Engineering Educators Bringing the World Together
2025 ASEE Annual Conference & Exposition
Palais des congrès de Montréal, Montréal, QC • June 22–25, 2025 ASEE
Paper ID #48221

BOARD # 201: Development of a Programming Environment to Bridge Students from Block-Based to Text-Based Programming (Work in Progress)

Elliot Benjamin Roe, Georgia Institute of Technology
Duncan Johnson, Tufts Center for Engineering Education and Outreach

Duncan Johnson is an undergraduate student at Tufts University majoring in Computer Science. He is the co-founder and Executive Director of BX Coding, a STEM education nonprofit dedicated to building open-source software and curriculum for STEM educators. Through his work with BX Coding and the Tufts Center for Engineering Education and Outreach (CEEO), his research has focused on educational methodologies and technologies that introduce elementary and middle school students to Computer Science. Currently, he's exploring how technologies can leverage generative AI to better support educators.

Dr. Ethan E Danahy, Tufts University

Dr. Ethan Danahy is a Research Associate Professor at the Center for Engineering Education and Outreach (CEEO) with secondary appointment in the Department of Computer Science within the School of Engineering at Tufts University. Having received his graduate degrees in Computer Science and Electrical Engineering from Tufts University, he continues research in the design, implementation, and evaluation of different educational technologies. With particular attention to engaging students in the STEAM content areas, he focuses his investigations on enhancing creativity and innovation, supporting better documentation, and encouraging collaborative learning.

Development of a Programming Environment to Bridge Students from Block-Based to Text-Based Programming (Work in Progress)

Introduction

Computer Science (CS) education continues to expand in U.S. schools, with high school CS offerings increasing from 35% to 60% between 2017 and 2024 [1]. However, significant demographic disparities persist, particularly affecting African American/Black, Hispanic/Latino/Latina/Latinx, and Native American/Alaskan students. These disparities are notably less pronounced in middle and elementary school contexts [1], suggesting that early intervention is crucial for building an inclusive CS pipeline. Teaching programming to young learners presents unique challenges, as elementary and middle school students are still developing reading comprehension, typing skills, and abstract thinking abilities. They need programming environments that are accessible and engaging while teaching fundamental computational concepts.

Block-based programming has emerged as a popular method for introducing young learners to CS fundamentals. By providing a visual, intuitive interface that removes syntactic barriers, block-based environments allow learners to focus on developing computational thinking skills rather than struggling with syntax and typing [2]. The success of this approach is exemplified by Scratch, a block-based programming platform that has reached over 130 million children worldwide since 2007 [3]. Scratch's design principles of "low bar, high ceiling, and wide walls" were intentional choices to nurture creative and computational thinking on the platform [4], [5].

Although block-based programming provides an excellent entry point to CS, the transition to text-based programming is a crucial step in a learner's CS education journey. Text-based languages remain the standard for professional software development [6], advanced placement courses like AP CS A [7], and college-level computer science curricula [8]. As CS education initiatives target increasingly younger audiences, facilitating an early and smooth transition from blocks to text becomes particularly important.

While Scratch excels at nurturing computational thinking and creative skills, it was not designed to facilitate the transition to text-based programming. The relationship between block-based and text-based environments remains an active area of research [9], [10], [11], [12], [13], with Kölling et al. identifying specific barriers in transitioning between the two [14].

Key Transition Challenges

Building on Kölling et al.'s framework for analyzing block-to-text transitions, we examine several key challenges specifically in the context of moving from Scratch to text-based programming.

First, while Scratch provides an easily browsable list of available block primitives, text-based environments typically require memorization of commands. Second, Scratch offers rich input/output capabilities through block primitives that allow the creation of engaging interactive projects, whereas comparable functionality in text-based environments often requires significantly more complex implementation. Finally, Scratch's "tinkerable" environment enables playful exploration and immediate feedback, with features like live programming and resilient error handling that allow learners to modify executing programs without interruption [15], [16]. These characteristics of block-based programming are often absent in text-based environments, leading to potential losses in student agency and creativity during the transition process.

