Position Paper: Block-based Programming Should Offer Intelligent Support for Learners

Thomas W. Price
North Carolina State University
Raleigh, North Carolina, 27606
Email: twprice@ncsu.edu

Tiffany Barnes
North Carolina State University
Raleigh, North Carolina, 27606
Email: tmbarnes@ncsu.edu

***Abstract*—Block-based programming environments make learning to program easier by allowing learners to focus on concepts rather than syntax. However, these environments offer little support when learners encounter difficulty with programming concepts themselves, especially in the absence of instructors. Textual programming environments increasingly use AI and data mining to provide intelligent, adaptive support for students, similar to human tutoring, which has been shown to improve performance and learning outcomes. In this position paper, we argue that block-based programming environments should also include these features. This paper gives an overview of promising research in intelligent support for programming and highlights the challenges and opportunities for applying this work to block-based programming.**

I. Introduction

Block-based programming environments provide useful scaffolding to novice learners by representing program code with visual blocks, which, among other benefits, reduces the burden of syntax and often precludes syntax errors altogether. This scaffolding has been shown to improve learners' performance and reduce idle time [1] and reduce perceived difficulty [2]. However, while block-based programming helps students bypass many of the difficulties of syntax to grapple with the core concepts of programming, it does little to support students when they encounter difficulty with the programming concepts themselves, especially when an instructor is unavailable. Constructing a loop may be easier to do in Scratch than in Java, but using that loop to calculate the sum of items in a list still requires the conceptual knowledge of how the algorithm is implemented.

Programming environments increasingly use AI and data mining to provide intelligent, adaptive support for struggling students, much in the way a human tutor would. These intelligent tutoring systems (ITSs) typically help students to practice and master specific concepts while working independently (e.g. on homework) [3]. Evaluations of programming ITSs have historically shown substantial improvements in learning gains compared to traditional independent practice [4], [5]. However, nearly all ITSs for programming have targeted textual programming languages, such as Java [6], Python [7] and Haskell [8], in introductory college courses. In this position paper, we argue that block-based programming environments should also include intelligent support features, and that our research community on block-based programming should help shape these features to best support our users.

Block-based programming environments have been developed for several reasons: to support novice learners in formal learning environments (e.g. [9]), to promote exploration in informal learning settings (e.g. [10]), and to make programming more accessible to younger or less experienced learners. However, these are also settings where it may be difficult for students to get individual support from expert teachers. There is a well-documented shortage of qualified K-12 CS teachers [11], introductory college courses are growing to overwhelming sizes [12], and informal learning may involve no teacher at all. Block-based programming environments are precisely where intelligent support could be most beneficial – encouraging novices and helping them overcome difficulties that might otherwise discourage them from learning computing.

This paper will introduce some of the key ways that intelligent systems can support novices learning to program and highlight early work that applies these techniques to block-based programming environments. We will discuss the challenges and opportunities that block-based programming poses to the addition of intelligent support, and suggest ways that the community could work to further this goal.

II. Intelligent Support for Textual Programming

Intelligent support for programming takes many forms, and in this section, we will review some proven and promising approaches, with the hope that readers will be inspired by what is possible. We also envision how each intelligent support feature might look like in a block-based programming context and the research challenges that it might entail.

A. Student Models

Many ITSs keep a detailed model of each student's domain knowledge and use this "student model" to adapt instruction to individual learners. Student models usually break the problem domain down into smaller concepts, or *skills*, such as "declaring a variable" or "constructing a for-loop." The student model itself can take many forms, but it is generally a collection of data on a student's past performance that is used to estimate which skills the student has learned [3]. Using a student model, a system can adapt to a student's knowledge, giving them problems that are challenging but for which the student has mastered the prerequisite skills. Student modeling can be used in conjunction with Mastery Learning [13] to

ensure that each student practices each skill until they have mastered it, but not longer. In the Lisp Tutor, one of the first systems to implement student modeling and mastery learning, these features greatly improved student learning [4].

In Open Learner Models (OLMs), the student model is made visible to the students, so they can track their own progress and mastery. The MasteryGrids systems visually depicts students' predicted mastery of various programming topics (e.g. loops, variables), and allows them to compare their progress to the class average, a feature which was shown to improve learning for weaker students [14]. Student modeling is easiest when students engage in many small exercises, which are each designed to practice a specific skill, as in MasteryGrids. It is much more challenging in "open programming," where students work on larger programming problems that comprise many skills at once. Recent attempts at automatically modeling skills on open programming problems using log data have had some initial success. Rivers et al. used learning curve analysis on automatically-extracted skills to identify concepts students struggled with [15], and Yudelson et al. showed that an automated student model could somewhat accurately predict learning in a programming MOOC [16]. The REACT system [17] identifies evidence of "computational thinking patterns" in the AgentSheets and AgentCubes visual programming environments and visualizes this information for students and teachers.

