Programs in the Palm of your Hand

How Live Programming Shapes Children's Interactions with Physical Computing Devices

Lautaro Cabrera
College of Education
University of Maryland
College Park, MD, USA
cabrera1@umd.edu

John H. Maloney
MIT Media Laboratory
Cambridge, MA, USA
jhmaloney@gmail.com

David Weintrop
College of Education
College of Information Studies
University of Maryland
College Park, MD, USA
weintrop@umd.edu

ABSTRACT

As physical computing devices proliferate, researchers and educators push to make them more engaging to learners. One approach is to make the act of programming them more interactive and responsive via live programming so that program edits are immediately reflected in the behavior of the physical device. To understand the impact of live programming on interactions with physical computing devices, we conducted a comparative study where children ages 11-15 programmed a BBC micro:bit device using either the MicroBlocks live programming environment or MakeCode, the micro:bit default environment. Results show that MicroBlocks users spent more time interacting directly with the physical device while showing different patterns of interaction compared to MakeCode users. We also found variations in the differences between environments related to activity structures. This paper contributes to the growing body of literature on how the design of interfaces—like programming environments—for physical computing devices shapes emerging interaction patterns.

CCS CONCEPTS

CCS → Human-centered computing → Interaction design → Empirical studies in interaction design

KEYWORDS

Live Programming; Physical Computing Kits; Children; Interaction Design

ACM Reference format:

Lautaro Cabrera, John H. Maloney and David Weintrop. 2019. Programming in the Palm of your Hand: How Live Programming Shapes Children's Interactions with Physical Computing Devices. In *2019 ACM Interaction Design and Children (IDC 2019), June 12-15, 2019, Boise, ID, USA. ACM, New York, NY, USA. https://doi.org/10.1145/3311927.3323138*

Introduction

The push to introduce children to computer science and the big ideas of computing is increasingly including physical computing devices. Ten years ago, engaging children with computing usually meant sitting them down in front of a computer screen and keyboard. Today, a new landscape of devices is changing how, where, and when kids can interact with computing. Robots, microcontrollers, wearables, and dozens of other elements in the Internet of Things now serve as entry points into the world of computing for young users. With this growing ecosystem of computing devices comes a similar growing set of ways for learners to interact with and control these computational devices.

The physical nature of the devices and their technical capabilities shapes both how a child can interact with them and what they can do through that interaction. Physical computing devices can include accelerometers, sensors, and physical inputs like buttons and switches alongside visual and audio inputs and outputs. These capabilities introduce new sets of challenges and opportunities for the interaction designers tasked with figuring out how to make these devices accessible and engaging to children. Towards this end, this paper seeks to understand one specific aspect of the design of introductory programming environments that has found widespread success in *virtual* introductory programming environments but is rarely seen in programming environments for *physical* devices—live programming. In a live programming environment, users can modify a program as it is running, thus allowing them to see the results of edits immediately. The resulting *liveness* of the environment makes for a highly engaging and interactive programming experience and has been credited for part of the success of the Scratch programming environment [1]. This direct control over the resulting program has interesting potential ramifications in the context of physical computing as it directly connects the physical device in the hand of the programmer with the virtual code they are writing. More specifically, this paper seeks to understand how liveness impacts user interaction in the context of physical computing by answering the following research question:

How does an environment supporting live programming impact the way kids interact with and program a physical computing device?

To begin to answer this question, we conducted an exploratory study using two comparable programming environments for the BBC micro:bit, one that supports live programming and a second that does not. We recruited 11 children between the ages of 11 and 15 and had them write programs for the micro:bit using one of the two environments. We then analyzed the results looking for differences in the programming practices used as well as differences in how they interacted with the device while programming it. This paper contributes to our understanding of the design of technology for children and how the interplay between programming a device and physically manipulating it shapes resulting interactions. Further, this paper advances our understanding of how to design programming environments for physical computing devices, specifically with respect to how liveness shapes engagement and interaction.

The paper continues with a review of relevant prior work. We then introduce the technologies that are the focus of this paper, the BBC micro:bit and our two programming environments: MicroBlocks and MakeCode. Next, we detail our methods before presenting our findings on how liveness shaped interactions with the physical computing device. We conclude with a discussion of the implications of this work with respect to design and interaction.

Prior Work

Microcontrollers and Physical Computing Kits

Microcontrollers and physical computing devices specifically designed for kids have a long and rich history [2]–[4]. The idea of programmable microcomputers for kids grew out of a collaboration between MIT's Media Lab and the LEGO group, which resulted in a programmable brick where basic programs could be written to control motors and servos built with the LEGO bricks [5], [6]. Over time, the LEGO bricks added new capabilities, particularly around the introduction of sensors that made it easier for builders to more directly interact with the world around them [7]. These early construction kits were the antecedents to the popular line of Lego Mindstorms construction kits still in use today. As the tools and their abilities advanced, designers sought to give users more direct access to the underlying computational capabilities, moving beyond the "black box", making processes more visible and accessible [8].

Over time, designers of physical computing kits for kids began to focus on ways to broaden participation in computing and access to knowledge. For example, designers sought to create programmable computing kits that were accessible to kids as young as 4 years old by creating direction manipulation, program-by-demonstration robots [9]. A second example of expanding access was the creation of open-hardware and open-source software boards that made it possible for devices to be built and used in places where such technologies were historically not available [10], [11]. This wave of computing kits reflects not a technological advance, but a shift in the focus of the design, towards new audiences, new interaction patterns, and new opportunities to reach children.

As the landscape of physical computing devices matured and toys and toolkits grew in popularity, new designs and new architectures emerged. Many of these new platforms and updates to existing platforms sought to achieve the goal of "low-floors, high ceilings, and wide walls" by adopting and implementing recommendations from the IDC community [12]. Constructions kits such as Topobo [13] and RoBlocks (later Cublets) [14] highlight new modular construction approaches for building and controlling computational constructions. The growth in popularity and diversity of physical computing kits has resulted in a growing number of tools and platforms designed explicitly for younger learners. A recent review of such technologies identified 25 distinct computational kits for kids under the age of 7 that include a physical component [15]. This review highlights the diversity of ways kids can be supported in giving instructions to physical computing devices. These included writing programs in a virtual context and transmitting them to the physical device (e.g. Finch Robot [16]), giving instructions directly to the physical device using buttons or on-board controls (e.g. Code-a-pillar), as well as tangible programming environments, where physical blocks are manipulated to compose programs that are then run either by a physical or a virtual sprite (e.g. Strawbies [17]) or a physical robot (e.g. KIBO [18]).

