BitBlocks: A Fully Offline Block-Coding Environment on a Microcontroller-Class Device

Anonymous Author 1
anonymous1@example.edu
Anonymous Institution
Anonymous City, Anonymous Country

Anonymous Author 2
anonymous2@example.edu
Anonymous Institution
Anonymous City, Anonymous Country

Abstract

Block-based programming has become an essential entry point into computer science education by replacing fragile textual syntax with visible, composable program structures. Yet commonly used environments still assume access to a personal computer or tablet, while their sharing and collaboration features often assume reliable connectivity. We present BitBlocks, a fully offline, touch-based block-coding environment intended for novice K–12 learners in low-resource or infrastructure-constrained classrooms. Authoring and execution occur on an ESP32-S3-WROOM-1-N16R8 microcontroller with a 480×320 display. Learners browse categorized blocks, drag and snap them into connected stacks, select sprites and backdrops, and execute programs without a browser, account, network connection, or separate programming host. The educational goal is to make the complete author–run–revise cycle available on an inexpensive, individually programmable device rather than limiting participation to scheduled access to a computer laboratory. BitBlocks combines a UI-independent program model, flash-resident media, touchscreen-oriented events, and a lightweight interpreter. The current prototype supports core workspace interactions, local media selection, sprite movement, and early conditional logic.

ACM Reference Format:
Anonymous Author 1 and Anonymous Author 2. 2026. BitBlocks: A Fully Offline Block-Coding Environment on a Microcontroller-Class Device. In *The 2nd ACM Virtual Global Computing Education Conference (SIGCSE Virtual 2026), November 12–15, 2026, Virtual Event*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/XXXXXXX.XXXXXXX

1 Introduction

Block interfaces make program structure visible and avoid many syntax errors; empirical studies report faster task completion and, in high-school settings, larger learning gains than comparable text interfaces [1, 2]. Scratch combines snap-together commands with programmable media for creating animations, stories, and games [3, 4]. BitBlocks preserves this familiar experience: students drag blocks from a palette, connect them into executable stacks, and run the resulting program.

Access also concerns whether each learner can repeatedly author, test, debug, and retain projects. BitBlocks is intended for novice K–12 learners in low-resource or infrastructure-constrained classrooms, where giving each learner an individually programmable device may be more practical than maintaining a computer laboratory. The intended classroom model shifts participation from taking turns at a shared computer toward each learner directly creating and debugging on a personal or classroom-issued device.

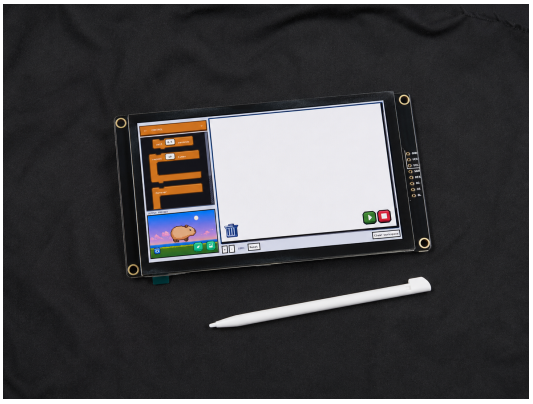

Figure 1: BitBlocks running on the target touchscreen without a separate programming computer.

BitBlocks places authoring, execution, and revision on one offline-first device. The current prototype has an estimated core hardware cost of approximately US$20, excluding the optional SD card. We ask how a low-cost standalone device can support creative introductory programming on a four-inch display. Figures 1 and 2 show the hardware and editor.

2 Related Work and Educational Position

Scratch and Monroy-Hernández emphasize low-floor creative expression, visible program structures, and programmable media [3, 4]. BitBlocks adopts sprites, backdrops, events, movement, and immediate execution so novices can connect code to visible outcomes and create personally meaningful projects, while treating networking as optional.

The GoGo Board reduced barriers to physical computing in economically challenged communities [5], and Blikstein argued for expressive tools grounded in learners' local contexts [6]. BitBlocks extends this access argument to authoring: learners should not require a second, more expensive computer to program an educational device. It is positioned as an access-oriented prototype for computer science education, not simply a microcontroller demonstration.

TileCode and MicroCode show that programming can occur directly on constrained handheld hardware [7, 8]. BitBlocks differs

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGCSE Virtual 2026, Virtual Event
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/26/11
https://doi.org/10.1145/XXXXXXX.XXXXXXX

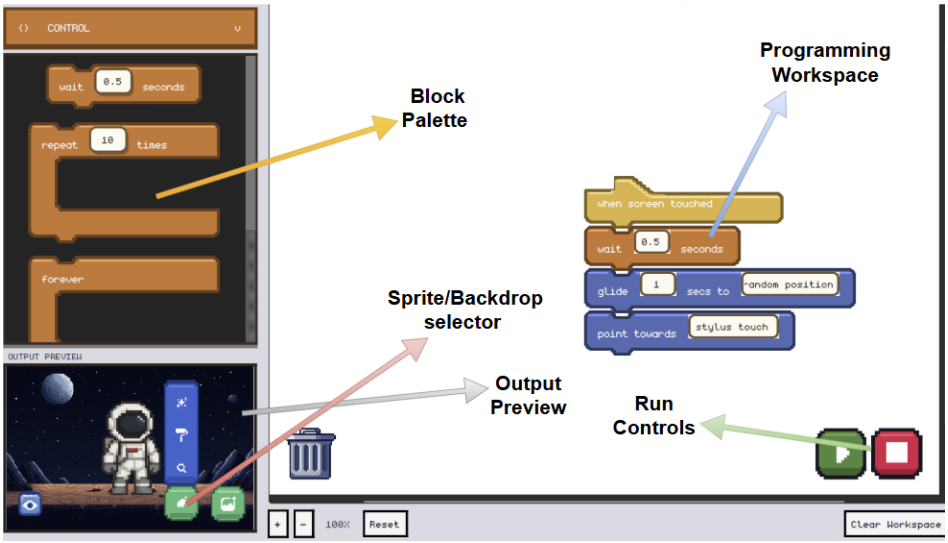

Figure 2: BitBlocks editor regions and the on-device edit–run workflow.

through its freely arranged, touch-operated stack workspace, integrated programmable media, and focus on an individually usable edit–run–debug cycle.