One could imagine adding a block-based language to an existing, exercise-based practice environment with auto-graded unit tests, where new learners could benefit from the block modality, and a student model could help select ideal exercises and promote mastery. However, block-based programming environments are often paired with curricula like the Beauty and Joy of Computing (BJC) [9] that emphasize exploration, tinkering and open-ended, creative projects, rather than repetitive practice. While the block modality itself is separate from this style of learning, we propose a vision of student modeling that embraces curricula like BJC. One could imagine students pursuing creative and exploratory programming projects, while a student model passively builds knowledge of the concepts they understand or struggle with. This student model could then be used to identify students' knowledge gaps, to notify the teacher of misunderstandings, and to recommend videos, readings or extensions of programming projects that will help students master needed skills.

B. Hints and Feedback

In programming classes, students can easily get stuck or confused. If unresolved, this confusion can result in lower achievement, while confusion that is resolved leads to better learning [18]. To ensure that students do not stay stuck, some programming environments offer adaptive hints and feedback. A student can ask for help when they need it [7], [19], or the environment may offer help proactively after a student submits an incorrect attempt at a problem [8]. The help may explain an underlying domain principle that the student needs to understand to progress, or it may off an edit-based hint (e.g. an insertion or deletion) to bring a student's current code closer to a correct solution. These context-sensitive hints can be generated using an expert model, an AI system with a large amount of expert domain knowledge encoded into rules (e.g. model tracing [8]). However, programming environments are increasingly using data-driven techniques to generate hints automatically using student data [7], [19], [20], allowing these hints to scale to new problems.

Corbett and Anderson compared a variety of help mechanisms in the ACT Programming Tutor, including on-demand hints, and found that students who received any type of feedback during tutoring completed the tutor faster and completed a subsequent programming assessment in significantly less time, with significantly fewer errors [5]. Other work has shown that automatically generated hints can closely approximate the feedback that human tutors give [20], [21], suggesting that these systems have similar potential to help students.

BlockPy [22] is a block-based programming environment that already offers students sophisticated, instructor-authored feedback and could easily be augmented with intelligent support to reduce the authoring burden. Our own work has investigated how to provide on-demand hints automatically in Snap*!*, a block-based programming environment (see Section III) [19]. As in other systems, these hints are edit-based, but they differ from existing programming hints by presenting all suggestions visually, comparing a student's current code to suggested code. This choice was inspired by the visual nature of the block modality, and it raises the question of how block-based programming could shape other aspects of hints. For example, blocks lend themselves to a tinkering and bottom-up problem-solving style that is difficult for current hint-generation algorithms to support [19]. Block-based hints might therefore shift away from edit-based hints to support students with explanations, suggested experiments and demonstrations that help the student discover the next step in their code independently.

C. Other Intelligent Support

1) Dialog and Affective Support: The JavaTutor system uses natural language dialog to offer both cognitive and affective support to students, in the form of explanations and encouragement. Students can use text chat to communicate with a virtual tutor, whose responses can adapt to students' utterances [23]. Research with JavaTutor suggests that non-cognitive factors, such as engagement and frustration are important for learning to program, and that these factors can be effectively predicted using multimodal data, including facial expressions, posture and gestures [24]. This work highlights the potential of a tutor that can adapt to not only students' actions, but also their personal traits and affective state. For example, in their ENGAGE game to teach CS to middle schoolers, Buffum et al. found that affective scaffolding through a learning companion reduced females' frustration, closing the "gender gap" with their male counterparts [25]. As with ENGAGE, block-based programming is often targeted at younger populations, who may struggle with frustration or confidence, and this type of affective support, adaptation and interactivity could be particularly impactful.

2) Teacher Amplification: Intelligent support does not mean replacing the teacher, and many intelligent technologies attempt to amplify teacher efforts. Piech et al. developed a "force multiplication" technique to propagate teacher feedback on a small number of programs to a much larger number of students [26]. Their machine learning algorithm first selects a subset of programs to be graded by humans, and then applies that feedback to tens or hundreds of times as many students. Head et al. [27] developed the FIXPROPAGATOR, which allows teachers to correct a small number of student programs with detailed feedback, and the system will generalize those fixes and propagate their corresponding feedback to other students whose programs would be corrected by the same fix. Many block-based programming environments have no trouble attracting users, but often lack qualified teachers able to instruct them or provide feedback. One could easily imagine using these propagation systems to make targeted teacher feedback available to learners working on common programming projects in informal learning settings, who may not have access to a domain-expert teacher of their own (e.g. in [10]).

