Journal of Computer Languages 67 (2021) 101075

Contents lists available at ScienceDirect

Journal of Computer Languages

journal homepage: www.elsevier.com/locate/cola

The landscape of Block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming

Yuhan Lin *, David Weintrop

University of Maryland, College Park, MD, USA

ARTICLE INFO

Keywords:
Computer science education
Block-based programming
Design of programming environments

ABSTRACT

Block-based programming (BBP) environments have become increasingly commonplace computer science education. Despite a rapidly expanding ecosystem of BBP environments, text-based languages remain the dominant programming paradigm, motivating the transition from BBP to text-based programming (TBP). Support students in transitioning from BBP to TBP is an important and open design question. This work identifies 101 unique BBP environments, analyzes the 46 of them and identifies different design approaches used to support the transition to TBP. The contribution of this work is to provide a snapshot of the current state of BBP environments and how they support learners in transitioning to TBP.

1. Introduction

Computing, and the technologies it enables, is reshaping the world. From how we work and learn, to how we play and socialize, few aspects of our lives have been unaffected by the long reach of technology. Given this growing presence in our lives, providing opportunities and tools to help people understand how technologies work and empowering them to control those technologies is becoming a growing focus of computing education efforts. This can be seen in formal education with the rise of the Computer Science for All movement [1] as well as in informal contexts, where a growing number of toys, games, and out-of-school learning opportunities focused on coding and computational thinking have emerged [2]. Across these contexts, block-based programming (BBP) is increasingly the way that novices are being introduced to the practice of programming and the field of computer science more broadly [3,4].

Research investigating the ease-of-use, accessibility, and effectiveness of BBP environments is finding the approach to be an effective way for novices to have early programming successes. In this way, BBP is succeeding in providing a low-threshold entry to the practice of programming, which was a central goal for BBP environments [3,5]. These findings, along with the growing popularity of BBP environments like Scratch, has resulted in a blossoming of BBP environments across a wide range of contexts, including robotics toolkits [6], e-textiles [7], data science [8], industrial robotics [9], and mobile app development [10]. At the same time, BBP has a growing presence in formal education contexts, with several influential and widely adopted computer science curricula teaching with BBP environments, including Exploring Computer Science [11], Scratch Encore [12], the Beauty and Joy of Computing [13], and the suite of materials developed and shared by code.org [14]. Given this rapidly expanding ecosystem of BBP environments, it is important to understand the current state of the design of BBP environments to make sense of where the field is and layout a roadmap of potential opportunities for innovation and advancement in the field. Further, taking stock of the current landscape can provide useful guidance for educators, parents, children, and anyone else starting out in the increasingly busy world of BBP.

While BBP can provide an accessible and engaging introduction to programming, in both formal and informal contexts, there are reasons learners may want to learn to program with a text-based programming (TBP) language, such as Java, Python, or JavaScript. For example, when looking at robotics toolkits, TBP languages frequently offer learners access to additional capabilities beyond what BBP interfaces provide, including advanced programming libraries that grant greater control over devices. Additionally, some more sophisticated devices do not provide BBP support. In formal educational contexts, the transition from block-based to TBP will happen if learners choose to pursue more advanced computer science instruction as BBP are often used as introductory tools with the goal being to transition learners to more professional text-based languages [15]. It is important to state this work is not arguing that such a transition is necessary, or even desirable, for all learners as we support a vision of many desirable endpoints for computing instruction [16,17]. Nevertheless, many learners will

Abbreviations: BBP, Block-based Programming; TBP, Text-based Programming

* Corresponding author.
E-mail addresses: jimmylin@umd.edu (Y. Lin), weintrop@umd.edu (D. Weintrop).

https://doi.org/10.1016/j.cola.2021.101075
Received 1 June 2021; Received in revised form 28 September 2021; Accepted 27 October 2021
Available online 4 November 2021
2590-1184/© 2021 Elsevier Ltd. All rights reserved.

transition from BBP to TBP, and thus, it is important we understand how to best support this transition as research has found this transition can be a source of difficulty for learners [18–20].

To date, numerous design approaches have been used to scaffold learners in making the transition from BBP to TBP but there has yet to be a systematic review of these approaches or a detailed analysis of design characteristics of if and how BBP environments are supporting learners in making this transition. In this paper, we seek to address this gap in the literature and respond to the call made by McGill & Decker [21] to more fully understand the landscape of computing education tools to best support educators, designers, and learners. In doing so, we provide a snapshot of the current state of the field, identifying characteristics of commonly used approaches as well as potential design opportunities for moving the field forward. More concretely, this work seeks to answer the following research questions:

RQ1: What is the current state of the design of block-based programming environments?

RQ2: What design approaches are currently being used to support learners in transitioning from block-based to text-based programming?

To answer these questions, we conducted a systematic literature review to identify the universe of programming environments that utilize a BBP approach. Using the results from this search, we analyzed and categorized the environments to understand how each approached the relationship between block-based and text-based programming. The contribution of this work is to take stock of the available set of BBP environments and to advance our understanding of the current approaches being used to bridge block-based and text-based programming. In doing so, we shed light on the form and nature of interactions being designed for students to support them with early programming experiences and scaffold them along their path towards more fully participating in computing cultures.

2. Prior work

Given the continually evolving nature of the field of human–computer interaction and the emergence of new tools and techniques for introducing technologies and computing to learners, researchers working in this area are constantly trying to take stock of the current state of the field [22–24]. Looking specifically at the growing ecosystem of tools and environments designed to introduce young learners to programming, McGill and Decker [25] map out the various approaches used and propose a taxonomy of tools, languages, and environments. As part of this taxonomy, they categorize environments by purpose (e.g., data manipulation, digital media) and provide a coarse breakdown of interaction approach: block-based symbolic, block-based text, hybrid, and text-based. In the work presented below, we further refine this ontology, more carefully articulating characteristics of interaction with a particular focus on those environments that straddle the block-based and text-based divide. Below we review research on BBP generally and then review research specifically focused on the blocks-to-text transition before presented the methods and results of this work.

Before continuing, it is important to acknowledge what can and cannot be claimed when speaking generally at the level of BBP environments and TBP languages due to the tremendous variation with both categories. Looking at TBP languages, we acknowledge it is potentially problematic to lump together languages like Python, Lisp, Java, and Haskell under the same label for analytic purposes given the difference in notional machine models, paradigms, and technical features. Similarly, the differences between Scratch, App Inventor, and AgentCubes are significant enough that one must be cautious when making claims about the BBP environments in aggregate. For this work, we draw on Weintrop & Wilensky's notion of modality to ground our decision to classify language and environments at the admittedly coarse level of block-based and text-based. "Given a semantics, we use the term modality to capture how one interacts with and composes within that semantics... In looking at the interactions enabled by the presentation of a semantics, modality is not a characteristic of a representational system alone but also captures the relationship between the representation, its interface, and how one uses it" [26]:84. From this perspective, we think it is reasonable to analyze TBP languages and BBP environments as two distinct modalities that share common presentation and interaction features, which is useful when bringing a learner-interaction, user-experience lens. Thus, in this work, we are not considering things such as mental models, expressive power, or syntactic features when considering BBP environments or TBP languages, but instead, focused on the experience of a learner using the environment to shape our analytic lens.

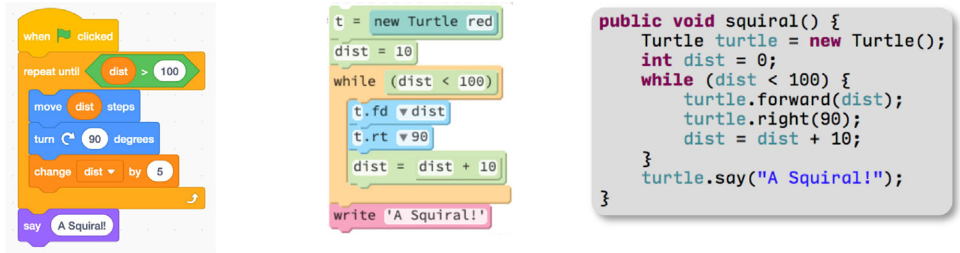
2.1. Block-based programming

BBP is an increasingly widespread approach to programming that uses a programming-command-as-puzzle-piece metaphor to help students understand how and where commands can be used [27] (Fig. 1a, 1b). To author programs in a BBP environment, users use a drag-and-drop interaction to snap program phrases together. Visual cues in the form of the shape and color of the programming phrase provide guidance on how they can be assembled. If two programming phrases cannot be joined to form a valid sequence, the environment prevents them from snapping together, thus preventing syntax errors while authoring programs. This authoring approach is driven by the abstract syntax tree of the language, creating an interface where the underlying structure of the program is made visible to the author as they are constructing their program [28]. BBP has become widespread in recent years, in part due to the successes of environments like Scratch [5], Snap! [29], Alice/Looking Glass [24], App Inventor [30], AgentSheets/AgentCubes [31], Blockly [32], Code.org's AppLab [14], and Microsoft Makecode [6]. Despite the recent growth in the approach, BBP is not a recent innovation. For example, Blox [33] developed in the mid-1980s, and LogoBlocks [34], developed in the mid-1990s, included many of the visual cues and interaction patterns found in contemporary BBP environments while AgentSheets, and its successor AgentCubes, have been in active use since the mid-90s [35]. Further, BBP environments draw from earlier work from the 1980s on structured editors that use information about the syntax of a language to inform how programs can be authored [36,37].

An analysis of learners' perceptions of BBP reported several features identified by learners that contribute to the overall ease-of-use of BBP environments, including the visual cues for how and where blocks can be used, the ability to browse available programming phrases, the drag-and-drop composition mechanism, and the fact that the text on blocks is more readable [38]. Research on the effectiveness of BBP has found it to be successful at providing positive early programming experiences for learners and an effective way to introduce learners to the practice of programming [39–43].

The widespread use of BBP environments can partially be explained by how well the approach can achieve all four aspects of Burke and Kafai's [44] framework: low floors, high ceilings, wide walls, and open windows. The result is an inviting programming environment enabling a wide array of uses that can support novices as they pursue personally meaningful projects. At the same time, BBP environments like Scratch have successfully cultivated a vibrant user base and online community, making programming a more social activity [45,46]. As a result of these successes, Scratch and other BBP environments are increasingly becoming a part of formal computer science education in K-12 classrooms (e.g. [12,47]) and beyond (e.g. [48,49]). While BBP is seeing widespread adoption, some students see the drawbacks in BBP, such as it being less powerful, less suitable for larger projects, or less authentic to the conventional TBP [38].

In addition to being used in more conventional computer science learning contexts, there is a wide variety of usage domains for BBP. BBP environments have been developed for mobile app development [30,



(a) (b) (c)

Fig. 1. Two sample block-based programs, written in (a) Scratch [5] and (b) Pencil Code, and (c) a comparable program written in Java.

