RCKid: A personal handheld for creative STEM beyond the classroom

Petr Maj
Faculty of Informatics
Czech Technical University
Prague, Czech Republic
peta.maj82@gmail.com

RCKid is an open-source handheld console designed for young creators. It is built to be the first piece of technology that feels truly personal to a child — not a classroom kit, but a daily companion that encourages creativity, ownership, and sharing. RCKid grows with the child: pre-literate users can customize existing applications through asset editors, then progress through visual behavior editor, block-based programming and textual coding. Advanced learners can unlock the full capabilities of the device through an SDK. The platform is extensible through cartridges that expand both hardware and software functionality, making children's creations tangible, portable, and easy to share with peers.

I. Introduction

Decades of creative learning research, beginning with Papert's *Mindstorms* [1], highlight the importance of children creating personally meaningful artifacts as a pathway to deep understanding. Today's STEM toolkits attempt to support creative expression, but they are rarely personal: they live in classrooms, often require extra equipment kids do not have access to on their own, and offer limited functionality compared to the rich, always-available digital devices children use in their daily lives. Even when these kits succeed pedagogically, they struggle to compete with the convenience, polish, and emotional resonance of consumer technology.

This paper introduces *RCKid* [1] (Figure 1), an open-source handheld console designed to help kids of wide age groups learn STEM skills by allowing them to make and share truly personally meaningful artifacts, thus increasing their engagement and learning. It does so via three main design principles:

A. Beyond classroom usability

RCKid is designed to look and feel like a real product and offers functionality beyond the classroom. Kids are more likely to take it with them after school, customize and make it truly personal. This leads to more interaction in general, which will naturally lead to more interaction with educational value, such as improving their creations from school. The device design incorporates the difference of the at-home environment compared to the classroom (no adult supervision, no extra equipment, such as computers, debuggers, etc.).

Figure 1 : Device prototype (front & back with inserted cartridge) showing the home screen.

B. Growing with the child

RCKid is designed to grow with its users, supporting them from very young pre-literate ages (5+) all the way to adulthood with an interaction ladder that naturally encourages kids to progress further and to build upon their existing skills and crucially their creations as well. A project that a five year old starts as simple asset modification that teaches them basic problem decomposition and game mechanics can be over the years extended into an ever more personalized game.

Furthermore, the more advanced interaction modes can enhance experience at the lower modes. A single advanced student progressing to the next level can create components in it that can be shared with its peers, providing gentle peer pressure for others to jump to the next level as well by being able to see the benefits first hand.

C. Tangibility & Modularity

Another defining feature of RCKid is the use of cartridges (shown in in Figure 2). Cartridges provide firmware for the

[1] http://rckid.org

device and can be swapped and copied. They provide tangibility to the creations as a kid can now create a game on their device, then give the physical cartridge to a friend, teacher, or parent to play. This provides a bridge for the abstract creations into the real world, making them feel more personal and meaningful and hence more engaging.

Figure 2 : RCKid Cartridge PCB in clear case. WiFi model with RaspberryPi RM2 module, 16MB flash and white LED. The cartridge is also ready for IR remote (unpopulated elements).

Cartridges also provide a way for extending the hardware capabilities of the base device, making it useful not only for software, but also embedded/hardware education, as well as allowing its use in novel contexts (walkie talkie, communicators, bench instruments, etc.).

II. Related work

RCKid integrates and builds upon previous work from four main areas: educational hardware platforms, physical devices for kids, creative programming environments, and on-device programming:

A. Educational Hardware Platforms

Arduino [11] is widely credited with bringing hardware into STEM education by providing development boards with integrated programmers expandable with the use of *shields*, standardized peripheral boards that plug into connectors on the main board. Raspberry Pi [12] represents similar concept, but instead of 8-bit microcontroller, it is based on a fairly powerful Linux system-on-chip. Micro:bit [8] sits in the middle performance-wise, but stands out for its educational focus and low barrier entry for classroom activities.