3) Analytics and Data Mining: Intelligent systems not only benefit their students in real time; they also give researchers the tools they need to better understand how students learn and struggle in these systems. This is accomplished by keeping detailed logs of students' interactions with the system, and then analyzing student outcomes, both in hypothesis-driven experiments and exploratory data mining. Good logging and analysis does not require an intelligent system, and increasingly it is a prerequisite for developing intelligent features, as data-driven techniques become more ubiquitous. For textual environments, numerous analyses have lent insight into students' programming abilities, strategies and difficulties, and predicted course performance and dropout [28]. Blikstein argues that these techniques can also be effective for open-ended and unscripted programming tasks [29] that are common in block-based environments. Some data mining analyses have targeted block-based programming, such Rodriguez and Boyer's analysis of pair programmers in Snap*!* [30]. However, many of the most popular programming environments currently keep no logs, and while there are many accessible databases of textual program logs [28], only a few block-based datasets ([19], [20]) have been shared.

III. INTELLIGENT SUPPORT FOR BLOCK PROGRAMMING

Some progress has already been made to integrate intelligent support into block-based programming environments. iSnap [19] is an extension of Snap*!* [9] that adds on-demand, next-step hints and feedback. If a student is stuck, they can ask iSnap to check their work, and it compares their partial solution to a database of other students' correct solutions. It matches them to the solution they have made the most progress towards and uses this to suggest edits that will bring the student closer to a correct solution. These suggestions are displayed as highlights on the student's code that indicate where a block should be deleted, moved or inserted.

Working with the Alice programming environment, Cooper et al. added a series of guided tutorials, which relied on "buggy" worked examples to preemptively correct or prevent common mistakes and misconceptions. However, while the authors call the system an ITS, their work focused more on creating tutorial content "to support the ITS," rather than its intelligent, adaptive features [31]. Diana et al. created a real-time dashboard to support Alice teachers in detecting struggling and idling students [32]. The dashboard used a predictive model to estimate students' learning outcomes using historical data relating Alice programming log data to a subsequent assessment. Using the same dataset, Grover et al. [33] developed a framework for identifying evidence of computational thinking from log data, based on an Evidence Centered Design methodology. This initial work is promising, but there is still much to be done to establish a research community around intelligent support for block-based programming.

IV. CONCLUSION

In this paper we have argued that intelligent support for programming is an empirically-tested way to improve learning outcomes and that integrating this support into block-based programming environments is a promising research direction that should have a clear positive impact on students. We close by touching on some limitations to our claims and by offering suggestions of ways that researchers and developers of block-based programming environments can contribute to the development of intelligent features for their systems.

A. Limitations and Challenges

We acknowledge that intelligent support has limitations and drawbacks, and that it may not be appropriate in all situations. Help features may be most effective for certain students (e.g. medium-skill level students) and in certain contexts [34]. Promising results from lab experiments may not always scale to real classrooms. Students can abuse help features and "game the system," which can negatively impact learning [34]. Some of the most promising evaluations of ITSs for programming are also quite old (e.g. [4]) and should be replicated in newer systems. In sum, after years of research, there are still many open questions on effective intelligent support. However, we see these as arguments for further research to ensure that intelligent features are employed effectively. It is our hope that these discoveries can also target learners using block-based programming.

We also acknowledge that most intelligent support was designed for well-structured courses, and blocks are commonly used in exploratory, creative and project-based learning. While we recognize that adapting these features to open-ended domains is challenging, it is possible and has been done for open-ended learning environments [35]. It can also be very challenging to collect education data ethically and legally, especially from children [36]. There are also barriers to adoption, especially in K12 classrooms, where teachers will have reasonable doubts about the utility of AI systems and automated approaches. However, previous work shows that ITSs can be made to work for schools by integrating knowledge from content, curricular, and classroom experts with that of AI and cognitive psychology experts [37].

B. Call to Action

As noted above, we will need the expertise of several kinds of experts to build successful, widely-adopted, block-based programming environments with intelligent support. To accomplish this, the designers and researchers of block-based systems must be equal partners with AI and cognitive science researchers. We close with suggestions for how to contribute to and partner in this research: 1) Add comprehensive logging to an existing block-based environment, and where possible work to disseminate the data, allowing for analysis and better evaluation. 2) Innovate and brainstorm on the *design and implementation* of intelligent support for block-based programming, addressing questions of usability and interface. 3) Attend a conference on intelligent support such as Artificial Intelligence in Education (AIED), Educational Data Mining (EDM), Learning at Scale (L@S) or User Modeling, Adaptation and Personalization (UMAP), and start a discussion. 4) Critically evaluate an existing system (e.g. those in Section III), experimentally or qualitatively, and share the results.