50], developing and playing videogames [51–53], database querying [22,54,55], physical computing [6], scientific modeling and simulation [56], and industrial robotics [57]. Further, a growing number of BBP environments are being developed for specific Integrated development environments (IDEs) including ArduBlockly [58] for Arduino and Amphibian [59] for IntelliJ. Much of this innovation can be attributed to the introduction of flexible and extensible BBP libraries like Blockly [32] that make it easy to design and embed BBP into existing programming environments. Likewise, the open-source and collaborative community that has grown around BBP environments, supported by the flexibility of programming approach, has resulted in many extensions being made to BBP environments beyond what was initially supported (e.g., [25,60]).

2.2. Transition from block-to-text

One question increasingly discussed in the literature is if and how BBP supports the transition to TBP [18,20]. As shown in Fig. 1, the visual presentation of text-based programs differs significantly from block-based programs, looking more like a highlighted text document than the colorful, jigsaw presentation of a block-based program. Helping learners identify the similarities between the two forms of programming so that knowledge gained and confidence built in BBP environments carries over to TBP contexts is an active area of investigation. Armoni et al. [61] explored this question by looking at students using block-based tools in middle grades and then progressing to text-based languages in high school. The result showed little quantitative difference in their performances on assessments but students who had Scratch experience showed higher performance than their peers in some specific content areas (e.g. iterating programming constructs) [61]. Grover and colleagues [62] showed that a block-based curriculum can prepare learners for TBP. Weintrop & Wilensky conducted a quasi-experimental study in two high school computer science classrooms, one using a BBP environment and the other using an isomorphic TBP environment. In analyzing students' performance on content assessments, they found that while students learning in a block-based condition performed better after an introductory period [43], after the transition to TBP, there was no significant difference between the two conditions [20]. In trying to understand the complexities associated with this transition, Kölling et al. [18] identified 13 distinct issues related to this transition, ranging from syntax memorization and readability of the code to managing layout and understanding error messages. This research suggests that supporting the transition is possible, but open questions remain as to how best to support learners in doing so.

To support the transition from block-based to text-based programming, a growing number of programming environments are exploring ways to blend the two approaches or support novices in moving back-and-forth between the two. Reviewing these environments and the design strategies employed to support this transition is the central focus of the second research question this paper seeks to answer and is explored in greater detail in the Findings section after reviewing the methods used to identify BBP environments.

3. Methodological approach

To answer the stated research questions, a systematic review was conducted to identify the current state of the BBP ecosystem. Our approach for compiling an exhaustive list of BBP environments began with a review of BBP environments introduced and discussed in the academic literature. The analysis began with a keyword search of "block-based programming" in the Association for Computing Machinery (ACM) and Institute of Electrical and Electronics Engineers (IEEE) digital libraries, resulting in 360 and 415 unique results respectively. These results included full journal articles, conference papers, as well as conference abstracts, and included articles that were not centrally focused on BBP or a specific BBP environment but just included references to the BBP approach. We chose this strategy as the goal of this work is to identify and examine the breadth of BBP programming environments that exist. This is in contrast to a review focused on research findings specific to the user of BBP environments. This strategy also yielded several BBP environments developed outside of academia and only referenced in the academic literature, thus expanding the scope beyond BBP environments developed within academia.

To continue to expand the list of BBP environments, we next reviewed the entirety of proceedings from three venues related to visual programming: IEEE's Blocks and Beyond workshop, the BLOCKS+ workshop at SPLASH 2018, and the IEEE Symposium on Visual Languages and Human-Centric Computing *(VL/HCC)*. Every article from these sources was reviewed to identify if a BBP environment was included as part of the research. Finally, we expanded the emerging list of BBP environments to include environments recommended by colleagues and reviewers that had not yet been identified by our systematic search.

In total, 101 unique BBP environments were identified. Of those 101 BBP environments, our analysis focuses on the 46 of these environments that we were able to run and interact with, thus allowing for a full evaluation of the environment and its capabilities. BBP environments identified in the literature were not included in the focal set of 46 if the environment could not be located online, if it was behind a paywall, if it required an accompanying physical device (e.g., a specific robotics toolkit), if only the source code was available and required the user to compile and/or host the environment, or if the environment had not been updated in 5 years or was too old to run on macOS version 11+. Disagreements on the inclusion/exclusion of a given environment were resolved through discussion and collaborative analysis based on inclusion and exclusion criteria for the review.

For the full set of 101 BBP environments identified, 30 were identified in the Blocks & Beyond workshops, 21 in the proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing *(VL/HCC)*, 8 BBP environments were introduced in the proceedings of Interaction Design and Children (IDC), and 6 were founded in the proceedings of the ACM Conference on Human Factors in Computing Systems (CHI). No other venue contributed more than 5 BBP environments. A majority of BBP environments were published in 2015 or after. This rise, along, with the high number of new environments introduced in 2017 (27 BBP environments) and 2019 (18 BBP environments) coincides with the convening of the Blocks & Beyond workshop, which has met every other year since 2015. This suggests that the Blocks & Beyond workshop has been a central venue for publicizing BBP environments and sharing design innovations in the space. The complete list of the 101 BBP environments, along with brief descriptions and references is presented in Appendix.

We analyzed the identified BBP environments by employing a structural coding approach [63] to both code and categorize BBP environments. We first systematically classified the BBP environments along a number of dimensions, including focal domain, runtime environment, and relationship to TBP. Identifying the focal domains was done through a process of descriptive coding followed by axial coding [63] to identify a robust set of domains that captured the breadth of the environment identified. A spreadsheet was used to record and organize the results of the coding process. Next, we examined each environment to record their existing block categories and define the capabilities of the environment. This coding step was necessary as different environments use different terms for the same capabilities. The first author completed an initial coding pass, after which the second author conducted a second analytic pass to ensure the codes could be reliably employed and to identify any potential discrepancies in the analysis. Any discrepancies found were resolved through discussion.

4. Findings

This section presents the results of our analysis. We first share a summary of the 46 environments we analyzed, followed by a presentation of the capabilities of the environments and the domains they focus on. We then shift to the second research question looking at the various approaches used by BBP environments to support learners transitioning to TBP.

4.1. The current state of block-based programming environments

Our review of the academic literature on the design of BBP environments identified 101 unique environments (Appendix 1). Of those 101 BBP environments, 46 were accessible to the authors for deeper analysis. Those environments, along with a classification of the domain the environment is designed for, the runtime environment for the environment, the number of block categories it has, and its relationship to TBP are shown in Table 1.

A few things stand out when looking across the set of 46 environments that were analyzed. The first is the sheer number of environments. When listed, the challenge educators, parents, kids, and programming novices in general face when they decide what BBP to use is clear. There are a lot of environments to choose between, making it potentially difficult to find the best environments without a non-trivial amount of research. A second pattern that stands out in this table is the prevalence of browser-based programming environments. All but five of the analyzed BBP environments were designed to run within a web browser. This makes sense for a number of reasons identified in the literature, including addressing issues of device/OS compatibility, the ease of updating the environment, and the potential to pull in other web-based resources [64]. A final thing that stands out when looking across this set of 46 BBP environments is the variety that exists within the design space. The next two sections explore this diversity as it relates to the capabilities of these environments and the domains they are designed for.

4.1.1. The capabilities and organization of block-based programming environments

The defining feature of BBP environments is the presence of blocks as the mechanism through which programs are authored. By analyzing the set of blocks provided by a given environment and the way they are organized, we gain insight into the capabilities of that environment and the priorities of its designers. Engaging in this exercise across all currently available BBP environments yields insights into the state of the field. BBP environments thematically organize blocks into categories (sometimes called "block palettes" or "drawers"), which can be seen in the red square on the left side of the images in Figs. 2a–2d. The first way to understand the capabilities of BBP environments is to understand how they organize their blocks both in terms of the number of categories as well as the conceptual groupings used. Of the 46 BBP environments we examined, 36 included block categories that could be analyzed. For the 10 BBP environments that were not included in this analysis, there were either no categories presented (i.e., all available blocks were grouped into a single unifying category) or the environment did not rely on a block palette (i.e., relied on keyboard entry or used a dynamic blocks interface based on cursor context). These latter environments will be discussed in greater detail later when we introduce Hybrid BBP environments.

For the 36 BBP environments that we analyzed the block categories, we found the environments had an average of 9.2 block categories (SD 4.31) with the number of categories ranging from 2 up to 24. For environments with a wide range of capabilities, and thus a large number of categories, additional organization was sometimes added. For example, some environments like ModKit (Fig. 2c) use nested block categories to not overwhelm users with too many block categories. Other BBP environments, such as by OzoBlockly (Fig. 2d) present "levels" to the user, where higher levels are for more advanced users, and thus, more blocks categories are available. These approaches are intended to help the environment have a low floor (i.e. few blocks for novices) and high ceiling (i.e., lots of options and advanced blocks for more experienced users). Another strategy used for organizing blocks is to group the most common blocks into a single category, thus giving novice users a clear place to start. The MakeCode environment uses this approach by having its top category called "Basic".

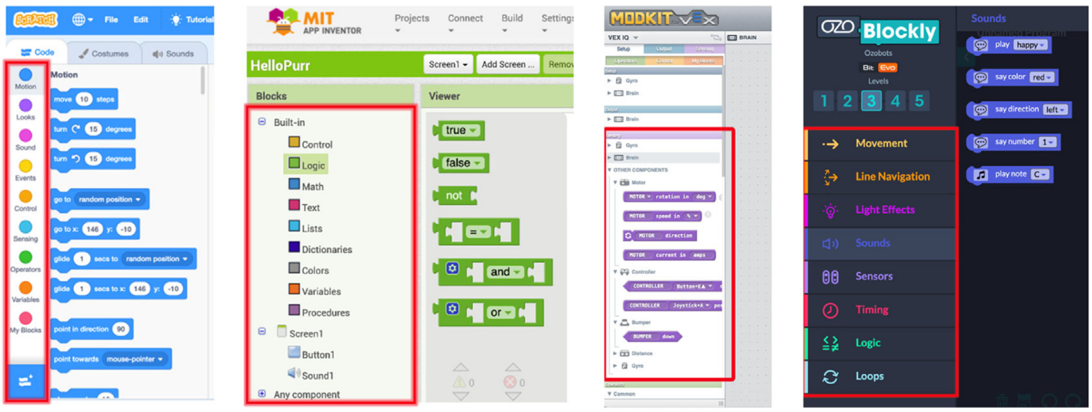
To understand the capabilities of the BBP environments, we qualitatively analyzed the blocks categories resulting in 14 Category Themes. Table 2 shows the results of this analysis in the form of a list of the Category Themes, a description of the blocks within that Category Theme, and the number of BBP environments that had a category that matched the Category Theme across the 36 block-based environments analyzed. While most block categories mapped to a single Category Type, some environments have categories that included blocks from multiple themes. For example, the Blocks4DS environment has a category called "Input and Output" that includes blocks that align to both the "Output" and the "Sense/Input" Category Themes. At the same time, some environments had multiple categories that aligned to a single Category Theme, such as GP, which has separate categories for "Variables", "Words", "Data", and "Structure", all of which map to the Category Theme of "Variables/Data Structures".