RCKid is not a development board, but a polished, personal device designed for long-term creative growth. It de-emphasizes hackability and hardware DIY ethos which may be overwhelming for young users while preserving the benefits: firmware flashing requires only USB cable and modularity is achieved via cartridges, which provide similar functionality to shields, but in a product-like packaging children can proudly show and share. This positions RCKid as a complementary category: a personal creative device rather than a general-purpose development platform.

B. Physical Devices

A variety of handheld, game-like devices aimed at hobbyists and makers exists across a wide range of form factors and performance tiers [9][10]. These devices often build upon existing hardware platforms, which allows them to reuse platform specific software and tools. As the original platforms were not designed with game consoles in mind, those handhelds lose some of the expandability of their original platforms. They typically target experienced users and focus on programming itself, often requiring familiarity with MicroPython or C++, firmware flashing, and external development tools.

RCKid is the device and the platform at the same time designed from the ground up with no inherited constraints. Its focus on education and beyond classroom usability has shaped every aspect of its design. From specs that are powerful enough for all-day usefulness, to details such as unique button shape design to aid navigation, RCKid has been designed with children as its primary users. Unlike a single purpose gaming device, RCKid aims to be personal device to the kids, almost like a cell phone, but safer, and oriented towards creation, not consumption.

C. Creative programming environments

Young learners benefit from visual programming environments that help them with program composition. Scratch [2] is a web-based programming environment that teaches algorithmic thinking through a block-based visual editor, while ScratchJr [3] extends the approach to younger children through simplified interaction and composition model.

Microsoft MakeCode [4] similarly uses block-based programming but introduces progression from blocks to TypeScript. MakeCode Arcade [5] augments the abstract web based runtimes with physical devices: the games created on MakeCode webpage can be flashed to a compatible handheld and played there.

RCKid integrates and extends this line of work by treating asset creation explicitly as a first step in computational thinking, enabling children to learn decomposition and game mechanics before encountering programming. It supports progression from assets to visual behaviors, blocks, and text-based code, combining the strengths of prior systems with an emphasis on tangibility. It goes a step further than MakeCode Arcade by allowing kids not only to enjoy their creations on a physical device, but as the games live in cartridges, allows physical sharing of the creations.

D. On-device programming

ScratchJr [3] recognized that programming environments should meet their users where they are and as part of its move towards younger children switched from computer and mouse based web environment to tablet application.

MakeCode Arcade devices face similar problem as the computers required for their programming are not as accessible to children as the devices themselves are, not mentioning the additional complexity a dual device setup requires. As the semantic gap between the handheld devices and computers is much wider, bridging it requires simplifications. TileCode [6] and MicroCode [7] both leverage MakeCode runtime to bring programming to physical devices in a tightly scoped manner: TileCode focuses on cellular automata inspired rule-based system for on device retro game programming, while MicroCode [7] emphasizes micro:bit control and pre-literate learners.

RCKid embraces the on-device programming model as a first class citizen: Its capabilities are high enough to support complex interactions and its runtime is explicitly designed to support and simplify on-device programming across multiple levels. The programming itself is not tied to specific domain and can always be extended with native classes to support new functionality without changing the interaction patterns. This multi-level on-device programming capability is formalized in RCKid's mastery ladder, described in the next section.

III. Mastery Ladder

A central contribution of RCKid is its five-tier mastery ladder, a progression of creative and programming experiences that grows with the child over multiple years. Each tier is a strict superset of the previous one: new capabilities extend, rather than replace, earlier ones, and the same game can evolve as the child advances. This continuity is deliberate: children do not "start over" when they learn something new - their creations grow with them.

This also reflects a pragmatic constraint: younger children often lack stable access to a computer, and the friction of dual-device workflows can interrupt short, opportunistic moments of creativity. For this reason, the lower tiers are designed to run entirely on the device, while the upper tiers introduce external tooling only when literacy, typing, and conceptual readiness make it appropriate. The continuity across levels also allows a project to exist in parts at multiple levels at the same time. This pushes the on-device programming experience even further as the initial development can be done on computer, followed by on-device tweaks.

