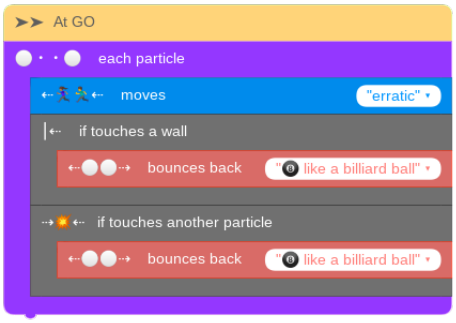

Fig. 1. An example of using "phenomenological" blocks to program gas particle behavior.

This approach also focuses on developing an adequate space to have students express their ideas, gain confidence with the computational integration, and perceive a purpose in the practice of using computation in their classroom. Without this sort of framing, students might leave a typical computational activity feeling more intimidated or confused about the role of computational tools in STEM.

III. Using Blocks as a Scaffold to Transition to Text-Based Languages

A more traditional use of blocks-based programming is to explicitly scaffold students' future text-based programming practices using blocks. For example, in one of our biology units, Modeling Ecosystems Computationally, students explore prey-predator ecosystems with a primary focus on population dynamics. Interacting species have enormous effects on the population dynamics of each other and modeling these effects computationally is a central part of this unit. Specifically, students are asked to write the rules governing the actions of wolves and moose on Isle Royale, an island in the northernmost part of the state of Michigan (U.S.A). This island features a nearly untouched ecosystem of wolves and moose that live in relative stability. In this curricula, students explore how the concept of population dynamics arise as the result of simple interactions between agents and their environment. While the goal of the unit is to have students create a fully functional NetLogo model of Isle Royale, similar to the classical Wolf-Sheep Predation model [26], most students have no NetLogo or programming experience at all. In order to scaffold their experimentation with both programming and agent-based modeling, we use blocks that directly translate to traditional text-based commands. In essence, the blocks-serve as a palette of sorts from which students draw [7]. Using the well-known benefits of blocks-based languages [27], [28], students are able to quickly create simple models of the wolf-moose ecosystem, as shown in Figure 2.

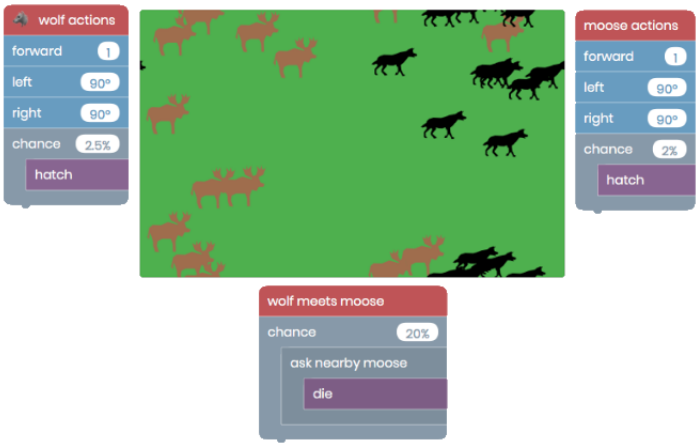

Fig. 2. An example ecosystem programmed by a student.

However, as their goal is to recreate the ecosystem at Isle Royale, students quickly express feelings of constraint when they are unable to easily add new elements to the model. As the students discuss how their model differs with the real world, they are guided to consider the text-based code behind their

existing models (see Figure 3). Due to the direct mapping from blocks-to-text, we see students quickly begin to author new code in text-based form rather than relying solely on the blocks-based modality. In this manner, students naturally pick up the text-based languages rather than expressing intimidation, which takes focus away from the underlying scientific content.

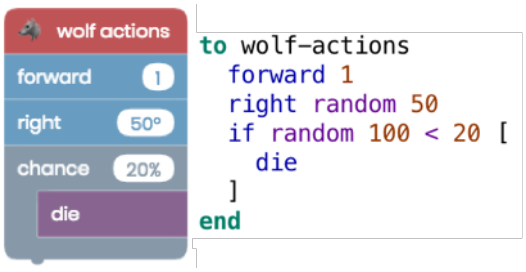

Fig. 3. The underlying code of a set of NetTango blocks.