REFERENCES

[1] T. W. Price and T. Barnes, "Comparing Textual and Block Interfaces in a Novice Programming Environment," in *Proc. of Int. Computing Education Research Conf.*, 2015.

[2] D. Weintrop and U. Wilensky, "To Block or not to Block, That is the Question: Students' Perceptions of Blocks-based Programming," in *Proc. of Interaction Design and Children*, 2015, pp. 199–208.

[3] K. Vanlehn, "The Behavior of Tutoring Systems," *Int. J. of Artificial Intelligence in Education*, vol. 16, no. 3, pp. 227–265, 2006.

[4] A. T. Corbett, "Cognitive Computer Tutors: Solving the Two-Sigma Problem," in *Proc. of User Modeling*, 2001, pp. 137–147.

[5] A. Corbett and J. R. Anderson, "Locus of Feedback Control in Computer-Based Tutoring: Impact on Learning Rate, Achievement and Attitudes," in *Proc. of ACM SIGCHI*, 2001, pp. 245–252.

[6] J. Holland, A. Mitrovic, and B. Martin, "J-LATTE: a Constraint-based Tutor for Java," in *Proc. of Computers in Education*, 2009, pp. 142–146.

[7] K. Rivers and K. R. Koedinger, "Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor," *Int. J. of Artificial Intelligence in Education*, vol. 27, no. 1, pp. 37–64, 2017.

[8] A. Gerdes, B. Heeren, J. Jeuring, and L. T. van Binsbergen, "Ask-Elle: an Adaptable Programming Tutor for Haskell Giving Automated Feedback," *Int. J. of Artificial Intelligence in Education*, vol. 27, no. 1, pp. 1–36, 2016.

[9] D. Garcia, B. Harvey, and T. Barnes, "The Beauty and Joy of Computing," *ACM Inroads*, vol. 6, no. 4, pp. 71–79, 2015.

[10] J. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk, "Programming by choice: urban youth learning programming with scratch," *ACM SIGCSE Bulletin*, vol. 40, no. 1, pp. 367–371, 2008.

[11] J. Cuny, D. A. Baxter, D. D. Garcia, S. Hall, J. Gray, R. Morelli, and D. Baxter, "CS Principles Professional Development: Only 9,500 to go!" in *Proc. of ACM SIGCSE*, 2014, pp. 543–544.

[12] D. D. Garcia, S. Hall, J. Campbell, J. Denero, S. Hall, M. L. Dorf, A. Arbor, S. Reges, and J. Campbell, "CS10K Teachers by 2017? Try CS1K+ students NOW! Coping with the Largest CS1 Courses in History," in *Proc. of ACM SIGCSE*, 2016, pp. 396–397.

[13] A. T. Corbett, "Cognitive Mastery Learning in the ACT Programming Tutor," Tech. Rep., 2000.

[14] P. Brusilovsky, S. Somyurek, J. Guerra, R. Hosseini, V. Zadorozhny, and P. J. Durlach, "Open Social Student Modeling for Personalized Learning," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 3, pp. 450–461, 2016.

[15] K. Rivers, E. Harpstead, and K. Koedinger, "Learning Curve Analysis for Programming: Which Concepts do Students Struggle With?" in *Proc. of Int. Computing Education Research Conf.*, 2016, pp. 143–151.

[16] M. Yudelson, R. Hosseini, A. Vihavainen, and P. Brusilovsky, "Investigating Automated Student Modeling in a Java MOOC," in *Proc. of Educational Data Mining*, 2014, pp. 261–264.

[17] K. H. Koh, A. Basawapatna, H. Nickerson, and A. Repenning, "Real time assessment of computational thinking," in *Proc. IEEE Symp. on Visual Languages and Human-Centric Computing*, 2014, pp. 49–52.

[18] D. M. C. Lee, M. M. T. Rodrigo, R. S. J. d. Baker, J. O. Sugay, and A. Coronel, "Exploring the Relationship between Novice Programmer Confusion and Achievement," in *Int. Conf. on Affective Computing and Intelligent Interaction*, Berlin, Heidelberg, 2011, pp. 175–184.

[19] T. W. Price, Y. Dong, and D. Lipovac, "iSnap: Towards Intelligent Tutoring in Novice Programming Environments," in *Proc. of ACM SIGCSE*, 2017.