The ladder design is guided by four principles:

1. Superset progression: each tier extends previous one in functionality
2. Downward extensibility: higher tiers not only unlock more options for game customization, but those can be encapsulated into components available at the lower tiers.
3. On-device first: whenever possible the tier should allow on device editing.
4. Project continuity: nothing is ever thrown away, as kids progress from tier to tier, they take their projects with them.

Table 1: RCKid's mastery ladder overview with editing experiences and their primary target devices

#	Tier	Primary Target
1	Asset Editor	RCKid
2	Expression based visual editor	RCKid
3	Control flow based visual editor	RCKid + Computer
4	Source Code	Computer + RCKid
5	Full SDK	Computer

Table 1 shows an overview of the mastery ladder tiers which are described in the following subsections. The ladder is under active development. At this point the internal mechanics of the game objects and runtime is validated and I am working on the on-device asset and visual editors. Remaining tiers are future work. The full ladder is presented here as the conceptual framework guiding the system's design.

A. *Asset Editors*

The entry point requires no literacy and no programming. Children modify visual and audio assets - tilesets and spritesets, tilemaps, sounds and music - for existing game templates. Although mechanics remain fixed, the expressive range is surprisingly large: changing characters, environments, and effects can fundamentally alter the feel of a game. This tier supports the youngest learners and introduces them to decomposition visually – a game consists of elements, the map consists of tiles, and so on. The creations are already tangible and can be shared via cartridges to be played by others.

B. *Expression based visual editor*

The first tier to introduce behavior changes: games expose objects (player, enemies, levels) and their properties and events, allowing children to tie game changes to events. To fit in the limited screen estate, the visual editor utilizes chevron-based visuals with icons that aid intra-expression composition as shown in Figure 3.

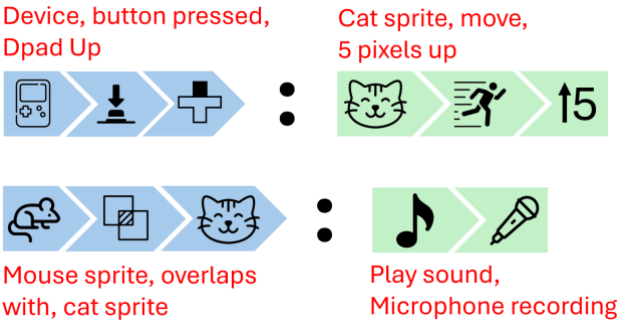

Figure 3: Expression based visual representation (design) of two actions in a cat-chase game. Blue chevrons represent trigger conditions, green actions.

Crucially, visual blocks reference assets - including newly created ones - enabling children to extend the game world while shaping its behavior.

C. *Control flow based visual editor*

This tier introduces state, conditionals, and structured control flow, similar to Scratch or MakeCode. This allows expressing more complex interactions. A key feature is the ability to define custom expression blocks using the block language: these appear as first-class blocks in the visual editor, allowing children to encapsulate complexity and reuse behaviors. This tier bridges visual composition and algorithmic thinking.

D. *Source Code*

At this stage, children transition from blocks to a textual DSL that mirrors the structure of the block language. The DSL provides richer composition mechanisms, more expressive control flow, and the ability to define new blocks for use in

lower tiers. The goal is to help students articulate algorithms textually while maintaining continuity with the game they have been building.

E. Full SDK

The final tier exposes the complete RCKid SDK. This allows creating new game objects to be used in the tiers below and to break from the constraints of the interpreter runtime used in the previous tiers, allowing for high performance, hardware-level programming. It is intended for teenagers and advanced learners who want to explore systems programming, graphics, or embedded development.

This is the only tier that does not provide on-device editing as changing the cartridge code at the SDK level requires flashing the firmware, and hence a computer.

IV. Hardware Prototype

RCKid exists in mkIII prototype, which was built in time for Christmas 2025 in order to be delivered to a few close beta testers (aged 5, 7 and 8). The purpose of this prototype is to demonstrate the viability of the device from hardware perspective and to serve as testbed for software development.