IV. Build Your Own Blocks: Using blocks as a thinking tool for complex systems

While in the prior two cases, blocks are used as an introduction into more complex computational practice, blocks-based programming could also be used as a way of organizing later, more advanced, computational tasks. In many of our curricula, students are asked to design their own blocks that they might add to the existing palette after having interacted with several NetTango models. We have completed a partial pass of inductive coding of student responses, and have found that students attend to a range of different elements relating to block design. Many students wanted to add additional features, not necessarily related to the scientific content at play, to the environment via blocks. For instance, one student in the chemistry unit wished to add a block so that "If particle splits, a nuclear explosion occurs," while in the biology unit, many students requested "adding a sound block" to make the wolves and moose in the model create sound as they moved. However, a large group of students seems to be using the blocks as thinking tools for agent actions that are related to the science content at hand. In a chemistry unit, one student wanted to add a block so that "if molecules touched they would form a bond" to "model covalent and ionic bonds." In a biology unit, a number of students advocated for a "attack moose" action block for wolves because they thought it would make the rules governing the wolves "more realistic."

We see this as an opportunity to use blocks-based programming to leverage student thinking on complex systems. ABM encourages students to think about complex natural phenomenon in terms of agents performing simple actions. A programming block is a useful thinking tool to abstract larger agent-actions (like "attack moose") into a single programming entity that could be specified later. In future curricular designs, we plan to use this category of activity as a capstone activity, where students are asked to design a set of blocks for future students in the course that will allow them to learn the same science content. More advanced students could be asked to actually implement those blocks in the underlying NetLogo code, similar to how BYOB/Snap! [29] and Scratch [2] allow learners to create higher-order blocks. In this manner, blocks serve not just as a starting point, but as a way of gaining insight into student thinking and a way of scaffolding further student involvement in model building practices.

V. Discussion and Future Work

In each of these three pedagogical designs, we recently collected student data across several two-week curricular implementations in high school STEM classrooms. While preliminary data suggests that each of the three designs encourages student participation engagement in the underlying science content, we have only just begun to analyze to what extent this engagement is successful in helping students engage in authentic computational practices and allowing students to gain a deep understanding of the scientific content at hand.

It is important to outline the possibility of expanding the horizon of what blocks-based programming languages can be used for in learning contexts outside of traditional computer science—where learning to program is secondary to actual scientific content. While these three pedagogical designs are preliminary, they serve as jumping off points for designers and thinkers leading toward new ways of positioning blocks-based programming as more than just an *initial* digital scaffolding device. In particular, designers should give thought to ways of building the difficulty-level of blocks-based programming activities when used in STEM classrooms.

However, these activities also raise interesting questions. For instance, implementations of these ideas could background key computational ideas like loops, conditionals, and more. Even in our own designs, we see elements of "blocks-based languages as a puzzle" rather than a true "programming environment." This could cause students to see computer science not as a creative, powerful, and flexible discipline, but instead as yet another activity that is simply a required part of classroom practice. Where does the line lie between these two types of environments? What are the possible benefits and detriments of students interacting with such environments? In addition, how can teachers effectively integrate these activities and environments into their classrooms without extensive professional development? These are all open questions that require further discussion and study.

Our position is that blocks-based programming activities can be leveraged as a way of interacting with science content, particularly ABM, across a spectrum of difficulty levels. From introducing students to computational ideas to being a literal "building block" for students to think about complex systems with, the blocks-based paradigm can offer STEM classrooms many different forms of computational engagement. These activities are crucial to developing a computational STEM classroom through which we can 1) broaden participation in computing; 2) engage students in authentic computational and scientific practices; and 3) deepen scientific content understanding.

REFERENCES

[1] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. S. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all." *Commun. ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[2] B. Harvey and J. Mönig, "Bringing "No ceiling" to Scratch: Can one language serve kids and computer scientists?" *Proc. Constructionism*, pp. 1–10, 2010.

[3] "Hour of code." [Online]. Available: https://code.org/learn

[4] J. Goode, G. Chapman, and J. Margolis, "Beyond curriculum: the exploring computer science program," *ACM Inroads*, vol. 3, no. 2, pp. 47–53, 2012.

[5] O. Astrachan and A. Briggs, "The CS Principles project," *ACM Inroads*, vol. 3, no. 2, pp. 38–42, 2012.

[6] "What will you create?" [Online]. Available: https://code.org/

[7] D. Weintrop, "Modality matters: Understanding the effects of programming language representation in high school computer science classrooms," Ph.D. dissertation, Northwestern University, 2016.