4.1.2. Domain

While many TBP languages are designed to be general purpose, it is common for BBP environments to be designed for a specific domain, be it games & simulations, multimedia (e.g. animations and storytelling), or controlling physical computing devices or robots. For the 46 environments that we were able to run on our local computers, we qualitatively coded the environments based on the types of programs they can support or a stated focal domain area for the environment (as reported by the authors). Like with the analysis of the block categories, it was possible for an environment to support multiple domains. For example, GameBlox has commands for creating games as well as data science tasks. Of the 7 domains identified, the most popular was games & simulations($n = 14$), followed by data sciences ($n = 11$), physical computing ($n = 10$), and multimedia ($n = 10$). There were also several BBP environments designed to provide a block-based interface to a general-purpose TBP language such as BlockPy, which is a block-based interface for Python. Table 3 provides a full breakdown of environments by domain. One BBP environment could be in multiple categories if it supported multiple domains, such as Scratch being designed to support both Games and Multimedia projects.

Table 1
An analysis of the 46 BBP environments that were identified and analyzed.

BBP Environment	Domain	Runtime Environment	Number of Block Categories	Relationship to TBP
AgentCubesOnline	Games & Simulations	Browser	12	Blocks-only
Alice	Multimedia	Win/Mac	Varies by Context	One-way Transition Read-only
APPInventor	APP	Browser	9	Blocks-only
ArduBlockly	Physical computing	Browser/Win/Mac	11	One-way Transition Read-only
BeetleBlocks	Games & Simulations	Browser	8	Blocks-only
Blockly	Physical computing/Data Science	Browser	8	One-way Transition Read-only
Blockly@rduino	Physical computing	Browser	7	One-way Transition Read-only
BlocklySQL	Data science	Browser	5	Dual-modality Distinct-View
BlockPy	General/Data science	Browser	12	Dual-modality Shared-View
Blocks4DS	Data science	Browser	7	Blocks-only
Bricklayer-lite	Tasks	Browser	6	One-way Transition Read-only
Calypso	Physical Computing	Browser/Win/Mac	Not Applicable	Blocks-only
Catroid/Catrobat/PocketCode	Games/Multimedia	iOS/Android	7	Blocks-only
code.org/hourofcode	Games & Simulations/Multimedia/App	Browser	13	Dual-modality Distinct-View
DBSnap	Data science	Browser	2	One-way Transition Read-only
DragonArchitect	Games/Task	Browser	Not Applicable	Blocks-only
Dronely	Physical Computing	Browser	9	One-way Transition Read-only
Droplet	General	Browser	6	Dual-modality Distinct-View
DwengoBlocks	Physical computing	Browser	6	One-way Transition Read-only
Frame-basedediting	General	Win/Mac	Not Applicable	Hybrid
GameBlox	Games & Simulations/Data science	Browser	24	Blocks-only
GP	General/Data science	Browser/Win/Mac	16	Blocks-only
Grasshopper	Tasks	Browser/iOS/Android	Not Applicable	Hybrid
iSnap	Games & Simulations/Data science	Browser	8	Blocks-only
KoDu	Games & Simulations	Win	Not Applicable	Blocks-only
Kokopelli'sWorld	Tasks	Browser	Not Applicable	Blocks-only
LookingGlass	Multimedia	Win/Mac	Varies by Context	Blocks-only
Makecode	Physical computing	Browser	18	Dual-modality Distinct-View
mBlock	Physical computing/Multimedia/Data Science	Browser/Win/Mac/iOS/Android	9	One-way Transition Read-only
Modkit	Physical computing	Browser/Win/Mac	6	Blocks-only
NetsBlox	Games & Simulations/Data science	Browser	10	Blocks-only
ozoblockly	Physical computing	Browser	15	One-way Transition Read-only
Pencil.CC	General/Multimedia	Browser	8	Hybrid
PencilCode	General/Multimedia	Browser	8	Dual-modality Distinct-View
Poliglot	General	Browser	5	Dual-modality Shared-View
Reduct	Task	Browser	Not Applicable	Blocks-only
Robomise	Task	Browser	Varies by Level	Blocks-only
Scratch	Games & Simulations/Multimedia	Browser/Win/Mac/Android	9	Blocks-only
ScratchJr	Games & Simulations/Multimedia	iOS/Android	5	Blocks-only
SparqlBlocks	Data science	Browser	10	Blocks-only
Snap!	Games & Simulations/Data science	Browser	8	Blocks-only
Sonification	Multimedia	Browser	6	Blocks-only
StarLogoNova	Games & Simulations	Browser	14	Blocks-only
TiledGrace/MiniGrace	General	Browser	Varies by Context	Dual-modality Distinct-View
TinkerCAD	Multimedia	Browser	5	Editable One-way Transition
VEXcode	Physical computing	Browser/Win/Mac/iOS/Android	10	Editable One-way Transition



(a) (b) (c) (d)

Fig. 2. The block palettes for (a) Scratch, (b) MIT App Inventor, (c) MODKit, and (d) OzoBlockly.

Table 2
Block categories identified in BBP environments and their frequency.

Category Theme	Definition	# of Categories (# of Envs)
Variables/Data structure	Creates custom variables, lists, arrays, or other data structures	60 (31)
Logic operators	Conventional programming primitives such as loops, conditionals, and controlling program execution	59 (33)
Domain-specific	Domain-specific blocks for hardware or domain	33 (15)
Sprite/Character Appearance	Controls the appearance of the sprite/physical item	32 (23)
Operators (numerical, textual, color)	Mathematical operators, Data manipulation, string manipulation, color manipulation	31 (29)
Customize Block	Custom functions with multiple blocks	23 (23)
Movement	Moves the main character/physical item	22 (19)
Sound	Creates or controls sound	19 (18)
Sense/Input	Senses state change for the sprite/physical item or prompts the user for input	19 (17)
Interacting with the environment (physical or virtual)	Controls visual aspects beyond the character/physical item (e.g. drawing with a pen)	18 (11)
Output	Output as text, histogram, or other external presentation	4 (4)
Debug	Debug function	4 (4)
Comment	Comments on the code	3 (3)
Extension	Add extension outside of current BBP environment	2 (2)

Table 3
BBP environments domains.

Domains	Count	Block-based Programming Environments
Games & Simulations	14	AgentCubes Online, BeetleBlocks, Catroid/Catrobat/Pocket Code, code.org, Dragon Architect, GameBlox, iSnap, KoDu, NetsBlox, Scratch, Scratch Jr, Snap!, StarLogo Nova
Data Science	11	BlocklySQL, Blockly, BlockPy, Blocks4DS, DBSnap, GameBlox, GP, iSnap, mBlock, NetsBlox, Snap!
Physical Computing	10	Blockly @rduino, ArduBlockly, Calypso, Blockly, Dronely, Dwengo Blocks, Makecode, mBlock, Modkit, ozoblockly
Multimedia	10	Scratch, Alice, Pocket Code, code.org, Looking Glass, mBlock, Pencil Code, Pencil.CC, Scratch Jr, sonification
Block-based implementation of text-based language	7	Frame-based editing, BlockPy, Droplet, Frame-based editing, Pencil Code, Pencil.CC, Tiled Grace/Mini grace
Tasks	7	Bricklayer-lite, Dragon Architect, Grasshopper, Kokopelli's World, Quizly, Reduct, Robomise
Mobile Application development	2	App Inventor, code.org

4.2. *Transitioning from block-based to text-based programming*

Having provided a high-level picture of the current state of BBP environments and their capabilities, we now shift our focus to an increasingly active area of research in the design of BBP environments: supporting the transition to TBP. Building upon definitions proposed in [65], we analyzed the 46 BBP environments based on four distinct approaches for supporting the block-to-text transition: *Blocks-only*, *One-way Transition*, *Dual-modality*, and *Hybrid*. Each category is further described in the following sections. Table 4 provides a summary of the categories and the number of BBP environments that employ each approach. A full mapping of the full list of BBP environments can be found in Appendix.

Table 4
The four approaches used to support the block-based to text-based transition.

Category	Text-Authoring Capability	Count	BBP Environments
Blocks-only	None	24	AgentCubes Online, APP Inventor, BeetleBlocks, Blocks4DS, Calypso, Catroid/Catrobat/Pocket Code, Dragon Architect, GameBlox, GP, iSnap, KoDu, Kokopelli's World, Looking Glass, Modkit, NetsBlox, Quizly, Reduct, Robomise, Scratch, Scratch Jr, SparqlBlocks, Snap!, Sonification, StarLogo Nova
One-way Transition	Read-Only	12	*Read-only*: ArduBlockly, Blockly, Blockly @rduino, Bricklayer-lite, DBSnap, Dronely, Dwengo Blocks, mBlock, ozoblockly *Editable*: Alice, TinkerCAD, VEXCode
Dual-modality	Full	8	*Shared-View*: BlockPy, Poliglot *Distinct-View*: BlocklySQL, code.org/hour of code, Droplet, Makecode, Pencil Code, Poliglot, Tiled Grace/Mini Grace
Hybrid	Combined with Block-editing	3	Frame-based editing, Grasshopper, Pencil.cc

4.2.1. Block-only

Blocks-only environments are BBP environments that do not provide any native support for transitioning to TBP. Slightly over half of the BBP environments found in the literature were classified as blocks-only environments. A canonical example of a blocks-only environment is Scratch (Fig. 3a) [5]. When working in Scratch, users only ever interact with a program in a block-based form. There is no mechanism native to the environment to allow learners to view or interact with a text-based version of their programs. In our analysis, 24 of the 46 BBP environments fell into this category, including Snap! (Fig. 3b), Scratch Jr, Pocket Code, and GP.

4.2.2. One-way transition

A One-way Transition environment is a BBP environment that allows the user to move from a block-based presentation of a program to a text-based presentation but does not give the learner the ability to turn a text-based program into blocks. In this way, it is as if the environment supports "exporting" block-based programs to text but does not provide integrated support beyond that. This approach has the benefit of making clear to the learner the relationship between block-based and text-based representations of code, while also retaining many of the benefits of authoring programs in BBP, such as removing syntax errors and providing a set of easily browsable commands. Within this category, we further broke down the BBP environments into two sub-categories: *Read-only* and *Editable*.