A. Technical Specifications

Figure 4 provides the overview of RCKid's architecture and visualizes the interactions between the main functional blocks.

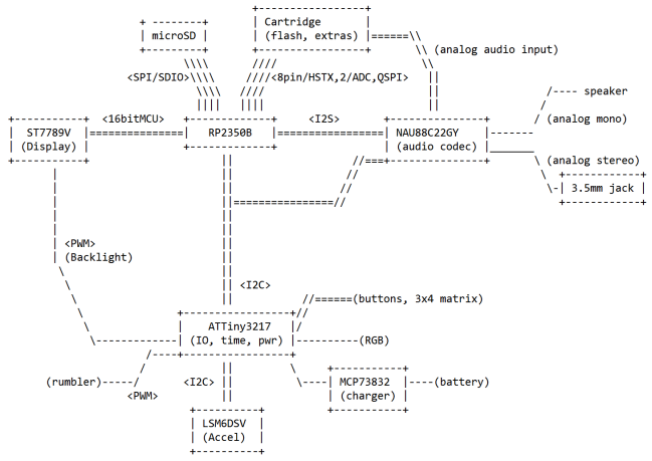

Figure 4: Overall architecture of RCKid showing main components and their interactions.

The device is powered by RP2350 by RaspberryPi, chosen for its alignment with RCKid's mission as well as its capabilities (520KB RAM, up to 16MB flash, 2 Cortex M-33 cores at 150MHz). 2.8" 320x240 16bpp IPS display is large enough for the intended use, yet does not overwhelm users with millions of pixels. The device also contains NAU88C22 audio codec with 16bit 48khz stereo sound via headphones, built-in mono speaker and microphone. DPad, A and B trigger buttons as well as Select & Start buttons come in different shapes for simple visual identification and support customized RGB backlight, all controlled by ATTiny3216 that also serves as power management controller, real-time clock and small non-volatile memory. The device uses micro SD cards for media storage and contains few extra peripherals such as rumbler motor and accelerometer with integrated pedometer (LSM6DSV). The device can run from either single li-po battery (standard 503759 1200mAh pack) or three AAA cells with both options providing roughly 10 hours of operation.

The cartridge connector provides power (3.3V, GND), QSPI for RP2350 external flash (W25Q128JVSIQTR) and 10 GPIOs that can be configured for general purpose IO, SPI, I2C, UART or custom IO peripherals. Furthermore 2 pins can function as analog inputs, 8 pins form the HSTX hardware interface and 2 pins can be multiplexed as analog inputs to the audio codec.

B. Prototype

This paper presents the prototype[2] as shown in Figure 1. It consists of a main PCB (4 layers) and cartridge connector daughterboard (printed and assembled by JLCPCB) and cartridge PCB (printed by JLCPCB, assembled by myself). The case is 3D printed by JLC3DP (subsidiary of JLCPCB) using transparent SLA resin.

The prototype has been delivered to 3 close beta testers (aged 5, 7 and 8 years) on Christmas 2025 for field testing. Results so far indicate that the transparent resin is too brittle for daily use, newer revisions will switch to 3D printed nylon back. Remaining aspects of the device have been successfully validated though.

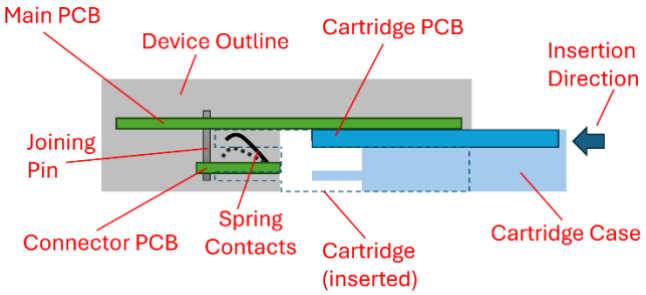

Figure 5 : Cross-section of RCKid's cartridge connector. The connector PCB carrying spring contacts is suspended from the main PCB via standard header pins that mechanically stabilize the connector board and electrically connects it to the main PCB.