Novice Programming Environments

Interestingly, the earliest programming environments design for young learners and the earliest physical robotics kits came out of the same research group at MIT. The creation of the Logo programming language was the first time that computing power was put at the fingertips of young people [19]. As computing became more widespread, programming language designers began to recognize the utility of having languages that were more accessible to novices and could be used in instructional contexts.

Today, novices are increasingly being introduced to programming using graphical programming environments that make programming easier and more accessible to learners [20]. One notable approach that has seen widespread adoption is block-based programming. Block-based programming uses a programming-command-as-puzzle-piece metaphor to provide additional information to the user about how and where a command can be used. Using visual cues and enforcing basic composition rules, the environment itself can minimize or altogether eliminate syntax errors while still retaining the practice of authoring programming by assembling commands one-after-the-other [1]. Led by the popularity of environments such as Scratch [21], Alice [22], and Blockly [23], block-based programming has a growing presence in both formal and informal educational contexts. This can be seen in the rise of curricula built around block-based tools (e.g. Exploring Computer Science [24] and many of the Code.org courses [25]) as well as a growing number of toys and games for younger learners that leverage the block-based approach [26]. Research on novices working with block-based tools shows that students perceive block-based programming as being more accessible [27] and also finds learners complete programming tasks more quickly [28] and perform better on assessments in block-based contexts [29], [30].

Creating intuitive and accessible programming interfaces for novices can also be seen in the world of physical computing kits. While the most popular microcontrollers are still mainly controlled with professional languages and technologies (e.g. C/C++ languages for Arduino boards), a growing number of platforms are introducing learner-friendly programming interfaces for their devices. For instance, the popular Scratch environment includes extensions for programming a number of devices, including the Makey Makey, micro:bit, Lego Mindstorms EV3 and Lego WeDo [31]. Collectively, innovations in the design of novice programming environment have made it easier for novices and younger users to programmatically control microcontrollers and physical robotics kits.

Live Programming

The defining feature of a live programming environment is that changes made to the program are reflected immediately giving the programmer the impression of changing the program while it is running [32]. This means no manual compilation or run step is required for edits or additions to a program to be reflected. Live programming is not a recent innovation (e.g. [33], [34]), however, there has been renewed attention in both industry and in academia in recent years [35]. In a 2018 literature review, researchers found 87 academic papers on the topics of live programming with another 112 papers on the topic of live coding, and 31 papers focusing on exploratory programming [32]. All three of three terms (live programming, live coding, and exploratory programming) share the characteristic of *liveness* in the act of programming but differ slightly on how it is used. Live coding is generally concerned with the creation of art and performance (e.g. [36], [37]), whereas live programming is more explicitly focused on the act of programming (e.g. [38], [39]). In the case of exploratory programming, the focus is on tools that support the creation of programs where final requirements are not yet defined, so ease of quick prototyping is emphasized (e.g. [40]). Each of the communities rely on different affordances of *liveness* with a growing literature-base teasing apart the various supports.

The literature cites a number of advantages of live programming. First is how the approach minimizes latency between programming and seeing its effect, aiding in the development process [41], what Burckhardt et al. [42] call the “temporal and perceptive gap”. A second benefit relates to enabling the act of programming to become a form of performance in a way not possible in non-live environments [43]. A third benefit of particular relevance in this work is the pedagogical potential of live programming. In live programming environments, learners receive immediate feedback on the effect a given instruction, parameter argument, or value will have on the resulting behavior of the program. This feature is particularly well-suited for languages and environments focused on visually rendering objects [44] and can be seen in the popularity of languages like Processing in educational contexts [45]. Other work shows how live programming can facilitate debugging by providing insight into the state of a program and how changes to the code impact outcomes [46]. Victor, in an influential series of prototypes, demonstrated a number of ways that features of live programming could support novices learning to program including direct manipulation of function parameters that immediately influence the resulting program output [47]. However, other research has failed to find difference in learner performance in live environments compared to non-live alternatives [48], however this work was quite different than this study as it was working with undergraduates in a formal learning context and did not consider aspects of engagement or programming practice.

Device and Environments

At the heart of this paper are three technologies: a small, inexpensive microcontroller called the BBC micro:bit and two programming environments for it: MakeCode and MicroBlocks.

Meet the BBC micro:bit

The BBC micro:bit (Fig. 1) is an inexpensive (around $20 in the United States), credit-card sized computer created by the British Broadcasting Corporation (BBC) as part of an effort to increase digital skills among UK school children and "inspire the digital visionaries of the future" [49]. In early 2016, the BBC distributed free micro:bit's to every 7th grader in the UK school system (about 800,000 children) [50], then spun off the non-profit micro:bit Educational Foundation with the goal of shipping 100 million micro:bits to children around the world [51]. To date, over 2 million micro:bits have been distributed [52].

The micro:bit combines a 32-bit ARM microcontroller with sensors, buttons, an accelerometer, and a 5x5 grid of LEDs that can show graphics or alpha-numeric outputs. Although the micro:bit can be connected to external devices such as lights, motors, and sensors, its rich set of built-in features allows learners to focus on programming first, then add electronics later, if desired. This makes it possible to use the micro:bit with younger users than systems that require learning electronics and programming at the same time, such as the Arduino.

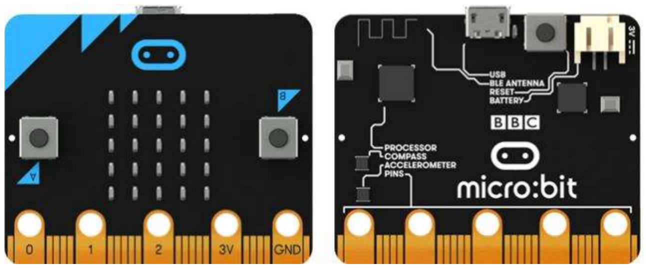

Figure 1: The front (left) and back (right) of the micro:bit.