In Read-only environments, the learner can view a text-based version of the block-based program they have authored but cannot edit it. In this way, learners can see the relationship between BBP and TBP but are not able to do anything beyond seeing the correspondence. For example, many of Code.org's Hour of Code activities include a "Show code" button that reveals a text-based version of the learners' block-based program with the block-based program grayed out in the background (Fig. 4a). Along with Code.org's Hour of Code environment, other Read-only One-way transition BBP environments include ArduBlockly, Blockly, DBSnap, Dronely, and mBlock.

In Editable One-way Transition BBP environments, the user can convert their block-based program to a text-based form and then can continue working on the program in the text form. However, the environment does not support converting the text-based program back into blocks. This allows the user to edit their program in the text-based form, but once they export their program to a text-based language the two forms of the program are no longer linked. Prior research has found this one-way transition to be productive for learners. For example, researchers developed a plugin for a Java integrated development environment to allow learners to translate programs written in Alice 3 (a BBP environment) into Java, which helped learners make the transition from block-based to text-based programming [15]. A contemporary version of a One-way Transition Editable environment is the Vex VR editor (Fig. 4b) which allows the learner to see a text-based version of their block-based program and includes a "Convert to Text Project" button that allows the learner to continue the development of their program in Python. Along with Alice and Vex VR, TinkerCAD is a third example of an Editable One-way transition BBP environment.

4.2.3. Dual-modality

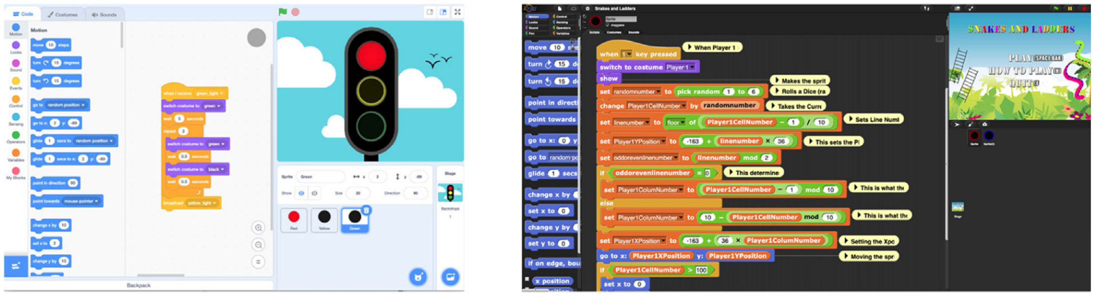
Dual-modality environments are environments that support both block-based and text-based authoring and give the learner the ability to move back-and-forth between the two. This allows learners to choose which modality they wish to work in with edits made in one modality carried over to the other (unlike one-way transition BBP environments). Research suggests that dual-modality programming environments can be engaging and effective for supporting students' transitioning from block to text programming [66–69]. We further break down the Dual-Modality environments into two sub-categories based on how the two modalities are presented to the learner: *Shared-View* and *Distinct-View*.

In Shared-View environments, the learner can see both block-based and text-based representations on the same page, usually presented side-by-side. BlockPy [70] and Poliglot [71] (Fig. 5) are two examples of Shared-View Dual-Modality environments as the block-based and text-based versions of the program are presented side-by-side. In both environments, both representations are editable with the editor keeping the two versions of the code in sync. In this way, the user can author either the block-based or text-based version of the program at any time with edits made in one interface being reflected in the other.

Distinct-View Dual-Modality environments allow learners to move back-and-forth between block-based and text-based presentations of the program, only showing the user one interface at a time. Often, the transition between the two modalities is accompanied by some form of animation that "morphs" one modality to the other. This transition helps reinforce the isomorphism of the two modalities (i.e., that block-based and text-based programs are equivalent). An early and influential version of this approach was BlockEditor, which supported block-based and text-based authoring of Java programs [68]. Several Distinct-View Dual-Modality environments are currently available for use, including the droplet editor [72] which is used by Pencil Code [16] and code.org's AppLab, GP [73], as well as the MakeCode environment [6] for programming physical computing devices (Fig. 6).

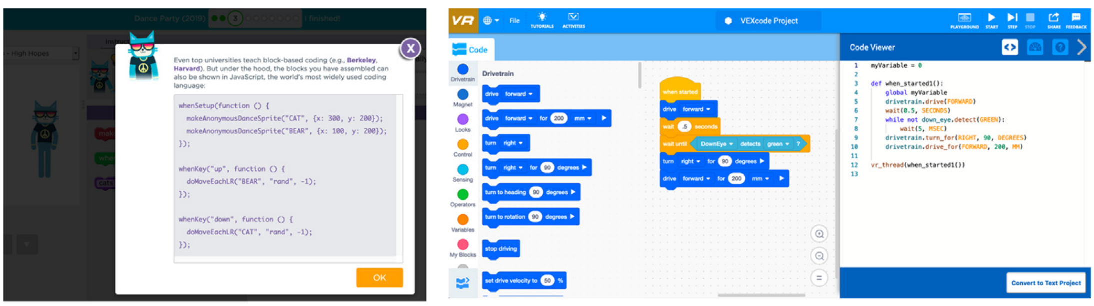
One challenge unique to Dual-Modality programming environments that support both block-based and text-based editing is the need to handle syntax errors. As discussed above, BBP environments are able to prevent invalid commands from assembling, making syntax error rare, and in some cases impossible. Thus, Dual-Modality environments that support text-based authoring need to handle the case when a syntax error is introduced in the text-based editor and then the user wants to convert their program to a block-based form. The Dual-Modality environments found in our analysis provide a number of different solutions to this design challenge.

One approach taken is to not allow the learner to convert a text-based program with a syntax error into the block-based modality. When this happens, environments take a few different ways to support



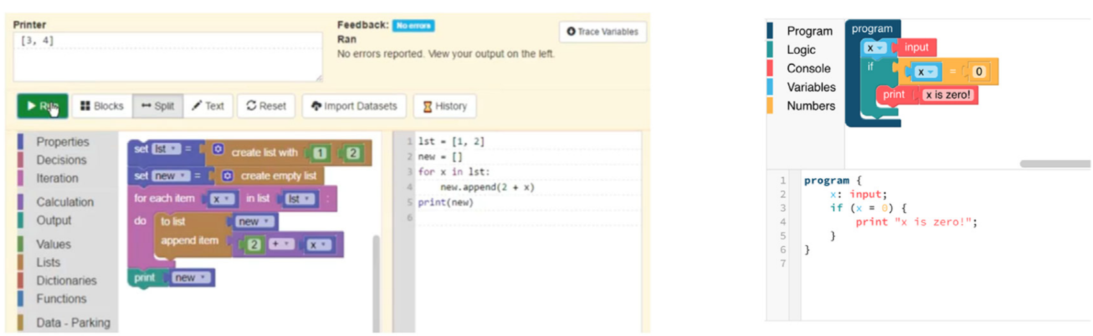
(a) (b)

Fig. 3. Scratch (a) and Snap! (b) — two Blocks-only environments.



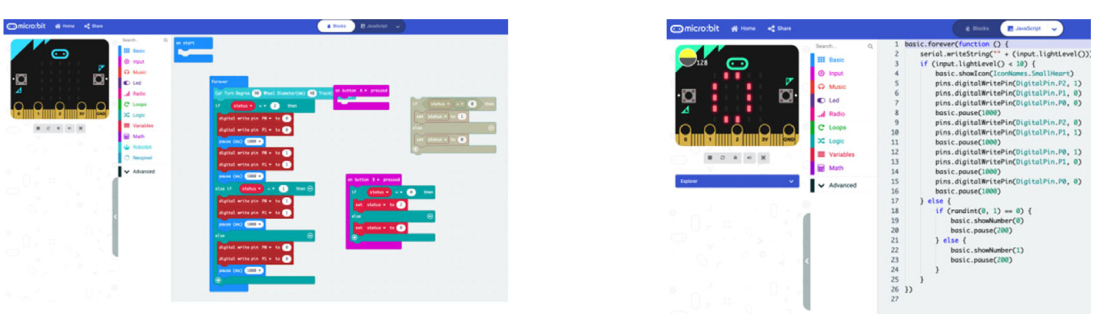
(a) (b)

Fig. 4. Two One-way Transition BBP environments. (a) Code.org's Hour of Code environment, a read-only BBP interface, and (b) Vex VR, an Editable One-way Transition BBP environment.



(a) (b)

Fig. 5. (a) BlockPy and (b) Poligot, Shared-View Dual-Modality environments.



(a) (b)

Fig. 6. MakeCode: a Distinct-View Dual-Modality programming environment that supports both (a) block-based and (b) text-based authoring.

the learner in moving forward. For example, in BlockPy [70], when a learner tries to convert a program with a syntax error to blocks, the environment displays a message saying it cannot complete the conversion and shows the syntax error that is present in the program (Fig. 7a). A second approach is for the environment to keep track of previous states of the program so that if the learner introduces an error, the environment gives the learner the option to discard the non-compiling code and proceed to the block-based interface, such as MakeCode (Fig. 7b). Another approach is for the environment to allow the transition to happen but then to isolate the syntax error into a single non-compiling block, this allows the learner to continue working in blocks, including the ability to drag the non-compiling block out of the program. These three design approaches make it possible for one environment to support both block-based and text-based editing.

4.2.4. Hybrid

The final category in this ontology of approaches used to support the block-based to text-based transition is Hybrid programming environments. Unlike the other categories that make distinctions between block-based and text-based presentations of programs, Hybrid environments blend features of the two approaches into a single interface, resulting in programming environments that have features of both modalities present at the same time.

For example, the Frame-based editing approach (Fig. 8a) was designed to have the low-threshold characteristics of block-based environments while also retaining the flexibility, expressiveness, and keyboard-driven of text-based tools [74]. A study of frame-based editing reported that learners find frame-based editing to be easy to use and were able to complete programming tasks more quickly than learners using a conventional introductory Java editor [75]. The idea of keyboard-driven block-based editing has also been explored in other environments, like the typeblocking in StarLogo Nova [76,77].

A second example of a Hybrid Programming Environment can be seen with Pencil.cc [65]. In Pencil.cc (Fig. 8b), the programmer is presented with a browsable block palette alongside a text editor. This allows learners to browse available commands and drop-and-drop blocks into the text editor, thus providing some features of BBP alongside conventional TBP. Our analysis only identified three Hybrid programming environments, Frame-based editing, Pencil.cc, and Grasshopper. This small number, despite the relatively large design space, suggests this is a potential area of future work.