The cartridge connector (Figure 5) is the main prototype-stage compromise in the mkIII design. It uses a small daughterboard with spring contacts suspended from the underside of the main PCB. This solution introduces uncertainty in the electrical characteristics of the cartridge interface and relies on tight mechanical tolerances (on the order of 1/10mm). For small batches this approach is acceptable, but for production runs of 1,000+ units the connector can be replaced with custom reverse-acting spring contacts mounted directly on the underside of the main PCB,

[2] This is already third iteration, first version (2023) was powered by the older RP2040, second version was a development board amenable for manual rework to verify the heavily revised design. The third prototype this paper presents integrates those results and is intended to verify the final product design, handling and materials.

enabling automated assembly and a cleaner, more predictable electrical design.

Table 2: RCKid cost estimates for 30+ unit orders, excluding full assembly and manual work

PCB, Assembly, Assembled Parts	$20-$30
3D printed parts (top/bottom/cartridge/buttons)	$10-$13
Battery	$8
Display	$8
Speaker	$3
Rumbler	$3
TOTAL	$50-$65

At the moment, prototype costs are largely driven by the custom component assembly fees. Table 2 shows estimated cost breakdown for larger orders (30+ units). The parts & third party assembly costs account for $50 to $65, the uncertainties being driven by changes in component costs & exact selection.

V. Future Work

I am currently finalizing a revised version of the prototype aimed towards cheaper bill of materials and simplified manual assembly. Notably I have removed the FM radio from the base device (in the future this can become dedicated cartridge), changed the microphone setup (I have swapped electret microphone for MEMS alternative), replaced hand soldered parts with spring-contact variants (speaker and rumbler motor) and reworked the battery connector.

On the software front, I have reworked the device's SDK for increased user-friendliness and speed of development. I have also created a proof-of-concept of the runtime responsible for user created games and am working on asset editors and on-device visual editors.

This is to be ready for the next big step: 30+ unit order to support a pilot program in local school where the device will be put in real classroom (and home) use. The aim of the pilot is to validate the basic premises of RCKid, namely that personal ownership, device self-sustainability for creative purposes and beyond classroom use have positive effects on acquiring STEM skills. Successful results of the pilot will then be used to get funding for expanding the ecosystem and more field testing.

VI. Responsible innovation

RCKid is designed with a strong emphasis on social responsibility, long-term sustainability, and the future of STEM education. A central part of the project's philosophy is personal ownership, which can be especially meaningful for children with limited access to technology. When a child has fewer toys and options in general, a creative tool like RCKid is more likely to become a recurring part of their daily life. This deeper engagement, aided by the on-device editing capabilities, leads to improved digital literacy. Ensuring that cost is not a barrier is therefore a long-term goal, as the children who stand to benefit the most are often those with the least access to personal digital technology.

The device is also designed for longevity and sustainability. Unlike many educational kits that serve a single age group or a narrow learning objective, RCKid is intended to grow with the child over multiple years. The progression from pre-literate asset editing to full programming extends the device's educational lifespan, reducing the need for frequent replacement. Its open-source design, durable construction, and repairability support long-term use, while the modular cartridge system allows new capabilities to be added without discarding the core device. This modularity reduces electronic waste and encourages a culture of tinkering, repair, and reuse.

Further responsibility is ensuring that RCKid integrates meaningfully into the existing educational ecosystem rather than contributing to its fragmentation. Teachers and learners are already familiar with existing environments, and introducing a new platform risks adding cognitive and logistical overhead. RCKid's cartridge system helps by supporting dedicated "compatibility" cartridges. As a concrete example, I have ensured that MakeCode Arcade cartridge with JACDAC [13] will be a low effort addition[3] that brings RCKid to the already flourishing ecosystem. MakeCode compatibility does not require compromising RCKid's identity — the personalised, beyond-classroom experience remains a cartridge swap away.