To program the micro:bit, the device must be physically connected by a cable (i.e. tethered) to the laptop or computer on which the program is being written. It is possible for the user to interact with the micro:bit and its features (e.g. buttons and accelerometer) while still tethered to the computer. In a non-live programming environment (like MakeCode), the program must be downloaded and transferred to the micro:bit manually, a process that usually takes around 10 to 20 seconds. In a live programming environment (like MicroBlocks), the program is downloaded automatically in the background so the programming being run on the micro:bit always matches what the user sees on the screen.

Meet MicroBlocks

MicroBlocks (Fig. 2) (http://microblocks.fun) is a block-based programming environment designed for physical computing kits. Inspired by Scratch [21], GP [53], Snap! [54], and other "low-threshold" block-based programming environments, the interface shares many features common to this popular design approach for introductory programming environments. These features include a blocks palette on the left side of the interface organized into high-level categories from which users can drag blocks onto the canvas, where scripts are assembled. The version of MicroBlocks used for this study was customized for the micro:bit, so it included categories such as LED and Pins and blocks such as Tilt X and show LEDs to give the user control over the capabilities of the device.

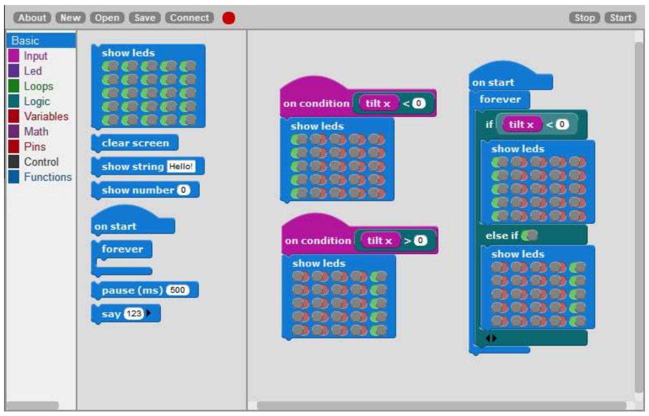

Figure 2: MicroBlocks and two solutions for Challenge 1.

MicroBlocks combines the benefits of liveness when tethered with autonomous operation when untethered. While the user is writing and debugging their code, MicroBlocks behaves like a tethered system, providing liveness, data readouts, and immediate feedback and allowing the user to discover new features and tinker with their code as they would in Scratch. This means a user can click on a block, such as button A, and get a real-time readout of whether the button is being pressed on the device. Behind the scenes, the user's code is incrementally downloaded and run on the physical device using a virtual machine to avoid the need for a compilation step. When the device is disconnected from the laptop, it continues to run the user's program autonomously. This design makes MicroBlocks a live programming environment with no explicit compilation or download step that allows the user to untether their device at any time and have the latest version of their program continue running on the device. A longer discussion of the technical implementation of MicroBlocks can be found in [55]. MicroBlocks was chosen for this study due to its novelty of being a live programming environment for a physical computing device.

Meet MakeCode

MakeCode (Fig. 3), (https://makecode.microbit.org/) the programming environment linked to directly from the micro:bit's home page, was developed by Microsoft and is an open source web-based environment for coding physical computing devices [56]. MakeCode is able to run entirely within a web browser, meaning there is no software to install and the platform can run on a variety of different types of devices (including tablets and smartphones). Like MicroBlocks, MakeCode provides a block-based programming interface that includes micro:bit-specific blocks alongside conventional programming commands. MakeCode also includes a text-based programming editor allowing users to write programs for the micro:bit in JavaScript.

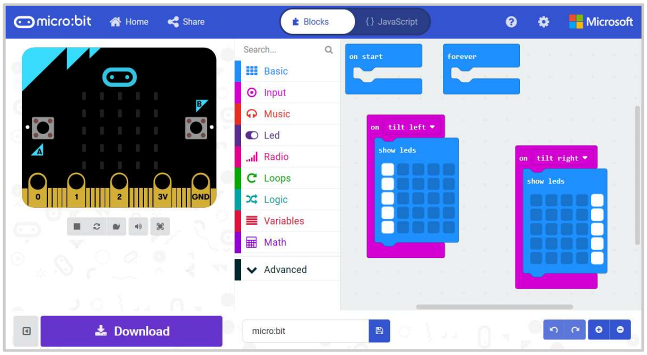

Figure 3: MakeCode and a solution for Challenge 1.

There are two defining features of MakeCode that distinguish it from MicroBlocks. First, in order to run a program on the physical device, the program must be compiled and downloaded to the device. This can be accomplished by clicking the download button and then dragging the file from the local computer's drive onto the micro:bit. For this study, the computer was configured so MakeCode programs downloaded directly to the micro:bit, a procedure that usually takes around 12 seconds. The second feature unique to MakeCode is the inclusion of a virtual micro:bit simulator (shown at the top left of Fig. 3). This simulator makes it possible to run micro:bit programs without needing to go through the download procedure. The simulator includes the 5x5 LED grid, two buttons that can be "pressed" by clicking on them, and supports tilting and shaking by moving the mouse over the simulator. These features (the need to download the program to run it on the physical device and the virtual simulator), along with the fact that MakeCode is the environment used on the official micro:bit website, are the reasons why it was chosen as the second environment for this study.

Methods

Study Design

This paper reports on the findings of an exploratory comparative study investigating if and how a live programming environment shapes the way children interact with and program physical computing devices. To begin to answer this question, we recruited 11 children and had them write a series of short programs using either a live programming environment (MicroBlocks) or a conventional programming environment that required downloading and transferring programs to the physical device (MakeCode). As much as possible, the two programming environments were isomorphic, meaning the two environments had the same set of blocks and presented the same set of capabilities to the user. The goal of the study design is to try and control for external factors beyond the live programming component as much as possible. Differences between the two environments beyond the live programming feature are discussed in greater detail later in the limitations section.

Participants & Procedures

Children were recruited to participate in the study via email, social media, and personal communications. A total of 11 children (6 males, 5 females) between the ages of 11 and 15 (M = 13.2, SD = 1.33) participated in the hour-long study and were allowed to keep the micro:bit as compensation.