Another form of Hybrid Programming Environments are environments that support learners embedding text-based programs, or partial programs, into a BBP environment. One common way of doing this is to allow the user to create a new block and define that new block behavior via a text-based program. Conceptually, this is equivalent to allowing text-based functions to be written that can be called from within a block-based program. An early implementation of this can be found in PicoCricket (Fig. 9a, 9b), where the `block` keyword allows the user to create a new block whose behavior is specified via a text-based program. A second example of this approach is the Snappier! Environment [78], where creating a new block opens up an embedded JavaScript editor (Fig. 9c). A variation of this form of Hybrid Programming Environment is to have a block that allows users to input text-based code directly. This can be seen in Snap!'s JavaScript extension which introduces a JavaScript block and the TAIL extension for App Inventor [79], where TAIL code can be embedded with a special TAIL expression block.

A final discussion topic related to Hybrid Programming Environments is block-based tools that support keyboard-based editing of block-based programming. While they are not classified as Hybrid environments because they have no persistent textual representation of the program, they are noteworthy as the ability to use the keyboard to modify BBPs was one of the cited motivations for the investigation of Hybrid Environments [74]. There have been a number of implementations of keyboard-based navigation and editing of BBPs, including environments already discussed in this section, like Frame-based Editing and Pencil Code, as well as extensions to existing block-based environments, like typeblocking in App Inventor [76] and the block-editing cursor in GP [73].

4.2.5. Text extensions

A final aspect of the transition from BBP to TBP that is worth discussing is the growing number of extensions to BBP environments designed to add on TBP support. These extensions are not part of the environments themselves but are TBP capabilities that can be introduced through configuration menus. Different text extensions take different approaches for supporting text-based programming. For example, Snap! has the option to enable "JavaScript Extensions", which introduces a new JavaScript block to the blocks palette that allows the user to enter JavaScript code inside a block that can be executed within a block-based program. Other environments and block-based libraries support developers adding their own extensions (which can include the introduction of TBP) such as Scratch [25] or are open source so developers can modify the codebase to add TBP features.

5. Discussion

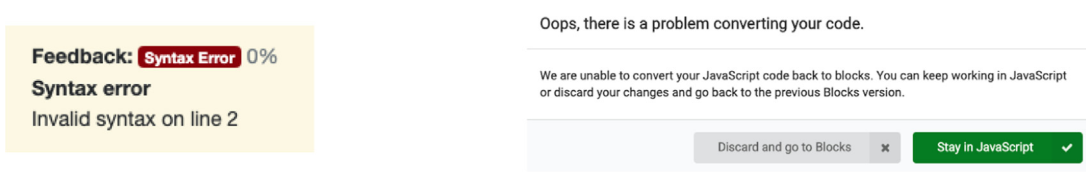
This paper presents the results of a systematic analysis of the features of BBP environments currently available and presents a classification scheme to define the current approaches used in supporting the blocks-to-text transition. Findings from this work have potential implications for a number of different audiences.

5.1. Implications for designers

The research reported here represents an important first step to understanding the design space thus far explored by the current set of BBP environments and the ways that transitioning from BBP to TBP is being supported. The analysis presented above can be useful to help guide those interested in designing new BBP environments to understand what has been done but also, where opportunities for innovation exist. For example, the fact that only three hybrid environments exist (discussed in 4.2.4) suggests that there are still fertile grounds yet to be tilled for finding new ways of supporting learners in transitioning from block-based to text-based languages. Likewise, the findings around the domains available and types of commands provided by BBP environments (Section 4.1.2) suggest gaps in the currently available set of BBP environments. For example, while there exist BBP environments for mobile app development and BBP environments for JavaScript, there does not yet seem to be a BBP environment focused on developing web applications or a BBP environment dedicated to video editing or statistical analysis. A review of the capabilities and organization of the currently available set of BBP environments (Section 4.1.1) may yield generative design ideas, including reimagining the way blocks are presented and organized and rethinking the way BBP environments' capabilities are organized and if/how they align to the target domain. Finally, while the design of TBP environments and IDEs was not the focus of the work, there may be implications for the design of TBP tools that draw inspiration from the BBP approach reviewed here, such as a consideration of ways to present and organize available commands (or functions) to leverage the browsability and tinkerability of BBP environments.

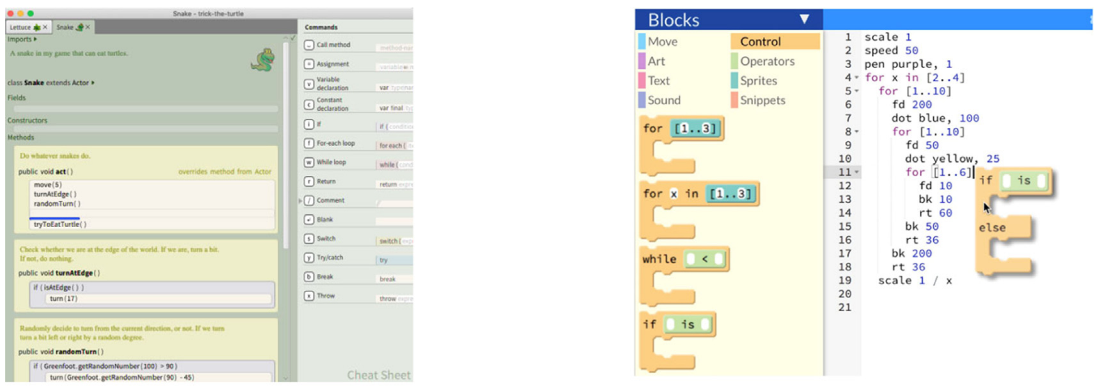
5.2. Implications for researchers

Just as this work presents a potential roadmap for designers of BBP environments, it also reveals potential future avenues for researchers interested in understanding how best to support novice programmers. For example, little work to date has investigated the relationship between BBP capabilities (reviewed in 4.1.1) and learning or affective outcomes. Do environments with greater capabilities help learners feel more confident or develop more robust understandings of computational ideas? Likewise, there are many open questions related to the



(a) (b)

Fig. 7. Two examples of handling syntax errors in Dual-Modality Environments.



(a) (b)

Fig. 8. Two Hybrid Programming Environments: (a) The Stride editor and (b) Pencil.cc.

```
block chirp2 :num
repeat :num [chirp wait 3]
end
```

(a) (b) (c)

Fig. 9. (a) Defining a new block in PicoCricket using the `block` keyword and a text-based program and (b) the resulting block that can be used in a block-based program. (c) The embedded JavaScript editor for defining custom blocks in Snappier!.

transition from BBP to TBP. The work that has been done to date usually presents BBP as a uniform approach to learning to program, rather than considering the breadth of implementations and features associated with BBP. Research into how specific features or capabilities of a BBP do or do not facilitate the transition to TBP seems a generative direction for future work. Likewise, the different forms of supporting the transition from BBP to TBP (Sections 4.2.2–4.2.4) are largely underexplored, with much of the research to date more focused on the design aspects than learning outcomes. Collectively, this work helps shed light on possible avenues for future research.

5.3. Implications for educators

Given the increased usage of Scratch and the popularity of many Scratch-inspired BBP environments in both classroom and informal settings, having a complete picture of the larger universe of available BBP environments could be beneficial to the educators and guide their decision making. While the goal of this analysis is not to rank or promote specific environments, this work can still be helpful for educators seeking to bring BBP into their classrooms by acting as a resource to understand the available set of environments and what features are available. This may include its inclusion of specific conceptual or technical features (reviewed in 4.1.1), alignment with a target domain (reviewed in 4.1.2), or its ability to scaffold the transition to TBP (reviewed in 4.2). For example, if a teacher is planning to use BBP as a stepping-stone for TBP, then they may want to use a BBP environment that supports that transition, such as Pencil Code, Tiled Grace, or one of Code.org's code studios. On the other hand, if the goal is to use BBP to support learners with a specific task, this analysis can help. Like suggesting BlockPy, NetsBlox, or DBSnap for data science, or Scratch or Alice for helping students create multimedia projects.

5.4. The role of pedagogy & curricula

This research was explicitly focused on the design of BBP environments and the various approaches used to support the transition to TBP. While these are important components of the overall task of teaching programming, there are a number of other very consequential aspects to educating learners not addressed by this research. Central among them is the role of teachers/educators and the curricular materials accompanying the BBP environment. This work did not take into consideration the ways a given BBP environment or approach to BBP supported various pedagogical approaches or whether or not a given tool has curricula that accompany it. This is not meant to downplay the importance of teachers or curricula, but rather, to constrain the focus of this work. Beyond features of an environment itself, it is important for educators to consider various other factors when deciding how to teach programming, a fact identified by other research on supporting educators making informed decisions on the materials they bring into their classrooms [80].

5.5. Availability of block-based programming environments

One interesting finding from this work is the large number of BBP environments that have been developed, studied, and published in the academic literature but are not available to the public. Of the 101 BBP environments identified, less than half could be used without needing to compile or locally host the environment. This is particularly surprising given that a majority of these environments were published in the last 5 years. While it is possible for the contribution of a BBP environment to stem from particular design innovations or environments to serve as proofs-of-concept for ideas without being robust enough to be generally useful, it also feels like a loss to the research community and the larger potential user-base for so many BBP environments to have been developed but be unavailable only a few years after their introduction. While the issue of long-term support of research projects is well-documented, especially as it relates to technology and education [23,81], it is worth considering how this fact is impacting the BBP community and explore potential solutions to make the contributions of this wide array of environments more sustainable.

5.6. Accessibility in block-based programming

A final point worth discussing is the importance of attending to issues of accessibility when creating or teaching with BBP. Many BBP environments do not support accessibility tools for students with disabilities [82]. In particular, BBP environments are often not screen-reader compatible, making them difficult or impossible for learners with visual impairments to use [82]. Work has been done to identify design approaches to make block-based tools more accessible [83], and there are some BBP environments that address this issue, such as Blocks4All [82] and the UncleGoose Toolkit [84], but environments that attend to this concern remain the minority of BBP. Beyond learners with visual impairments, there are numerous features in the design of BBP that align to Universal Design for Learning (UDL) recommendations or ways BBP can be used that align with UDL, including using multiple representations and including scaffolding to help learners work at their own pace [85].