3 Learning Design and Implementation

The interaction design follows familiar block-programming conventions. Control is orange, Movement is blue, Events are yellow, and Operators are green. Learners drag blocks from a scrollable palette, snap compatible connectors together, and move connected stacks as units. This interaction design is intended to reduce syntax demands while preserving sequencing, events, parameters, conditionals, and visible cause-and-effect relationships central to introductory programming.

Desktop interactions are adapted to a stylus and touchscreen: *point towards mouse-pointer* becomes *point towards stylus touch*, and event blocks include *when display touched*. This vocabulary also creates a path toward sensor- and hardware-triggered conditionals. Trash, clear, zoom, and reset controls support revision and debugging. Pressing Play temporarily replaces the editor with a full-resolution output view, providing a larger view of program behavior before learners return to revise their blocks.

The embedded system enables this experience without a host computer. BitBlocks runs on an ESP32-S3-WROOM-1-N16R8 with a 480×320 touchscreen. Its 16 MB of flash and 8 MB of PSRAM support local sprites, backdrops, and program data; SD-card expansion is planned as the libraries grow. C++ firmware uses Arduino/PlatformIO and LVGL 9.5. A UI-independent model stores connected blocks, and a lightweight interpreter executes them. A Python/PySide6 prototype validates interactions before embedded implementation.

4 Preliminary Design Observations

The prototype supports a complete local author–run–revise loop without accounts, connectivity, or a separate computer after firmware installation. This makes one-device-per-learner activities technically plausible, although classroom feasibility and learning effects remain to be evaluated.

Internal testing identified display density as the main constraint. Larger stacks exceed the visible workspace, while zooming out reduces text and touch-target legibility. For novices, this may complicate tracing, debugging, and maintaining an overview. These are design hypotheses, not learning-outcome evidence; they motivate the planned pages feature and a study of learner authoring, navigation, and error recovery.

5 Future Work

We plan to expand the opcode library toward the breadth expected from mature block environments, enabling more sophisticated animations, games, and interactive stories. As projects grow, a planned *pages* feature will partition a project across multiple workspaces or scenes while preserving shared sprites, variables, and project state. This organization is intended to reduce the need to view an entire large program on a single four-inch canvas.

A second direction is hardware-dependent blocks inspired by educational robotics kits. Sensor, actuator, and device-event blocks would let students connect physical electronics to their programs: a light, distance, or touch sensor could trigger a conditional branch, alter a sprite, or control an attached motor, LED, or other output. This would allow the same environment to bridge on-screen programmable media and real-world experimentation.

The next evaluation phase will involve novice learners completing short authoring and debugging tasks. We will examine task completion, incorrect placements, snapping and deletion recoveries, assistance requests, and how page organization changes navigation in larger projects. These studies will determine whether BitBlocks is merely technically compact or genuinely usable as an offline-first programming environment.

References

[1] Thomas W. Price and Tiffany Barnes. 2015. Comparing Textual and Block Interfaces in a Novice Programming Environment. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. Association for Computing Machinery, 91–99. https://doi.org/10.1145/2787622.2787712

[2] David Weintrop and Uri Wilensky. 2017. Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education* 18, 1, Article 3 (2017), 1–25. https://doi.org/10.1145/3089799

[3] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Communications of the ACM* 52, 11 (2009), 60–67. https://doi.org/10.1145/1592761.1592779

[4] Andrés Monroy-Hernández and Mitchel Resnick. 2008. Empowering Kids to Create and Share Programmable Media. *Interactions* 15, 2 (2008), 50–53. https://doi.org/10.1145/1340961.1340974

[5] Arnan Sipitakiat, Paulo Blikstein, and David P. Cavallo. 2004. GoGo Board: Augmenting Programmable Bricks for Economically Challenged Audiences. In *Proceedings of the 6th International Conference of the Learning Sciences*. International Society of the Learning Sciences. https://doi.org/10.5555/1149126.1149185

[6] Paulo Blikstein. 2008. Travels in Troy with Freire: Technology as an Agent of Emancipation. In Pedro Noguera and Carlos Alberto Torres (Eds.), *Social Justice Education for Teachers: Paulo Freire and the Possible Dream*. Sense Publishers, Rotterdam, The Netherlands, 205–244. https://doi.org/10.1163/9789460911446_015

[7] Thomas Ball, Shannon Kao, Richard Knoll, and Daryl Zuniga. 2020. TileCode: Creation of Video Games on Gaming Handhelds. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, 1217–1230. https://doi.org/10.1145/3379337.3415839

[8] Kobi Hartley, Elisa Rubegni, Lorraine Underwood, Joe Finney, Thomas Ball, Steve Hodges, Eric Anderson, Peli de Halleux, James Devine, and Michal Moskal. 2024. Meet MicroCode: A Live and Portable Programming Tool for the BBC micro:bit. In *Proceedings of the 23rd Annual ACM Interaction Design and Children Conference*. Association for Computing Machinery, 355–370. https://doi.org/10.1145/3628516.3656995