Participants were assigned to use either the MicroBlocks or MakeCode environment. Our approach for assigning participants to an environment was to alternate environments between each interview: the first participant used MakeCode, the second used MicroBlocks, the third used MakeCode, and so on. The MicroBlocks group was comprised of 5 girls and one boy (age M = 13.5), while the MakeCode group was composed of 5 boys (age M = 12.8). The gender imbalance was coincidental but still potentially problematic and is discussed in the limitations section.

The study procedure began by going through a consenting procedure and was followed by a short introduction to the micro:bit, showing off the main features of the device. This included showing participants the 5x5 LED matrix, the two buttons, and describing how the device has an accelerometer so can detect being tilted or shaken. Next, the researcher gave the participant a brief introduction to the programming environment that he or she would be using. This included describing how to compose a program as well as how to run it on the micro:bit. In the case of the MakeCode, the interviewer also described how the micro:bit simulator worked on the screen. As much as possible, the introduction to the programming environments was the same between the two conditions. The introduction lasted around 3 minutes.

The remainder of the study was divided into two 20-minute parts. The first "tinkering" phase had participants play around with the micro:bit to create any program that they wanted. The goal of this stage was to give kids time to get acquainted with both the device and the programming environment, as well as to see how the environments shaped interactions with the device during free play. Since the participants had already seen the initial demonstration, we only provided additional assistance if the participant requested it. While we wanted to give participants the freedom to explore the tools, we also wanted them to acquire some initial knowledge that would at least give them a chance to succeed in completing the challenges.

The second phase of the study consisted of three programming challenges. Whereas the tinkering phase was looking at aspects of engagement and playfulness, the challenges were designed to get the participants to use specific programming concepts (e.g. conditional logic) and capabilities of the device (e.g. button pressing). The tasks were designed to be of increasing difficulty.

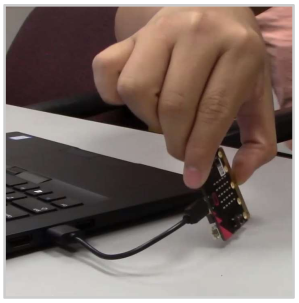

Figure 4: Tilting micro:bit on Challenge 1

The first challenge was to create a program so that when the micro:bit was tilted to the left (Fig. 4), its leftmost column of LEDs would turn on and when tilted to the right, the right-most column of lights should turn on (Fig 5a). For the second challenge, participants had to create a program so that when the left button was pressed and the micro:bit was tilted to the left, and arrow would show on the left side (Fig 5b). The same arrow would need to show on the right side when the right button was pressed and the micro:bit was tilted to the right.

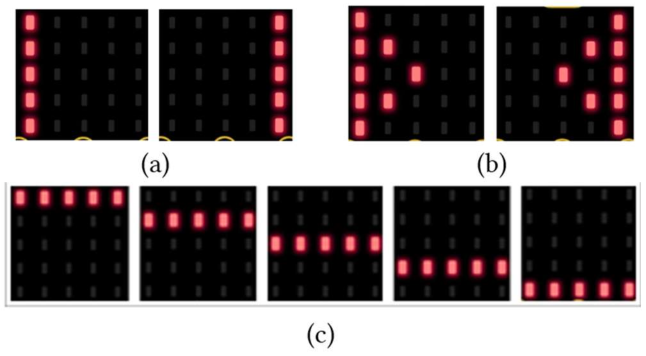

Figure 5: LED patterns for challenges 1 (a), 2 (b), and 3 (c).

The last challenge was to create a program so that when the micro:bit was tilted upwards, it would display a "cascade" of lights where the top row lit up, followed by the 2nd row and so on, as shown in Fig. 5c. The participant was asked to make the cascade repeat three times.

Data Collection & Analysis

We captured three main sources of data. First, we video-recorded each participants' interaction with the micro:bit (Fig. 4). Second, we used screen capture software to record how participants interacted with the programming environments. Third, we audio-recorded the sessions. These data sources were individually analyzed and then collectively deductively coded using NVivo [57].

For each screen recording, we coded each time a new block was added to the program, as well as each time the participant explicitly ran their program on the micro:bit. In MakeCode, this took the form of the user clicking on the "Download" button and transferring the program onto the device. In MicroBlocks, the live feature incrementally downloads programs automatically, so we recorded only when participants "forced" a run by clicking on the "Start" button or on program script. For participants using MakeCode, we also recorded each time they used the simulator—defined as tilting the simulated micro:bit or clicking on either of its buttons. These results are reported in the "Sim. Buttons" and "Sim. Tilts" columns in Table 1.

As we are interested in how the children interacted with the micro:bit, we quantitatively analyzed each micro:bit video, coding for four specific types of interaction: how many times the participant touched the micro:bit, how long (in total) they touch the device for, how many times they tilted the micro:bit, and how many times they press the micro:bit's buttons. We also calculated the average time per touch for each child.

While our quantitative findings are reported, we did not conduct formal statistical analyses to compare interactions between the two groups. This is due both to the size of our participant pool (n = 11) as well as the exploratory nature of the study. As such, the findings below report mainly qualitative findings seeking to understand how the live programming feature shaped the way children engaged and interacted with the technology. When we use quantities to describe interactions, we do so to illustrate differences or similarities across groups—not to make generalizable conclusions. One goal for this work is to lay the groundwork for future studies with larger participant pools that would allow us to make more robust inferences. Beyond the quantitative measures, the analysis of the micro:bit video and the screen capture were then aligned to create timelines for each participant showing their sequence of interactions. This was done to analyze participants' behavior and look for patterns between the physical and virtual aspects of programming the micro:bit.

Findings

This section begins by discuss findings from the Tinkering phase before moving on to results from the challenges phase.

Tinkering

Micro:bit Capabilities.