5.7. Limitations

While this analysis was conducted to the best of our abilities, it is not without its limitations. First and foremost, among them is the difficulty of trying to analyze a rapidly changing landscape. This work only includes environments we were able to identify at the time of analysis. With new environments being introduced and older environments no longer being supported, as soon as this analysis was complete, it was immediately at least a little out of date. This is a limitation of this methodology, but we nevertheless think this analysis is a useful snapshot of the state of the field. Similarly, many environments have been forked, support user-developed extensions, or had additions made to them (e.g., [25]). In this work, we focus on the canonical versions of the tools, but we acknowledge that many BBP environments support additional features beyond what was analyzed. A second limitation stems from the academic orientation of the research approach. Our methodology began with searching academic repositories. This means environments developed by those not connected to academia, including much of the commercial sector, may have BBP environments not captured by this review. With the rapid growth of coding-related toys, we suspect a number of commercial environments were missed but reviewing that space was beyond the scope of this work (see [2] for work in this area). A final limitation of this work stems from the fundamental mismatch between unique, learner-directed, and often multiple purpose technologies and the effort to systematically categorize them. This mismatch can be seen in a few ways, including trying to define a domain for an environment or trying to systematically catalog capabilities and organization despite varying designs. While we think the result is useful, it is nevertheless a limitation of trying to quantify such diverse and personalized technologies.

6. Conclusion

The success of Scratch has influenced many researchers and designers to create BBP environments. In this systematic review of the academic literature, we identified 101 distinct BBP environments and analyzed 46 of them to identify the domain of focus, capabilities, and if and how they support users in transitioning from block-based to text-based programming. Our analysis revealed four distinct approaches for supporting the block-to-text transition: Blocks-only, Dual-modality, One-way Transition, and Hybrid. Given the increasing presence of BBP environments in K-12 education and the growing ecosystem of educational programming environments and curricula, understanding the current state of the design space is important for identifying potential future directions and opportunities for design innovations. Similar to many aspects of computer science, BBP environments are constantly updating and rapidly changing. Since the initial queries for the research were conducted, a number of environments that were available at the outset are no longer accessible. At the same time, there have also been new BBP environments introduced. The introduction of additional environments seems in part spurred by the COVID-19 pandemic and rise of virtual learning, as several robotics companies that have historically focused on physical devices, such as VEX Robotics and Sphero, have started working to bring fully virtual platforms to market. This rate of change makes a review such as this useful as a means to take stock of where we are and also to consider future directions for these types of environments. As new BBP environments emerge, it is our hope that this work can inform the next generation of tools, curricula, and classroom practice. In doing so, we can help welcome the next generation of computationally literature learners to the world of computer science.

CRediT authorship contribution statement

Yuhan Lin: Conceptualization, Methodology, Formal analysis, Investigation, Writing – original draft. **David Weintrop:** Conceptualization, Validation, Writing – review & editing, Resources, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix

See Table A.5.

Table A.5
Block-based Programming Environments identified. Appendix presents the full list of BBP environments identified as part of this analysis and provides a brief description of each. Where possible, the descriptions were pulled from the academic article or website where the environment is presented.

Block-based Programming Environment	Brief Description
AgentCubes [86]	AgentCubes is a 3D rapid game-prototyping environment that enables even 10-year-old children to make simulations and games in just a few hours.
Alice [87]	Alice is a 3-D Interactive Graphics Programming Environment that makes it easy for novices to develop interesting 3-D environments.
Amphibian [59]	Amphibian, based on Pencil Code's Droplet Editor, enables switching between blocks and text within IntelliJ IDEA, a common IDE for the Java language.
APP Inventor [10]	MIT App Inventor enables beginners and non-programmers to create apps for their phones and tablets.
ARcadia [88]	ARcadia can create rapid, low-cost prototyping of tangible user interfaces that only requires access to a webcam, a web browser, and paper.
ArduBlock [89]	ArduBlock is a graphical programming language for Arduino.
ArduBlockly [90]	Ardublockly is a visual programming editor for Arduino built with Google's Blockly library.
Arduviz [91]	Arduviz is a visual programming integrated development environment for Arduino.
BeetleBlocks [92]	Beetle Blocks combines a BBP language with a generative model for 3D space, drawing on 'turtle geometry,' that allows designers to learn computational concepts and use them for their designs.
Block-C [93]	Block-C is a block-based programming learning tool for C programming language that uses the recognition over recall approach on top of the rigid and complex constructs of the C language.
BlockEditor [68]	BlockEditor is a BBP environment that allows users to move back and forth between Block (the block language used here) and Java.
Blockly [94]	Blockly is an open-source library that makes it easy to add block-based visual programming to web-based applications.
Blockly @rduino [95]	Blockly @rduino is a visual programming editor for Arduino.
BlocklySQL [55]	BlocklySQL is a block-based editor for SQL.
BlockPy [96]	BlockPy is a Python-based environment that supports data science and promotes transfer for students to text-based programming.
Blocks4DS [97]	Blocks4DS is an instructional block-based technology for students to learn data structures.
BlockyTalky [98]	BlockyTalky is an interactive computer music toolkit for kids.
Bots & (Main) Frames [99]	Bots & (Main) Frames is an educational programming game that asks users to move a robot through two-dimensional puzzles.
Bricklayer-lite [100]	Bricklayer is a functional programming environment that supports artistic and mathematical expression.
Calypso [101]	Calypso is an intelligent robot programming framework for the Cozmo robot, with real computer vision, speech recognition, and AI.
Catroid/Catrobat/Pocket Code [102]	Catroid is an open-source visual programming system that allows casual and first-time users to develop their own animations and games solely using their Android phones or tablets.
Cellular [103]	Cellular is a version of Snap! that supports the creation of password-protected blocks such that they can be used but not viewed
ChoiCo [104]	ChoiCo integrates three different affordances for designing its games: a map-based (GIS) game scene, a simplified database, and block-based programming editors.
CoBlox [105]	CoBlox is a block-based programming interface for Roberta, a single-armed industrial robot.
code.org/hour of code [106]	Code.org is a nonprofit dedicated to expanding access to computer science. Their website and courses include a number of block-based environments based on Pencil Code and Blockly.
CodePlayground [107]	CodePlayground is a block-based programming environment designed for the Adafruit Circuit Playground.
COPPER [108]	COPPER (CustOmizable Puzzle Programming EnviRonment) is a meta-configurable tool for creating coding puzzles on a grid using a blocks-based programming language.
CT-Blocks [109]	CT-Blocks is a cloud and block-based programming language that helps users learn computational and mathematical ideas.
Cubely [110]	Cubely, an immersive VR programming environment in which novice programmers solve programming puzzles within a virtual world.
DB-Learn [111]	DB-Learn is a browser-based, visual programming environment that facilitates the learning of relational algebra concepts.
DBSnap [112]	DBSnap is a web-based application to build database queries, particularly relational algebra queries with block-based tools.
DBSnap++ [113]	DBSnap++ is a web-based environment that allows users to build database queries and enables the specification of dynamic data-driven programs.
Dragon Architect [114]	Dragon Architect is an educational block-based programming game.
Dronely [115]	Dronely is a visual block programming language that allows users to build programs for the control of small drones.
Droplet [72]	Droplet is a programming editor, which has a drag-and-drop block interface like Scratch but allows users to edit the program through a text editor.
Dwengo Blocks [116]	Dwengo Blocks is a web-based tool for programming microcontrollers and simulations.
Entry [117]	Entry is a Korean BBP environment designed to help students learn to code and supports automated feedback and teacher dashboards.
Flowboard [118]	Flowboard is a BBP that uses Flow-Based Programming (FBP) rather than the usual imperative programming paradigm.
Frame-based editing [18]	Frame-based editing has the resistance to errors and approachability of BBP while retaining the flexibility and more conventional programming semantics of text-based programming languages.
FreeCoffee [119]	FreeCoffee is a blocks-based language whose editor goes well beyond terse slot labels and communicates the meaning of a block using complete grammatical sentences.
GameBlox [120]	Gameblox is a game editor that uses a BBP language to allow anyone to make games.
Gamemaker studio [121]	GameMaker Studio is a visual programming software to create digital games.

(*continued on next page*)

Table A.5 (*continued*).

Block-based Programming Environment	Brief Description
GP [122]	GP is a general-purpose BBP language.
Grasshopper [123]	Grasshopper is an Android phone application that aims to introduce adults with no coding experience to JavaScript and computational thinking through short, self-contained programming puzzles.
Hybrid Pencil.cc [124]	Hybrid Pencil.cc presents a blocks palette alongside a text-based editor to support drag-and-drop authoring in a text-based program.
Intelliblox [125]	IntelliBlox is a Blockly-inspired toolkit for the Unity cross-platform game engine that enables learners to create block-based programs within immersive game-based learning environments.
Interactex [126]	*Interactex* is a visual programming environment specifically designed for smart textiles.
iSnap [127]	iSnap is an extension of Snap!, which provides on-demand hints and feedback to help students complete programming assignments.
Jeeves [128]	Jeeves is a visual language to facilitate Experience Sampling Method (ESM) application creation.
KoDu [129]	Kodu is a rule-based programming environment for children to define behaviors for objects in a virtual world.
Kokopelli's World [130]	Kokopelli's World is a BBP environment and set of lessons designed to introduce and teach the basics of computer programming to students with learning disabilities.
LaPlaya [131]	LaPlaya is an environment based on Snap! that includes scaffolds to make programming more accessible for younger learners.
LEGO MINDSTORMS EV3 [132]	LEGO Mindstorms EV3 includes a BBP environment to control the behaviors of learner-constructed LEGO robots.
LogoBlocks [133]	LogoBlocks is an early BBP environment designed for the Programmable Brick.
Looking Glass [134]	Looking Glass is an Alice-based BBP environment for ages 10 and up to create and share animated stories, simple games, and even virtual pets.
Madeup [135]	Madeup supports the generation of 3D models through Logo-like commands.
Makecode [136]	Microsoft MakeCode is a platform and accompanying web app for simplifying the programming of microcontroller-based devices.
Map-Blocks [137]	Map-Blocks is based on CT-Blocks and allows K-12 students to explore online data sources.
MARBLS [138]	MARBLS (Medical Alert Rule BuiLding System) is a visual end-user programming environment to facilitate the design and testing of clinical alert rules.
mBlock [139]	mBlock is a BBP environment to build interactive Arduino projects.
Milo [140]	Milo is a web-based visual programming environment for data science education, designed for students without prior programming experience.
Modkit [7]	Modkit is a toolkit that designers can use to create their own interactive objects for the Arduino.
MUzECS [141]	MUzECS is a microcontroller and accompanying BBP environment that allows users to create music programmatically.
NetsBlox [142]	NetsBlox is a cloud-based BBP environment based on Snap! that enables novice programmers to create networked programs.
NETTango [143]	NetTango is an agent-based modeling environment designed for elementary school students to use on a multi-touch tabletop surface.
NeuroBlock [144]	NeuroBlock is a BBP environment for neurofeedback application development.
OpenBlocks [145]	OpenBlocks is an extendable framework for creating BBP languages
Ozoblockly [146]	Ozoblockly is a BBP environment to program behaviors and patterns for the Ozobot robot.
Patch [147]	Patch is a Scratch-based environment that incorporates bits of Python syntax into the blocks.
Pencil.cc [43]	Pencil.cc is a customized version of Pencil Code developed for research purposes to study the effects of BBP on learners.
Pencil code [16]	Pencil Code is a BBP tool that based on the Droplet editor supports learners moving back-and-forth between blocks and text and embeds numerous web technologies and libraries.
PictureBlocks [148]	PictureBlocks is a BBP environment that allows users to transform primitive pictures into complex geometric designs which can then be physically created with digital fabrication tools.
Poliglot Environment [71]	Poliglot environment aims to smooth the transition from block to text by presenting the program simultaneously in both notations and allows editing in either one.
PRIME [149]	PRIME is an adaptive block-based programming environment designed to support novices as they learn introductory programming concepts at the undergraduate level.
PyBlocks [60]	PyBlocks is a blocks-based environment that allows novice programmers to construct and execute Python programs.
QIS [150]	QIS is a novice-friendly refactoring tool for Scratch 3.0.
Reduct [151]	Reduct is an educational game that teaches novices core programming concepts which include functions, Booleans, equality, conditionals, and mapping functions over sets.
Resource Rush [152]	Resource Rush is a game designed to resemble BBP and allows users to learn the fundamentals of programming in an open-ended game environment.
Robomise [153]	Robomise is a BBP assessment tool.
Scratch [5]	Scratch allows users (primarily ages 8 to 16) to learn computer programming while working on personally meaningful projects such as animated stories and games.
Scratch Data Block [154]	Scratch Data Blocks is a Scratch-based toolkit that allows Scratch users to program with public data from the Scratch online community.
Scratch Jr [155]	ScratchJr is an environment based on Scratch and redesigned for the unique developmental and learning needs of children in kindergarten to second grade.
Smart Block [156]	Smart Block is a BBP language for SmartThings home automation.
Snap! [157]	Snap! is a broadly inviting programming language for kids and adults that is also a platform for serious study of computer science.