Our Solution: Patch

We present Patch[1], a free and open-source online coding environment built to help novice learners bridge the gap between Scratch and Python. Figure 1 shows Patch's editor. Built on the Scratch VM [17], Patch integrates Pyodide [18], a library that enables web-based Python execution, to allow learners to write Python code that directly interacts with the Scratch game engine. Patch is built to mirror many of the successful aspects of Scratch's programming environment that aren't seen in a traditional text-based programming environment. We describe below how Patch addresses the key transition challenges:

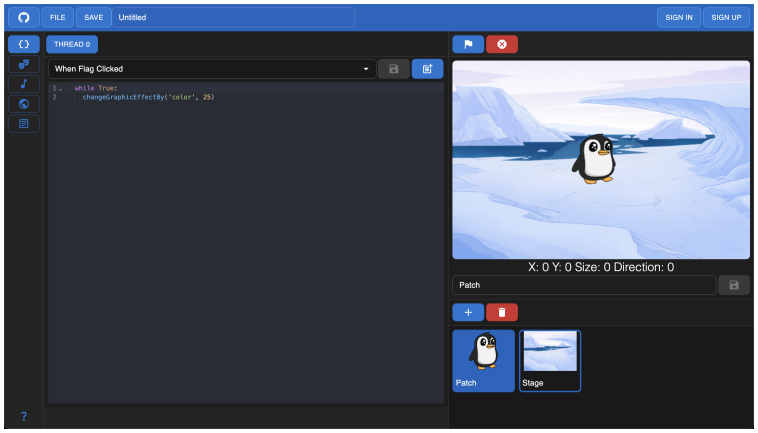
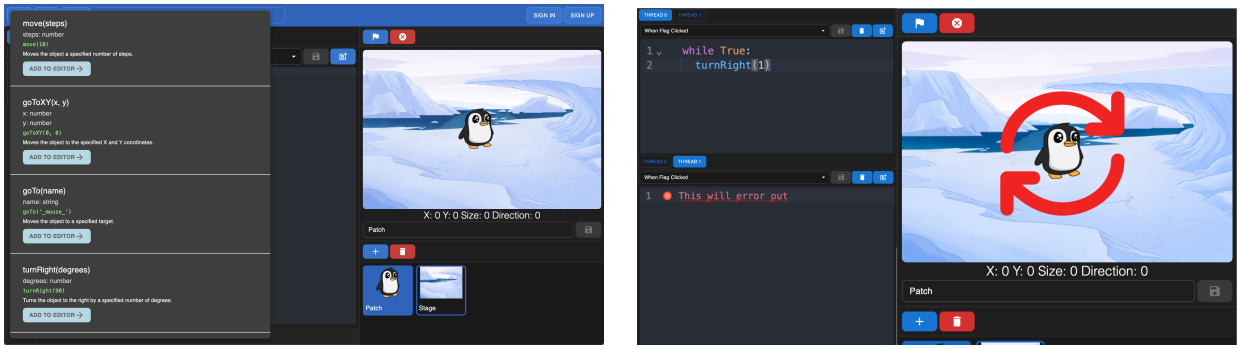


Figure 1. The editor of the Patch coding environment.

How Patch Addresses Transition Challenges

In addressing the key transition challenges identified above, Patch implements specific design choices. To address readability and command recognition, Patch uses direct semantic mappings between Scratch blocks and corresponding Python functions, preserving Scratch's intuitive naming conventions while introducing proper Python syntax. To support command discovery, as shown in Figure 2 (a), we created a command palette displaying all available Patch functions.

[1]Available at: https://codepatch.org



(a) The pop-up command palette menu that allows students to scroll through the available commands

(b) Multi-thread Patch program where Thread 0 would still run despite the compilation error in Thread 1

Figure 2. Patch interface features demonstrating command discovery and error handling

Like Scratch's block palette, this allows learners to explore Patch's functions through recognition rather than recall, with an "Add to Editor" button for easy code insertion.

For handling inputs and outputs, we leverage the same engine as Scratch, maintaining compatibility with many of Scratch's capabilities and preserving much of the functionality during the transition. Regarding tinkerability, while Patch does not mirror all of Scratch's live programming capabilities, it notably allows separate threads to execute independently and in parallel. This means thread programs can execute even while others are failing, as shown in Figure 2 (b). In this example, Thread 0 continues to make the Patch penguin spin despite the compilation error in Thread 1.

We designed Patch to balance familiarity and advancement, helping learners maintain creative agency during their transition to text-based programming. Through this design philosophy, we hope that students can create interactive stories and games similar to their Scratch projects while learning Python fundamentals. We chose a web-based implementation with visual feedback to reduce technical barriers while supporting an experimental, iterative learning style. To explore whether these design choices effectively support the block-to-text transition, we present a preliminary qualitative study investigating Patch's classroom implementation and provide recommendations for future development.