The first difference that emerged in our analysis of the tinkering phase was in the micro:bit capabilities that participants chose to incorporate into their programs. Five of the six participants who used MicroBlocks created programs that involved either button pressing or device tilting. In contrast, only 2 out of the 5 MakeCode users created such programs. All participants also created programs that involved the LED lights in the micro:bit. This suggests that the live programming aspects of MicroBlocks and the in-editor simulator of MakeCode impacted how children interacted with the micro:bit—particularly on their intent to integrate physical interactions. One potential explanation is that the ability to immediately test the effects of tilts or button presses in the live programming environment led to an increase in the use of those interactions in the program. However, we simply counted participants who *attempted* to integrate tilting or buttons, not just those who were successful in doing so. Therefore, another possible explanation is that the mere *expectation* of liveness that MicroBlocks participants had led them to attempt to integrate physical interactions that could be tested immediately. The same principle was true for the MakeCode participants as most created programs that could be tested immediately by looking at the LEDs in the simulator. We will return to the idea of expected liveness again in our analysis of the challenges.

Table 1. Interaction averages per group. Times are in seconds.

	n	Gender	Age	Blocks	Runs	Time touching	Time per touch	micro:bit Touches	micro:bit Buttons	micro:bit Tilts	Sim. Buttons	Sim. Tilts
MicroBlocks												
Tinkering				21.6	5.6	65.9	4.54	14.5	17.3	1.8	–	–
Challenges				21	6.3	425.8	22.8	18.7	13.3	67	–	–
Total	6	1M / 5F	13.5	42.6	11.3	491.7	14.8	33.2	30.67	68.83	–	–
MakeCode												
Tinkering				23	3.8	59.6	11.04	5.4	28.4	14.6	4.4	4
Challenges				19.2	5.4	223.2	22.3	10	34.4	42	6	21.4
Total	5	5 M	12.8	42.2	9.2	282.8	18.4	15.4	62.8	56.6	10.4	25.8

Of the 7 children who created programs with a physical interaction, all but one of them used only one type of interaction, either tilting or button pressing, but not both. This suggests that while the live programming encouraged children to experiment with the physical capabilities, they were not motivated to try all of the capabilities on display. It is possible that these results stemmed from the relatively limited time provided to explore the new device and its programming environment.

Interaction Patterns.

Looking at our basic measures of interaction during the Tinkering stage (Table 1), we see that MakeCode users touched the micro:bit for longer periods of time but less often. In other words, users of MicroBlocks seemed to make short but numerous touches, while MakeCode users made fewer, longer micro:bit touches. In terms of specific interactions, the MakeCode group recorded more button presses but fewer tilts than the MicroBlocks participants during this phase. Within the MakeCode group, 4 out of the 5 participants used the simulator during the tinkering phase.

There are a number of possible factors that contributed to these differences we noticed between groups. For example, MicroBlocks participants could only interact with the micro:bit one way (by touching it), while MakeCode participants could both touch the physical device or use the simulator. This may have contributed to the additional number of touches in the first group. Therefore, it is possible that the presence of the simulator promoted MakeCode users to more heavily focus on the screen—therefore neglecting interactions with the device. As one participant from this group explained after the challenges: "*I would prefer to program something that is just in the computer [as opposed to a device] because that was easier*". It's also possible that the pattern of "short but often" touches in the MicroBlocks group stemmed from the live feature in the environment. Seeing immediate changes in the device may have prompted more interactions with it to check programming progress. This more direct interaction and immediate feedback could explain the greater number of interactions with the micro:bit.

In summary, MakeCode participants tended to touch the micro:bit less often but for longer periods of time per touch, while MicroBlocks users touched the device more often but for shorter durations.

Challenges

Challenge Success Rates.

Looking across the two conditions, we found the MakeCode group had slightly more success with respect to completing the challenges. All five of the MakeCode participants completed the first challenge, compared to only four of the six MicroBlocks participants. One child from each group completed the second challenge. Further, children in the MakeCode condition typically finished the first challenge more quickly, leaving more time to work on the second challenge.

Our explanation of this difference has less to do with the live programming capability of the environments and instead, we attribute it to slight differences in the programming interface. As shown in Fig. 3., MakeCode has a single event block for a tilt event. On the other hand, MicroBlocks required participants to use the block tilt X alongside another command (either an if block or the on condition event block, both shown in Fig. 2). It is possible that children in the MicroBlocks group were less successful in completing the challenge because it required a more complex combination of blocks. As such, the finding from this difference supports other literature looking at how language design impacts novice programmers [29], [58], but does not lead to conclusive evidence on the role of live programming in accomplishing tasks.

Interaction Patterns.

During the challenges phase, we noticed that participants in the MicroBlocks condition seemed to interact with the micro:bit more often and for a longer total amount of time—an interaction pattern that aligns with the results of the tinkering phase. Addutuinally, MakeCode participants pressed the micro:bit's buttons more often than MicroBlocks users while the opposite was true for tilting interactions. Finally, the number of simulated tilts by MakeCode users seemed to increase during the challenges phase, which can be logically explained by the nature of the challenges.

The differences in interaction during the challenges can in part be explained by the success rates reported in the previous section. Because the MakeCode environment has built-in blocks for actions associated with tilting, it is possible that kids who used MakeCode tilted the device fewer times because they did not *need* to test their tilting logic as often. Likewise, because the MakeCode participants advanced to the second challenge faster, which required them to program a button press, it is not surprising that we recorded more button presses in the MakeCode condition.

Programming Practices.

Combining the analyses of the interaction cameras and the screen recordings, we identified different patterns of programming during the challenges. We hypothesized that each group would show different interaction patterns based on the affordances of the environment. We expected to see MicroBlocks users add blocks, test the results on the micro:bit, and then go back to adding blocks to correct or expand on the program (Fig. 6a). On the other hand, we expected MakeCode participants to use the simulator for iterative development and then download and test the program on the physical device once it was closer to completion (Fig. 6b).

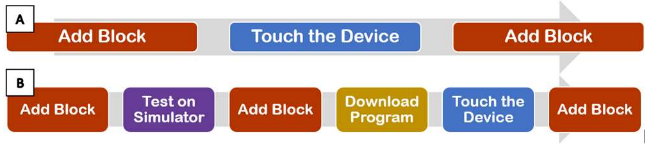

Figure 6: Hypothesized programming patterns.

All six of the participants in this group showed instances where the sequence shown in Fig. 6a was followed: add block, touch the device, add block was repeated multiple times consecutively. While all 5 MakeCode participants used the simulator at least once during the challenge phase, three of them used the simulator to test recently added blocks and continued to edit the program based on the simulator's results without interacting the device.