[20] C. Piech, M. Sahami, J. Huang, and L. Guibas, "Autonomously Generating Hints by Inferring Problem Solving Policies," in *Proc. of ACM Conf. on Learning @ Scale*, 2015, pp. 1–10.

[21] T. Price, R. Zhi, and T. Barnes, "Evaluation of a Data-driven Feedback Algorithm for Open-ended Programming," in *Proc. of Educational Data Mining*, 2017.

[22] A. C. Bart, J. Tibau, E. Tilevich, C. A. Shaffer, and D. Kafura, "BlockPy: An Open Access Data-Science Environment for Introductory Programmers," *Computer*, vol. 50, no. 5, pp. 18–26, 2017.

[23] A. Ezen-Can and K. E. Boyer, "A tutorial dialogue system for real-time evaluation of unsupervised dialogue act classifiers: Exploring system outcomes," in *Int. Conf. on Artificial Intelligence in Education*, 2015, pp. 105–114.

[24] J. F. Grafsgaard, J. B. Wiggins, K. E. Boyer, E. N. Wiebe, and J. C. Lester, "Predicting Learning and Affect from Multimodal Data Streams in Task-Oriented Tutorial Dialogue," in *Proc. of Educational Data Mining*, 2014, pp. 122–129.

[25] P. S. Buffum, K. E. Boyer, E. N. Wiebe, B. W. Mott, and J. C. Lester, "Mind the Gap: Improving Gender Equity in Game-based Learning Environments with Learning Companions," in *Int. Conf. on Artificial Intelligence in Education*. Springer, 2015, pp. 64–73.

[26] C. Piech, J. Huang, A. Nguyen, M. Phulsuksombati, M. Sahami, and L. Guibas, "Learning program embeddings to propagate feedback on student code," *Proc. of Machine Learning*, vol. 2, pp. 1093–1102, 2015.

[27] A. Head, E. Glassman, G. Soares, R. Suzuki, L. Figueredo, L. D'Antoni, and B. Hartmann, "Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis," in *Proc. of ACM Conf. on Learning @ Scale*, 2017, pp. 89–98.

[28] P. Ihantola, M. Butler, S. H. Edwards, V. Tech, A. Korhonen, A. Petersen, K. Rivers, M. Á. Rubio, J. Sheard, J. Spacco, C. Szabo, and D. Toll, "Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies," in *Proc. of ACM ITiCSE Conf.*, 2015.

[29] P. Blikstein, "Using learning analytics to assess students' behavior in open-ended programming tasks," in *Proc. of Learning Analytics and Knowledge*, 2011, pp. 110–116.

[30] F. J. Rodríguez and K. E. Boyer, "Discovering Individual and Collaborative Problem-Solving Modes with Hidden Markov Models," in *Proc. of Artificial Intelligence in Education*, 2015, pp. 408–418.

[31] S. Cooper, Y. J. Nam, and L. Si, "Initial Results of Using an Intelligent Tutoring System with Alice," in *Proc. of ACM ITiCSE Conf.*, 2012, pp. 138–143.

[32] N. Diana, M. Eagle, J. Stamper, S. Grover, and M. Bienkowski, "An Instructor Dashboard for Real-Time Analytics in Interactive Programming Assignments," in *Learning Analytics*, 2017.

[33] S. Grover, M. Bienkowski, S. Basu, M. Eagle, N. Diana, and J. Stamper, "A framework for hypothesis-driven approaches to support data-driven learning analytics in measuring computational thinking in block-based programming," in *Proc. of Learning Analytics & Knowledge*, 2017, pp. 530–531.

[34] V. Aleven, I. Roll, B. M. McLaren, and K. R. Koedinger, "Help Helps, But Only So Much: Research on Help Seeking with Intelligent Tutoring Systems," *Int. J. of Artificial Intelligence in Education*, vol. 26, no. 1, pp. 1–19, 2016.

[35] S. Karkalas and S. Gutierrez-Santos, "Enhanced JavaScript Learning Using Code Quality Tools and a Rule-Based System in the FLIP Exploratory Learning Environment," in *IEEE Int. Conf. on Advanced Learning Technologies*, jul 2014, pp. 84–88.

[36] J. Sabourin, L. Kosturko, C. Fitzgerald, and S. Mcquiggan, "Student Privacy and Educational Data Mining : Perspectives from Industry," in *Proc. Int. Conf. on Educational Data Mining*, 2015, pp. 164–170.

[37] K. R. Koedinger and J. R. Anderson, "Intelligent tutoring goes to school in the big city," *Int. J. of Artificial Intelligence in Education*, vol. 8, pp. 30–43, 1997.