Methodology

An earlier version of this research, focused on curriculum development, was presented as an internal report through the Laidlaw Scholars program [19]. This paper presents new analysis focused specifically on how Patch functions as a transitional programming tool based on instructor observations and student behaviors. The preliminary qualitative study investigated Patch's effectiveness as a transitional programming tool through two week-long educational programs during Summer 2023.

We conducted two summer programs: a half-day workshop with 11 students (ages 11-14) in June and a full-day workshop with 15 students (grades 6-9) in July. Data collection included instructor observations and daily debrief discussions, student work artifacts from Scratch, Patch, and Python projects, and entrance surveys assessing prior programming experience. Three instructors documented student engagement, knowledge transfer between programming environments, and project development throughout both workshops.

Key limitations include instructor bias (two were involved in Patch's development), the intensive week-long format versus typical distributed instruction, small sample size (26 students), and potential selection bias from the paid summer program format. These factors limit the generalizability of our findings.

Our analysis focused on identifying instances where Patch supported the block-to-text transition through a review of instructor notes and student projects. These preliminary observations were validated through instructor discussion, but warrant more rigorous future study.

Results & Discussion

During our pilot implementation of Patch across two summer workshops with 26 total middle school students, we observed instances of the platform's potential as a transitional programming tool, along with important limitations that inform future development:

Knowledge Transfer and Independent Work

Students demonstrated an ability to transfer knowledge from Scratch to Patch even with minimal instruction. In a notable example during our first workshop, after just 90 minutes of exposure to Patch, a student successfully implemented character cloning functionality without specific instruction by applying their prior understanding of Scratch's cloning blocks. Instructors observed that students generally worked more independently in Patch compared to Python, specifically noting the silence during the introductory exploration of Patch, as they perceived the students to be intensely focused on exploring with Patch. Overall, instructors found it easier to motivate students to engage with Python programming via the rich set of activities that could be brought over from Scratch into the Patch programming environment.

Technical and Motivational Challenges

While the basic functionality supported student learning, we identified several key challenges. Performance issues emerged when students created programs with multiple threads, and typing skills posed a barrier for some middle school students, making rapid iteration more difficult than in Scratch. Perhaps most significantly, the interoperability between Scratch and Patch was a double-edged sword. While it allowed learners to more easily begin programming in Python,

because the output of Patch is very similar to Scratch, it meant that students had little inherent motivation to transition to typed code. Students with limited typing skills found they could create games more quickly using Scratch's block-based interface than by typing code in Patch, making it difficult for instructors to encourage continued use of the platform.

These preliminary findings suggest that while Patch's scaffolded approach enables knowledge transfer between block-based and text-based programming, additional development is needed to address technical limitations and provide clearer motivation for students to progress beyond block-based programming. Future iterations of the platform should focus on performance optimization, typing support, and introducing features that demonstrate the enhanced capabilities available through text-based programming.

Future Work

As this work is preliminary, we have identified several key directions for future research and development of Patch as a transitional programming tool. Our immediate research priorities include conducting controlled studies comparing Patch to traditional Python introduction methods to quantify its effectiveness in supporting the block-to-text transition. These studies will require developing specific assessment tools to measure knowledge transfer between programming environments and should include diverse student populations across different classroom settings.

Platform development will focus on two critical areas: improving error message support for young learners and implementing social features similar to those in Scratch—such as project sharing and remixing—to enable peer learning and creative collaboration. Through these combined efforts, we hope to significantly improve a learner's first experience with text-based programming. While Scratch has been highly successful in cultivating engagement in creative and computational thinking skills among young learners, there is no such text-based environment that allows exploration in a similar manner for younger ages. With Patch, we hope to make the first step after Scratch in a CS learner's journey more accessible, engaging, and creative.

Acknowledgments

This work would not have been possible without the many amazing and dedicated people at BX Coding and everyone that helped us along the way. Thank you to Ava Wandersleben, Leighton Miguel, and Benjamin Montgomery for their hard work on Patch. We also thank the Laidlaw Scholars program at Tufts University for supporting the initial phase of this research.

References

[1] Code.org, "2024 state of computer science education," Code.org, CSTA, ECEP Alliance, 2024. [Online]. Available: https://advocacy.code.org/stateofcs.

[2] D. Weintrop, "Block-based programming in computer science education," *Commun. ACM*, vol. 62, no. 8, pp. 22–25, Jul. 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3341221 (visited on Jan. 8, 2025).