Comparing the interaction patterns across environments, two differences arose. First, MicroBlocks users appeared to alternate between adding blocks and touching the device more often than MakeCode users. A second—and unexpected—result was that some children in the MakeCode group often showed our hypothesized MicroBlocks interaction of making modifications and going right to the physical device (shown in Fig. 6a). For example, one participant who was working on challenge 2 found that his LED pattern was incomplete—the micro:bit was turning on three lights instead of the required eight. To fix this problem, he proceeded to change his program to add the correct amount of lights on his show LEDs block and then immediately tilted the micro:bit to see any changes. After unsuccessfully trying to tilt the micro:bit twice, he realized his mistake and pressed the "download" button. Here again, we see the phenomenon of *expected liveness*, even when the environment is not actually "live".

Logically, engaging in the block-touch-block pattern in MakeCode is problematic as the code modifications will never be reflected on the physical devices as they have not been downloaded to it. But, as illustrated in this example, some participants made edits to their program and then checked to see the outcome on the micro:bit despite the micro:bit still running an older version of the code that did not have the most recent change. Four out of the five MakeCode participants touched the device after adding blocks without downloading the program first. Two of them did so repeatedly. In other words, it appeared as though the participants were expecting a live programming behavior that was not present. This expectation was particularly problematic while debugging as there was a case where—contrary to our example—a correct modification was made but the participant did *not* think it was correct because he did not see the changes reflected on the device. This is a concrete demonstration of one of the benefits of live programming reported in the literature [59].

Discussion

The numerous small comparisons between the two groups help us illuminate the bigger point of how the liveness in MicroBlocks impacted how participants programmed and interacted with the micro:bit. During the tinkering stage, liveness seemed to prompt shorter, more numerous interactions with the physical device, as well as more attempts to integrate button presses and tilts into the program. When the activity shifted to structured tasks where interaction with the micro:bit was *required*, those differences persisted, with participants in the MicroBlocks condition touching the device more often and for a longer total amount of time. However, it is also possible that the differences in interaction were not *exclusively* due to liveness. The presence of the simulator in MakeCode may have changed how and when users chose to interact with the device directly.

In terms of programming patterns, the liveness of MicroBlocks seemed to support an incremental process that continuously alternated between adding blocks to the program and testing their effects directly on the physical device. On the other hand, the simulator in MakeCode provided a few situations where testing could be done virtually, and physical interaction was limited to testing final solutions. In fact, one participant completed the whole program using only the simulator, stating he was done without testing it on the physical device.

The Expectation of Liveness

One of the interesting findings to emerge from this study was documenting children's expectations of a live programming interaction, even when the environment did not have that capability. This can be seen most clearly by the fact that all but one of the participants in the MakeCode condition tried to test a programmatic change on the physical board without downloading it to the board first. This is a potentially significant finding with respect to the design of programming environments for novices. As more and more interfaces and devices demonstrate live programming behavior, young users may come to expect such immediate feedback, meaning non-live environments may become less engaging or enjoyable. It is also possible that the mismatch between the users' expectations and the behavior of the system may become the source of errors and frustration. The lack of liveness can also negatively impact the user's experience, as one MakeCode participant said mid-session: "*hurry up and save!*" In this case, the child knew the environment was not updating "live", but it was clear he would have preferred if it was.

Environment Design Impacts

The two environments used in this study also provided some insights into the tensions of designing environments with low floors and high ceilings [12] and how interface design decisions we make impact the paths users take. One clear example of this from this study can be seen in how the design of the languages impacted the transition from challenge 1 to challenge 2. Challenge 1 asked users to write a program to respond to the micro:bit being tilted. Challenge 2 added the requirement of a new behavior when the button was pushed while the micro:bit was being tilted. MakeCode provides a single event block (on tilt left) that made it easy for participants to complete the first challenge, but in order to modify the program to also accomplish the second challenge, the child would need to add a conditional statement inside the event, as there is no on tilt left and button pushed command. As a result, all MakeCode participants completed the first challenge but most struggled with the second. In this way, MakeCode was low-threshold but left a significant conceptual hurdle for Challenge 2.

MicroBlocks, on the other hand, did not have a single block to respond to the micro:bit being tilted. Instead, it required the user to build the behavior from a generic on condition event block.

This made challenge 1 more challenging (two of the MicroBlocks participants did not complete it), but the transition from a tilting behavior to a tilting-and-button-pressed behavior was relatively straightforward. In this difference between how the two languages supported the novice in implementing this functionality, we can see the impact of language and interface design shaping user interactions.

Physical Computing Devices for Younger Ages

One contribution of this study is additional evidence to support the introduction of children to basic programming, and the field of computer science more broadly, through physical computing. All of the participants seemed to enjoy the experience and were excited to keep the micro:bit at the conclusion of the session, with some saying that the tinkering stage was "*fun*" and "*cool*".

While this study did not focus on learning outcomes, the identified differences in interaction due to the liveness do have potential educational implications. The programming patterns users develop and their emerging conceptualizations of basic computing ideas are all shaped by the tools used. Identifying how liveness impacts learning in physical computing contexts is a clear future direction of our work—especially given the large-scale rollouts of physical computing curricula (e.g. [60]).

Limitations and Future Work

There are a few limitations related to this work that are important to discuss. First, we recognize that the gender breakdown of the two conditions is problematic. It is possible that some of the differences between groups we reported were due in part to the differences in gender between the two conditions. That being said, we do not see an obvious reason that would be the case, nor did we identify any instances where gender seemed to play a role in shaping interactions. Nevertheless, it is a noted limitation and one that will be addressed in future iterations of this line of work.

A second limitation stems from differences between the two programming environments beyond liveness. While efforts were made to make the two environments as similar as possible, differences remained. Specifically, the variation in how tilting was handled in the two conditions potentially confounds some of the differences observed, especially in the challenges. MakeCode required a more pronounced tilt to recognize the interaction; MicroBlocks was more sensitive to the micro:bit's orientation. Redesigning the environments to more clearly isolate the liveness is another avenue of future work as we look to scale up the study.

Finally, as this was the first study of its kind, we began with a relatively small sample and a short intervention. We understand that the numerical data we show is merely descriptive and insufficient to make strong causal claims about the effects of liveness on children's interactions with the micro:bit. In future work, we hope to work with larger numbers of participants and in more ecologically valid contexts. Our hope is the findings from this study serve as the foundation for future, larger research undertakings.