[8] S. Dabholkar, A. Peel, G. Anton, M. Horn, and U. Wilensky, "Analysis of teachers involvement in co-design and implementation of CT integrated biology units," Unpublished.

[9] K. Martin, U. Aslan, C. Bain, S. Dabholkar, M. Horn, and U. Wilensky, "Programing science phenomena with NetTango?" Unpublished.

[10] U. Wilensky and K. Reisman, "Thinking like a wolf, a sheep, or a firefly: Learning biology through constructing and testing computational theories—an embodied modeling approach," *Cognition and Instruction*, vol. 24, no. 2, pp. 171–209, 2006.

[11] U. Wilensky, "Gaslab—an extensible modeling toolkit for connecting micro-and macro-properties of gases," in *Modeling and Simulation in Science and Mathematics Education*. Springer, 1999, pp. 151–178.

[12] D. Abrahamson and U. Wilensky, "Problab goes to school: Design, teaching, and learning of probability with multi-agent interactive computer models," in *Proceedings of the Fourth Conference of the European Society for Research in Mathematics Education. San Feliu de Gixols, Spain*, 2005.

[13] M. H. Wilkerson-Jerde and U. Wilensky, "Restructuring change, interpreting changes: The DeltaTick modeling and analysis toolkit," *Proceedings of Constructionism*, 2010.

[14] S. T. Levy and U. Wilensky, "Students' learning with the connected chemistry (CC1) curriculum: Navigating the complexities of the particulate world," *Journal of Science Education and Technology*, vol. 18, no. 3, pp. 243–254, 2009.

[15] I. C. Olson and M. S. Horn, "Modeling on the table: agent-based modeling in elementary school with NetTango," in *Proceedings of the 10th International Conference on Interaction Design and Children*. ACM, 2011, pp. 189–192.

[16] U. Wilensky, "NetLogo. Evanston, IL: Center for connected learning and computer-based modeling, northwestern university," 1999.

[17] D. Franklin, G. Skifstad, R. Rolock, I. Mehrotra, V. Ding, A. Hansen, D. Weintrop, and D. Harlow, "Using upper-elementary student performance to understand conceptual sequencing in a blocks-based curriculum," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 2017, pp. 231–236.

[18] S. Grover and S. Basu, "Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 2017, pp. 267–272.

[19] D. Weintrop and U. Wilensky, "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*. ACM, 2015, pp. 199–208.

[20] D. Weintrop, E. Beheshti, M. Horn, K. Orton, K. Jona, L. Trouille, and U. Wilensky, "Defining computational thinking for mathematics and science classrooms," *Journal of Science Education and Technology*, vol. 25, no. 1, pp. 127–147, 2016.

[21] H. Swanson, G. Anton, C. Bain, M. Horn, and U. Wilensky, "Introducing and assessing computational thinking in the secondary science classroom," in *Computational Thinking Education*. Springer, 2019, pp. 99–117.

[22] R. Ridgway, "Project guts," *Science Scope*, vol. 42, no. 3, pp. 28–33, 2018.

[23] I. Lee, F. Martin, J. Denner, B. Coulter, W. Allan, J. Erickson, J. Malyn-Smith, and L. Werner, "Computational thinking for youth in practice," *ACM Inroads*, vol. 2, no. 1, pp. 32–37, 2011.

[24] E. Klopfer, H. Scheintaub, W. Huang, and D. Wendel, "StarLogo TNG," in *Artificial Life Models in Software*. Springer, 2009, pp. 151–182.

[25] A. A. DiSessa, "Toward an epistemology of physics," *Cognition and instruction*, vol. 10, no. 2-3, pp. 105–225, 1993.

[26] U. Wilensky, "Wolf sheep predation NetLogo model," 1997.

[27] C. Kelleher and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," *ACM Computing Surveys (CSUR)*, vol. 37, no. 2, pp. 83–137, 2005.

[28] D. Bau, J. Gray, C. Kelleher, J. Sheldon, and F. Turbak, "Learnable programming: Blocks and beyond," *Commun. ACM*, vol. 60, pp. 72–80, 2017.

[29] B. Harvey and J. Mönig, "Lambda in blocks languages: Lessons learned," in *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. IEEE, 2015, pp. 35–38.