Finally, responsible innovation requires considering the future relevance of the skills the device aims to teach. The rapid rise of AI may change how programming is practiced, but it does not diminish the importance of foundational thinking skills such as decomposition, sequencing, abstraction, and debugging. These cognitive abilities predate modern programming languages and remain essential even in an AI-assisted world. RCKid focuses on cultivating these durable mental models through creative exploration, ensuring that the learning it supports remains valuable regardless of how the world evolves.

VII. Author bio

Most of my professional skills come from the field of programming languages and runtimes. I have PhD with thesis about large scale source code analysis [14] and I work as compiler engineer.

Throughout my career I have always enjoyed teaching, and to this day I keep a part-time job at the university teaching compilers and programming paradigms, where we introduced lambda calculus as a programming language [15]. I started being intrigued about the usefulness of embedded systems in STEM in 2015 when I was awarded Google CodeWeek funding to purchase hardware to jumpstart high

[3] MakeCode Arcade (PXT) currently supports RP2040; adding RP2350 and the RCKid hardware profile requires more than simple GPIO remapping but is entirely feasible. JACDAC support is particularly straightforward thanks to RP2350's programmable IO blocks.

school (ages 13-17) embedded programming course I designed at the time. I was impressed how the tangibility of the course motivated students to create a lot more complex projects than in previous years.

After becoming a parent, I became increasingly interested in bringing the joy of programming and building to even pre-literate children. This led me to explore how personal, tangible devices could support early creative expression, eventually culminating in RCKid, the platform presented in this paper.

I maintain a personal webpage with more information about my education, work experience and past projects at https://zduka.rckid.org.

VIII. Acknowledgements

Latest RCKid prototype has been partially supported by OSHWLab stars.

Icons in Figure 3 are from various authors on flaticon.com.

IX. References

[1] Papert, S., Mindstorms: Children, Computers and Powerful Ideas, 2nd ed. New York: Basic Books, 1993.

[2] Resnick, M. et al., "Scratch: Programming for All," *Commun. ACM*, vol. 52, no. 11, 2009.

[3] Flannery, L. et al., "Designing ScratchJr: Support for Early Childhood Learning Through Computer Programming," in *Proc. Interaction Design and Children (IDC '13)*, 2013.

[4] Ball, T. et al., "Microsoft MakeCode: Embedded Programming for Education," in *Proc. ACM SIGPLAN SPLASH-E Symposium*, 2019.

[5] Moskal, M. et al., "Web-based Programming for Low-cost Gaming Handhelds," in Proc. 16th Int. Conf. Foundations of Digital Games (FDG '21), 2021. doi: 10.1145/3472538.3472572.

[6] Ball, T. et al., "TileCode: Creation of Video Games on Gaming Handhelds," in Proc. ACM Symp. User Interface Software and Technology (UIST '20), 2020, pp. 1182–1195. doi: 10.1145/3379337.3415839.

[7] Underwood, L. et al., "MicroCode: live, portable programming for children via robotics," in *Adjunct Proc. 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24)*, 2024.

[8] BBC, "BBC micro:bit," 2024. [Online]. Available: https://microbit.org

[9] Arduboy, 2026. [Online]. Available: https://www.arduboy.com

[10] CircuitMess: Bit, 2026. [Online] Available: https://circuitmess.com/cz-en/products/bit-2-0

[11] Arduino, 2026. [Online]. Available: https://www.arduino.cc

[12] RaspberryPi, 2026. [Online]. Available: https://www.raspberrypi.com

[13] Devine, J. et al., "Plug-and-play Physical Computing with Jacdac," Proc. ACM Interact. Mob. Wearable Ubiquitous Technol., vol. 6, no. 3, 2022. doi: 10.1145/3550317.

[14] Maj, P., *Analyzing Large Code Repositories*. Ph.D. dissertation, Czech Technical University in Prague, 2023.

[15] Sliacky, J. and Maj, P., "Lambdulus: teaching lambda calculus practically," in *Proc. SPLASH-E 2019*, 2019.