Conclusion

In this paper, we reported the findings of our exploratory study investigating how live programming impacts children programming and interacting with a physical computing device. Our results suggest that liveness does affect children's interactions, both in free play and structured activities. This work advances our understanding of the design of programming environments for physical computing kits and the ways kids interact with them. Given the rise of such technologies, understanding how best to support children in meaningfully and successfully interacting with such devices can potentially have a significant impact. Through this study, we hope to help inform the design of future platforms so as to help children have positive early experiences with computing.

Selection and Participation of Children

As the ideas and actions of children were central to this work, care was taken to ensure our young participants felt comfortable, were treated respectfully, and all ethical research standards were followed. The primary method of recruitment was personal communications with parents engaged in research projects associated with our institution. The parents of minors completed consent forms on their behalf and the participants themselves completed child assent forms. Participants were given the micro:bit to take home for participating.

REFERENCES

[1] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The Scratch Programming Language and Environment," *ACM Trans. Comput. Educ.*, vol. 10, no. 4, pp. 1–15, 2010.

[2] P. Blikstein, "Gears of Our Childhood: Constructionist Toolkits, Robotics, and Physical Computing, Past and Future," in *IDC '13*, 2013, pp. 173–182.

[3] F. Martin, B. Mikhak, M. Resnick, B. Silverman, and R. Berg, "To Mindstorms and Beyond: Evolution of a Construction Kit for Magical Machines," in *Robots for kids*, San Francisco, CA: Morgan Kaufmann Publishers Inc, 2000, pp. 9–33.

[4] T. S. Mcnerney, "From turtles to Tangible Programming Bricks: explorations in physical language design," *Pers Ubiquit Comput*, vol. 8, pp. 326–337, 2004.

[5] F. Martin, R. Sargent, and B. Silverman, "Programmable Bricks: Toys to think with," *IBM Syst. J.*, vol. 35, no. 3&4, pp. 443–452, 1996.

[6] R. Sargent, M. Resnick, F. Martin, and B. Silverman, "Building and Learning with Programmable Bricks," in *Constructionism in Practice: Designing, Thinking, and Learning in a Digital World*, Y. B. Kafai and M. Resnick, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 1966, pp. 161–173.

[7] M. Resnick *et al.*, "Digital manipulatives: New toys to think with," in *Proceedings of the CHI '98 conference*, 1998.

[8] M. Resnick, R. Berg, and M. Eisenberg, "Beyond Black Boxes: Bringing Transparency and Aesthetics Back to Scientific Investigation," *J. Learn. Sci.*, vol. 9, no. 1, pp. 7–30, 2000.

[9] P. Frei, V. Su, B. Mikhak, and H. Ishii, "Curlybot: designing a new class of computational toys," *Proc. SIGCHI Conf. Hum. factors Comput. Syst.*, 2000.

[10] A. Sipitakiat, P. Blikstein, and D. P. Cavallo, "GoGo Board: Augmenting Programmable Bricks for Economically Challenged Audiences," in *In Proceedings of the International Conference of the Learning Sciences*, 2003, no. 617.

[11] A. Sipitakiat and P. Blikstein, "Think globally, build locally: a technological platform for low-cost, open-source, locally-assembled programmable bricks for education," *Proc. fourth Int. Conf. Tangible, Embed. embodied Interact. - TEI '10*, 2010.

[12] M. Resnick and B. Silverman, "Some Reflections on Designing Construction Kits for Kids," in *IDC 2005*, 2005.

[13] H. Raffle, A. Parkes, and H. Ishii, "Topobo: a constructive assembly system with kinetic memory," *Proc. SIGCHI Conf. ...*, 2004.

[14] E. Schweikardt and M. D. Gross, "roBlocks: A Robotic Construction Kit for Mathematics and Science Education," in *ICMI '06*, 2006, pp. 6–9.

[15] J. Yu and R. Roque, "A Survey of Computational Kits for Young Children," in *IDC 2018*, 2018, pp. 289–299.

[16] T. Lauwers and I. Nourbakhsh, "Designing the Finch: Creating a Robot Aligned to Computer Science Concepts," *Proc. Twenty-Fourth AAAI Conf. Artif. Intell.*, 2010.

[17] F. Hu, A. Zekelman, M. Horn, and F. Judd, "Strawbies: Explorations in Tangible Programming," in *Proceedings of the 14th International Conference on Interaction Design and Children*, 2015, pp. 410–413.

[18] A. Sullivan, M. Elkin, and M. U. Bers, "KIBO Robot Demo: Engaging Young Children in Programming and Engineering," *Proc. 14th Int. Conf. Interact. Des. Child. - IDC '15*, 2015.

[19] S. Papert, "Mindstorms: Computers, children, and powerful ideas," *NY Basic Books*, 1980.

[20] B. Y. D. Bau, J. Gray, C. Kelleher, J. Sheldon, and F. Turbak, "Learnable Programming: Blocks and Beyond," *Commun. ACM*, vol. 60, no. 6, pp. 72–80, 2017.

[21] M. Resnick *et al.*, "Scratch: Programming for All," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[22] S. Cooper, W. Dann, and R. Pausch, "ALICE: A 3-D Tool for Introductory Programming Concepts," in *CCSC '00 Proceedings of the fifth annual CCSC northeastern conference on The journal of computing in small colleges*, 2000, vol. 5, pp. 108–117.

[23] N. Fraser, "Ten Things We've Learned from Blockly," in *IEEE Blocks and Beyond Workshop*, 2015, pp. 49–50.

[24] J. Goode and J. Margolis, "Exploring Computer Science: A Case Study of School Reform," *ACM Trans. Comput. Educ.*, vol. 11, no. 2, pp. 1–16, 2011.

[25] Code.org, "Teach Computer Science," 2019. [Online]. Available: https://studio.code.org/courses?view=teacher. [Accessed: 04-Aug-2019].

[26] C. Duncan, T. Bell, and S. Tanimoto, "Should your 8-year-old learn coding?," in *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, 2014, pp. 60–69.