(*continued on next page*)

Table A.5 (*continued*).

Block-based Programming Environment	Brief Description
Snappier! [78]	Snappier! is a Snap!-based environment that allows users to program new blocks using JavaScript.
Sonification [158]	Sonification is a BBP language for data sonification, the process of creating audio algorithms and controlling them with streams of data.
Squeak eToys [159]	Squeak eToys is a media-rich authoring environment and visual programming system
StarLogo TNG [160]/StarLogo Nova [161]	StarLogo Nova (formerly StarLogo TNG) is an agent-based game and simulation programming environment that combines an easy-to-use BBP language with a powerful simulation engine and 3D renderer.
StoryBlocks [162]	StoryBlocks is a tangible block-based game that enables blind programmers to learn basic programming concepts by creating audio stories.
StoryMakAR [163]	StoryMakAR is a new augment reality, Internet of Things system for children that uses block programming, physical prototyping, and event-based storytelling to bring stories to life.
TAIL [164]	TAIL is a textual programming language isomorphic to the blocks language of MIT App Inventor 2 that has extended App Inventor 2 with code blocks, a novel mechanism that enables bidirectional isomorphic conversions between blocks and text program fragments.
Tiled Grace/Mini grace [67]	Tiled Grace is a BBP system backed by a conventional textual language that allows switching back and forth between block-based and textual editing of the same code.
TinkerCAD [165]	TinkerCAD is an online programming environment for users to create and animate 3D designs.
TouchDevelop [166]	Touch Develop combines a cross-platform browser-based IDE for the creation of mobile cloud apps, an online programmer/user community, and an app store.
Tuk Tuk [167]	Tuk Tuk is a BBP game for children ranging from kindergarten (junior version) to middle school level (standard version).
TurtleBlocks [148]	TurtleBlocks is a BBP environment that allows users to author geometric patterns which can then be physically created with digital fabrication tools.
Typeblocking [76]	Typeblocking integrates keyboard input with block programming and translates user text input to blocks and to improves the experience of BBP.
Unruly Splats [168]	Unruly Splats are a set of foot-sized floor buttons that light up, sense pressure, and make sounds, according to programs that learners age 6 and up author using a Scratch extension.
Venbrace [169]	Venbrace is a fully-braced textual syntax for App Inventor.
VEXcode VR [170]	VEXcode VR is an online BBP environment for the VEX VR Robot that with virtual robotics that supports exporting programs to text-based languages.
XLBlocks [171]	XLBlocks is a block-based formula editor for Excel.

References

[1] Barbara Ericson, W. Richards Adrion, Renee Fall, Mark Guzdial, State-based progress towards computer science for all, ACM Inroads 7 (4) (2016) 57–60, http://dx.doi.org/10.1145/2994607.

[2] Jody Clarke-Midura, Victor R. Lee, Jessica F. Shumway, Megan M. Hamilton, The building blocks of coding: a comparison of early childhood coding toys, Inf. Learn. Sci. 120 (7/8) (2019) 505–518, http://dx.doi.org/10.1108/ILS-06-2019-0059.

[3] David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, Franklyn Turbak, Learnable programming: blocks and beyond, Commun. ACM 60 (6) (2017) 72–80, http://dx.doi.org/10.1145/3015455.

[4] David Weintrop, Block-based programming in computer science education, Commun. ACM 62 (8) (2019) 22–25, http://dx.doi.org/10.1145/3341221.

[5] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, Yasmin Kafai, Scratch: programming for all, Commun. ACM 52 (11) (2009) 60–67, http://dx.doi.org/10.1145/1592761.1592779.

[6] James Devine, Joe Finney, Peli de Halleux, Michał Moskal, Thomas Ball, Steve Hodges, MakeCode and CODAL: intuitive and efficient embedded systems programming for education, in: Proceedings of the 19th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems, LCTES 2018, ACM Press, Philadelphia, PA, USA, 2018, pp. 19–30, http://dx.doi.org/10.1145/3211332.3211335.

[7] Amon Millner, Edward Baafi, Modkit: blending and extending approachable platforms for creating computer programs and interactive objects, in: Proceedings of the 10th International Conference on Interaction Design and Children, IDC '11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 250–253, http://dx.doi.org/10.1145/1999030.1999074.

[8] Austin Cory Bart, Javier Tibau, Dennis Kafura, Clifford A. Shaffer, Eli Tilevich, Design and evaluation of a block-based environment with a data science context, IEEE Trans. Emerg. Top. Comput. 99 (2017) (2017) 1, http://dx.doi.org/10.1109/TETC.2017.2729585.

[9] David Weintrop, David C. Shepherd, Patrick Francis, Diana Franklin, Blockly goes to work: Block-based programming for industrial robots, in: 2017 IEEE Blocks and beyond Workshop, 2017, pp. 29–36, http://dx.doi.org/10.1109/BLOCKS.2017.8120406.

[10] David Wolber, Harold Abelson, Mark Friedman, Democratizing computing with app inventor, GetMob.: Mob. Comput. Commun. 18 (4) (2015) 53–58.

[11] Joanna Goode, Gail Chapman, Jane Margolis, Beyond curriculum: the exploring computer science program, ACM Inroads 3 (2) (2012) 47–53, http://dx.doi.org/10.1145/2189835.2189851.

[12] Diana Franklin, David Weintrop, Jennifer Palmer, Merijke Coenraad, Melissa Cobian, Kristan Beck, Andrew Rasmussen, Sue Krause, Max White, Marco Anaya, Zachary Crenshaw, Scratch encore: The design and pilot of a culturally-relevant intermediate scratch curriculum, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, ACM, Portland OR USA, 2020, pp. 794–800, http://dx.doi.org/10.1145/3328778.3366912.

[13] The beauty and joy of computing: BJC. Retrieved from https://bjc.berkeley.edu/.

[14] Code.org. Retrieved from https://code.org/.

[15] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, Steve Cooper, Mediated transfer: Alice 3 to Java, in: Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, SIGCSE '12, ACM Press, Raleigh, North Carolina, USA, 2012, p. 141, http://dx.doi.org/10.1145/2157136.2157180.

[16] David. Bau, D. Anthony Bau, Mathew. Dawson, C. Sydney Pickens, Pencil code: block code for a text world, in: Proceedings of the 14th International Conference on Interaction Design and Children, IDC '15, ACM Press, Boston, Massachusetts, 2015, pp. 445–448, http://dx.doi.org/10.1145/2771839.2771875.

[17] David Weintrop, Nathan Holbert, Michael Tissenbaum, Considering alternative endpoints: An exploration in the space of computing educations, 2020, p. 11, (2020).

[18] Michael Kölling, Neil C.C. Brown, Amjad Altadmri, Frame-based editing: Easing the transition from blocks to text-based programming, in: Proceedings of the Workshop in Primary and Secondary Computing Education on ZZZ, WiPSCE '15, ACM Press, London, United Kingdom, 2015, pp. 29–38, http://dx.doi.org/10.1145/2818314.2818331.

[19] Kris Powers, Stacey Ecott, Leanne M. Hirshfield, Through the looking glass: teaching CS0 with Alice, in: Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '07, Association for Computing Machinery, Covington, Kentucky, USA, 2007, pp. 213–217, http://dx.doi.org/10.1145/1227310.1227386.

[20] David Weintrop, Uri Wilensky, Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms, Comput. Educ. 142 (2019) (2019) 103646, http://dx.doi.org/10.1016/j.compedu.2019.103646.

[21] Monica M. McGill, Adrienne Decker, Construction of a taxonomy for tools, languages, and environments across computing education, in: Proceedings of the 2020 ACM Conference on International Computing Education Research,

Virtual Event New Zealand, ACM, 2020, pp. 124–135, http://dx.doi.org/10.1145/3372782.3406258.

[22] Miguel Ceriani, Paolo Bottoni, SparqlBlocks: using blocks to design structured linked data queries, J. Vis. Lang. Sentient Syst. 3 (2017) (2017) 1–21.

[23] David K. Cohen, Educational technology, policy, and practice, Educ. Eval. Policy Anal. 9 (2) (1987) 153–170, http://dx.doi.org/10.3102/01623737009002153.

[24] Stephen Cooper, Wanda Dann, Randy Pausch, Alice: a 3-D tool for introductory programming concepts, J. Comput. Sci. Colleges 15 (5) (2000) 107–116.

[25] Sayamindu Dasgupta, Shane M. Clements, Abdulrahman Y. Idlbi, Chris Willis-Ford, Mitchel Resnick, Extending scratch: New pathways into programming, in: 2015 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, IEEE, Atlanta, GA, 2015, pp. 165–169, http://dx.doi.org/10.1109/VLHCC.2015.7357212.

[26] David Weintrop, Uri Wilensky, How block-based, text-based, and hybrid block/text modalities shape novice programming practices, International Journal of Child-Computer Interaction 17 (2018) 83–92, http://dx.doi.org/10.1016/j.ijcci.2018.04.005.