[3] *Our story*, Publication Title: Scratch Foundation. [Online]. Available: https://www.scratchfoundation.org/our-story (visited on Jan. 8, 2025).

[4] M. Resnick, B. Myers, K. Nakakoji, B. Shneiderman, R. Pausch, T. Selker, and M. Eisenberg, "Design principles for tools to support creative thinking," MIT Media Lab, Carnegie Mellon Univ., Univ. of Tokyo, Univ. of Maryland, Univ. of Colorado, Oct. 2005, p. 15.

[5] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009. [Online]. Available: https://dl.acm.org/doi/10.1145/1592761.1592779 (visited on Jan. 6, 2025).

[6] *TIOBE index*, Publication Title: TIOBE. [Online]. Available: https://www.tiobe.com/tiobe-index/ (visited on Jan. 15, 2025).

[7] "AP computer science a course and exam description, effective fall 2019," CollegeBoard. [Online]. Available: https://apstudents.collegeboard.org/sites/default/files/2019-05/ap-computer-science-a-course-and-exam-description.pdf.

[8] Cc2020 Task Force, *Computing Curricula 2020: Paradigms for Global Computing Education*. New York, NY, USA: ACM, Nov. 2020. [Online]. Available: https://dl.acm.org/doi/book/10.1145/3467967 (visited on Jan. 15, 2025).

[9] 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*, ser. IDC '15, New York, NY, USA: Association for Computing Machinery, Jun. 2015, pp. 199–208. [Online]. Available: https://doi.org/10.1145/2771839.2771860 (visited on Jan. 6, 2025).

[10] C. M. Lewis, "How programming environment shapes perception, learning and goals: Logo vs. scratch," in *Proceedings of the 41st ACM technical symposium on Computer science education*, ser. SIGCSE '10, New York, NY, USA: Association for Computing Machinery, Mar. 2010, pp. 346–350. [Online]. Available: https://doi.org/10.1145/1734263.1734383 (visited on Jan. 6, 2025).

[11] S.-M. Liao, "SCRATCH to r: Toward an inclusive pedagogy in teaching coding," *Journal of Statistics and Data Science Education*, vol. 31, no. 1, pp. 45–56, Jan. 2023. [Online]. Available: https://doi.org/10.1080/26939169.2022.2090467 (visited on Jan. 6, 2025).

[12] Y. Lin, "Switch mode: Exploring authoring python inside a block-based programming environment," in *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct. 2023, pp. 312–313. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10305663?casa_token=QSI2CUUq8fQAAAAA:H56Mk2DMlk9DkTIPlcLaapY2UzUVY8X4YlDTpeZ_lp-TCJg8hYrrgyZAQ_wBGjiNY45IywQ (visited on Jan. 6, 2025).

[13] M. J. Johnson, R. Baker-Ramos, C. L. Hovey, and B. DiSalvo, "Keeping mindful of modality: A comparison of computer science education resources for learning," in *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, Koli Finland: ACM, Nov. 2023, pp. 1–14. [Online]. Available: https://dl.acm.org/doi/10.1145/3631802.3631819 (visited on Jan. 10, 2025).

[14] M. Kölling, N. C. C. Brown, and A. Altadmri, "Frame-based editing: Easing the transition from blocks to text-based programming," in *Proceedings of the Workshop in Primary and Secondary Computing Education*, ser. WiPSCE '15, New York, NY, USA: Association for Computing Machinery, Nov. 2015, pp. 29–38. [Online]. Available: https://doi.org/10.1145/2818314.2818331 (visited on Jan. 6, 2025).

[15] M. Resnick and E. Rosenbaum, "Designing for tinkerability," in *Design, Make, Play*, Routledge, 2013.

[16] S. L. Tanimoto, "A perspective on the evolution of live programming," in *2013 1st International Workshop on Live Programming (LIVE)*, May 2013, pp. 31–34. [Online]. Available: https://doi.org/10.1109/LIVE.2013.6617346 (visited on Jan. 9, 2025).

[17] *Scratch VM*. [Online]. Available: https://github.com/scratchfoundation/scratch-vm.

[18] *Pyodide*. [Online]. Available: https://pyodide.org.

[19] Duncan Johnson, "PATCH-ED creating curricula to effectively transition young computer science students from block-based to text-based programming," Tufts University (Laidlaw Scholars Program), Internal Program Report, Sep. 2023.