[27] D. Weintrop and U. Wilensky, "To Block or not to Block , That is the Question: Students' Perceptions of Blocks-based Programming," in *IDC '15*, 2015, pp. 199–208.

[28] T. W. Price and T. Barnes, "Comparing Textual and Block Interfaces in a Novice Programming Environment," in *ICER'15*, 2015.

[29] D. Weintrop and U. R. I. Wilensky, "Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, pp. 1–25, 2017.

[30] D. Weintrop, H. Killen, and B. Franke, "Blocks or Text? How Programming Language Modality Makes a Difference in Assessing Underrepresented Populations," in *ICLS 2018*, 2018, pp. 328–335.

[31] S. Dasgupta, S. M. Clements, Y. Abdulrahman, C. Willis-ford, and M. Resnick, "Extending Scratch: New Pathways into Programming," in *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2015.

[32] P. Rein, S. Ramson, J. Lincke, R. Hirschfeld, and T. Pape, "Exploratory and Live, Programming and Coding: A Literature Study Comparing Perspectives on Liveness," *Art, Sci. Eng. Program.*, vol. 3, no. 1, 2019.

[33] J. H. Maloney and R. B. Smith, "Directness and Liveness in the Morphic User Interface Construction Environment," in *UIST '95*, 1995, pp. 21–28.

[34] I. E. Sutherland, "Sketch Pad a Man-machine Graphical Communication System," in *Proceedings of the SHARE Design Automation Workshop*, 1964, p. 6.329--6.346.

[35] J. Kubelka, R. Robbes, and A. Bergel, "The Road to Live Programming: Insights From the Practice," in *ICSE*, 2018.

[36] N. Collins, A. McLean, J. Rohrhuber, and A. Ward, "Live coding in laptop performance," *Organised Sound*, vol. 8, no. 3, 2003.

[37] A. F. Blackwell and N. Collins, "The Programming Language as a Musical Instrument," in *Psychology of Programming Languages Interest Group*, 2005.

[38] C. M. Hancock, "Real-Time Programming and the Big Ideas of Computational Literacy," 2003.

[39] S. L. Tanimoto, "VIVA: A visual language for image processing," *J. Vis. Lang. Comput.*, vol. 1, no. 2, pp. 127–139, 1990.

[40] D. W. Sandberg, "Smalltalk and exploratory programming," *ACM SIGPLAN Not.*, vol. 23, no. 10, pp. 85–92, 1988.

[41] S. L. Tanimoto, "A Perspective on the Evolution of Live Programming," in *LIVE '13 Proceedings of the 1st International Workshop on Live Programming*, 2013, pp. 31–34.

[42] S. Burckhardt *et al.*, "It's Alive! Continuous Feedback in UI Programming," in *PLDI '13*, 2013, pp. 95–104.

[43] N. Collins, A. Mclean, J. Rohrhuber, and A. Ward, "Live coding in laptop performance," *Organised Sound*, vol. 8, no. 3, pp. 321–329, 2003.

[44] S. Mcdirmid, "Living it up with a Live Programming Language," in *OOPSLA*, 2007, pp. 623–637.

[45] H. Tsukamoto, Y. Takemura, H. Nagumo, I. Ikeda, A. Monden, and K. Matsumoto, "Programming Education for Primary Schoolchildren Using a Textual Programming Language," in *2015 IEEE Frontiers in Education Conference (FIE)*, 2015, pp. 1–7.

[46] J. P. Kramer, J. Kurz, T. Karrer, and J. Borchers, "How live coding affects developers' coding behavior," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2014, pp. 5–8.

[47] B. Victor, "Learnable Programming," 2012. [Online]. Available: http://worrydream.com/LearnableProgramming/.

[48] C. D. Hundhausen and J. L. Brown, "An experimental study of the impact of visual semantic feedback on novice programming," *J. Vis. Lang. Comput.*, vol. 18, no. 6, pp. 537–559, 2007.

[49] J. Wakefield, "BBC gives children mini-computers in Make it Digital scheme," *BBC:News*, 2015. [Online]. Available: https://www.bbc.com/news/technology-31834927.

[50] S. Sentance, J. Waite, S. Hodges, E. Macleod, and L. E. Yeomans, "'Creating cool stuff' – Pupils' experience of the BBC micro:bit," in *Proceedings of the 48th ACM Technical Symposium on Computer Science Education: SIGCSE 2017*, 2017.

[51] J. Curtis and C. Hopping, "Foundation aims to ship 100 million micro:bits," *ITPRO*, 2016. .

[52] BBC, "Two million BBC micro:bits distributed globally," 2018. [Online]. Available: https://www.bbc.co.uk/mediacentre/latestnews/2018/two-million-bbc-micro-bits-distributed-globally.

[53] J. H. Maloney, M. Nagle, and J. Mönig, "GP: A General Purpose Blocks-Based Language.," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 2017, pp. 739–739.

[54] B. Harvey and J. Mönig, "Bringing 'no ceiling' to Scratch: Can one language serve kids and computer scientists?," in *Proceedings of Constructionism 2010 Conference*, 2010, pp. 1–10.

[55] Authors, "Authors." 2019.

[56] T. Ball, J. Bishop, and J. Finney, "Multi-Platform Computing for Physical Devices via MakeCode and CODAL," in *ACM/IEEE 40th International Conference on Software Engineering: Companion Proceedings Multi-Platform*, 2018, pp. 552–553.

[57] R. C. Bogdan and S. K. Biklen, "Qualitative Data Analysis and Interpretation," in *Qualitative Research in Education: An Introduction to Theories and Methods*, 5th Editio., Pearson, 2007, p. 336.

[58] A. Stefik and S. Siebert, "An Empirical Investigation into Programming Language Syntax," *ACM Trans. Comput. Educ.*, 2013.

[59] H. Lieberman and C. Fry, "Bridging the gulf between code and behavior in programming," *Proc. SIGCHI Conf. Hum. factors Comput. Syst. - CHI '95*, 1995.

[60] D. A. Fields, Y. B. Kafai, T. Nakajima, and J. Goode, "Teaching Practices for Making E-Textiles in High School Computing Classrooms," in *Proceedings of the 7th Annual Conference on Creativity and Fabrication in Education*, 2017, p. 5:1--5:8.