[27] David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, Franklyn Turbak, Learnable programming: blocks and beyond, Commun. ACM 60 (6) (2017) 72–80, http://dx.doi.org/10.1145/3015455.

[28] Daniel Wendel, Paul Medlock-Walton, Thinking in blocks: Implications of using abstract syntax trees as the underlying program model, in: 2015 IEEE Blocks and beyond Workshop, Blocks and Beyond, IEEE, Atlanta, GA, USA, 2015, pp. 63–66, http://dx.doi.org/10.1109/BLOCKS.2015.7369004.

[29] Brian Harvey, Jens Mönig, Bringing no ceiling to scratch: Can one language serve kids and computer scientists? 2010, p. 10, (2010).

[30] David Wolber, Hal Abelson, Ellen Spertus, Liz Looney, App Inventor: Create Your Own Android Apps, O'Reilly Media, Sebastopol, Calif, 2011.

[31] Alex Repenning, Agentsheets: a tool for building domain-oriented visual programming environments, in: Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems, ACM, 1993, pp. 142–143, Retrieved October 22, 2014 from http://dl.acm.org/citation.cfm?id=169119.

[32] Neil Fraser, Ten things we've learned from blockly, in: 2015 IEEE Blocks and beyond Workshop, Blocks and Beyond, IEEE, Atlanta, GA, USA, 2015, pp. 49–50, http://dx.doi.org/10.1109/BLOCKS.2015.7369000.

[33] Ephraim P. Glinert, Towards' Second Generation'Interactive, Graphical Programming Environments, Department of Computer Science, Rensselaer Polytechnic Institute, 1986.

[34] Andrew Begel, A graphical programming language for interacting with the world, 1996, p. 23, (1996).

[35] Alexander Repenning, Moving beyond syntax: Lessons from 20 years of blocks programing in AgentSheets, J. Vis. Lang. Sentient Syst. 3 (2017) (2017) 68–91, http://dx.doi.org/10.18293/VLSS2017.

[36] Veronique Donzeau-Gouge, Gerard Huet, Gilles Kahn, Bernard Lang, Programming Environments Based on Structured Editors: The MENTOR Experience, INSTITUT NATIONAL DE RECHERCHE D'INFORMATIQUE ET D'AUTOMATIQUE ROCQUENCOURT (FRANCE, 1980.

[37] Sten Minör, Interacting with structure-oriented editors, Int. J. Man-Mach. Stud. 37 (4) (1992) 399–418, http://dx.doi.org/10.1016/0020-7373(92)90002-3.

[38] David Weintrop, Uri Wilensky, To block or not to block, that is the question: students' perceptions of blocks-based programming, in: Proceedings of the 14th International Conference on Interaction Design and Children, IDC '15, ACM Press, Boston, Massachusetts, 2015, pp. 199–208, http://dx.doi.org/10.1145/2771839.2771860.

[39] Diana Franklin, Gabriela Skifstad, Reiny Rolock, Isha Mehrotra, Valerie Ding, Alexandria Hansen, David Weintrop, Danielle Harlow, Using upper-elementary student performance to understand conceptual sequencing in a blocks-based curriculum, in: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE '17, ACM, New York, NY, USA, 2017, pp. 231–236, http://dx.doi.org/10.1145/3017680.3017760.

[40] Shuchi Grover, Satabdi Basu, Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic, in: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, ACM Press, New York, NY, 2017, pp. 267–272, http://dx.doi.org/10.1145/3017680.3017723.

[41] Colleen M. Lewis, How programming environment shapes perception, learning and goals: logo vs. scratch, in: Proceedings of the 41st ACM Technical Symposium on Computer Science Education, SIGCSE '10, Milwaukee, Wisconsin, USA, 2010, p. 346, http://dx.doi.org/10.1145/1734263.1734383.

[42] O. Meerbaum-Salant, M. Armoni, M.M. Ben-Ari, Learning computer science concepts with Scratch, in: Proceedings of the Sixth international workshop on Computing education research, 2010, pp. 69–76.

[43] David Weintrop, Uri Wilensky, Comparing block-based and text-based programming in high school computer science classrooms, ACM Trans. Comput. Educ. 18 (1) (2017) 1–25, http://dx.doi.org/10.1145/3089799.

[44] Quinn Burke, Yasmin B. Kafai, Decade of game making for learning: From tools to communities, in: Marios C. Angelides, Harry Agius (Eds.), Handbook of Digital Games, John Wiley & Sons, Inc., Hoboken, NJ, USA, 2014, pp. 689–709, http://dx.doi.org/10.1002/9781118796443.ch26.

[45] Deborah Fields, Veena Vasudevan, Yasmin B. Kafai, The programmers' collective: fostering participatory culture by making music videos in a high school Scratch coding workshop, Interact. Learn. Environ. 23 (5) (2015) 613–633, http://dx.doi.org/10.1080/10494820.2015.1065892.

[46] Ricarose Roque, Yasmin Kafai, Deborah Fields, From tools to communities: Designs to support online creative collaboration in scratch, in: Proceedings of the 11th International Conference on Interaction Design and Children, ACM, 2012, pp. 220–223, Retrieved July 17, 2015 from http://dl.acm.org/citation.cfm?id=2307130.

[47] Dan Garcia, Brian Harvey, Tiffany Barnes, The beauty and joy of computing, ACM Inroads 6 (4) (2015) 71–79, http://dx.doi.org/10.1145/2835184.

[48] Ryan Garlick, Ebru Celikel Cankaya, Using Alice in CS1: A quantitative experiment, in: Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education, ACM, 2010, pp. 165–168.

[49] David J. Malan, Henry H. Leitner, Scratch for budding computer scientists, in: ACM SIGCSE Bulletin, ACM, 2007, pp. 223–227.

[50] Wolfgang Slany, Tinkering with Pocket Code, a Scratch-like programming app for your smartphone, in: Proceedings of Constructionism 2014, Vienna, Austria, 2014.

[51] Douglas B. Clark, Paul Medlock-Walton, Raúl Boquín, K. Klopfer, Multiplayer disciplinarily-integrated agent-based games: SURGE gameblox, Simul. Gaming (2018) (2018) 89–107.

[52] Sarah Esper, Stephen R. Foster, William G. Griswold, CodeSpells: embodying the metaphor of wizardry for programming, in: Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education, ACM, 2013, pp. 249–254.

[53] David Weintrop, Uri Wilensky, RoboBuilder: A program-to-play constructionist video game, in: Proceedings of the Constructionism 2012 Conference, Athens, Greece, 2012.

[54] Sven Jacobs, Steffen Jaschke, SQheLper: A block-based syntax support for SQL, in: 2021 IEEE Global Engineering Education Conference, EDUCON, IEEE, Vienna, Austria, 2021, pp. 478–481, http://dx.doi.org/10.1109/EDUCON46332.2021.9453897.

[55] Nicolai Pöhner, Timo Schmidt, André Greubel, Martin Hennecke, Matthias Ehmann, BlocklySQL: A new block-based editor for SQL, in: Proceedings of the 14th Workshop in Primary and Secondary Computing Education on, WiPSCE'19, ACM Press, Glasgow, Scotland, Uk, 2019, pp. 1–2, http://dx.doi.org/10.1145/3361721.3362104.

[56] Michael S. Horn, Corey Brady, Arthur Hjorth, Aditi Wagh, Uri Wilensky, Frog pond: a codefirst learning environment on evolution and natural selection, in: Proceedings of the 2014 Conference on Interaction Design and Children, IDC '14, Association for Computing Machinery, Aarhus, Denmark, 2014, pp. 357–360, http://dx.doi.org/10.1145/2593968.2610491.

[57] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C. Shepherd, Diana Franklin, Evaluating coblox: A comparative study of robotics programming environments for adult novices, in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, Vol. 366, ACM Press, Montreal QC, Canada, 2018, pp. 1–12, http://dx.doi.org/10.1145/3173574.3173940.

[58] Carlos Pereira Atencio, ArduBlockly Retrieved from https://ardublockly.embeddedlog.com.

[59] Jeremiah Blanchard, Chistina Gardner-McCune, Lisa Anthony, Amphibian: Dual-modality representation in integrated development environments, in: 2019 IEEE Blocks and beyond Workshop (B B), 2019, pp. 83–85, http://dx.doi.org/10.1109/BB48857.2019.8941213.

[60] Matthew Poole, Extending the design of a blocks-based python environment to support complex types, in: 2017 IEEE Blocks and beyond Workshop (B B), 2017, pp. 1–7, http://dx.doi.org/10.1109/BLOCKS.2017.8120400.

[61] Michal Armoni, Orni Meerbaum-Salant, Mordechai Ben-Ari, From scratch to real programming, ACM Trans. Comput. Educ. 14 (4) (2015) 1–15, http://dx.doi.org/10.1145/2677087.

[62] Shuchi Grover, Roy Pea, Stephen Cooper, Designing for deeper learning in a blended computer science course for middle school students, Comput. Sci. Educ. 25 (2) (2015) 199–237, http://dx.doi.org/10.1080/08993408.2015.1033142.

[63] Johnny Saldaña, The coding manual for qualitative researchers, Sage (2015) Retrieved December 7, 2016 from https://books-google-com.turing.library.northwestern.edu/books?hl=en&lr=&id=ZhxiCgAAQBAJ&oi=fnd&pg=PP1&dq=The+Coding+Manual+for+Qualitative+Researcher&ots=yHZa1BTQaV&sig=xnAdqqr_XT9yJuqlWPhTbGNvnLg.

[64] David Weintrop, David Bau, Uri Wilensky, The cloud is the limit: A case study of programming on the web, with the web, Int. J. Child-Comput. Interact. 20 (2019) 1–8, http://dx.doi.org/10.1016/j.ijcci.2019.01.001.

[65] David Weintrop, Uri Wilensky, Between a block and a typeface: Designing and evaluating hybrid programming environments, in: Proceedings of the 2017 Conference on Interaction Design and Children, ACM, Stanford California USA, 2017, pp. 183–192, http://dx.doi.org/10.1145/3078072.3079715.

[66] Jeremiah Blanchard, Christina Gardner-McCune, Lisa Anthony, Dual-modality instruction and learning: A case study in CS1, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, ACM, Portland OR USA, 2020, pp. 818–824, http://dx.doi.org/10.1145/3328778.3366865.

[67] Michael Homer, James Noble, Lessons in combining block-based and textual programming, J. Vis. Lang. Sentient Syst. 3 (2017) (2017) 22–39, http://dx.doi.org/10.18293/VLSS2017.

[68] Yoshiaki Matsuzawa, Takashi Ohata, Manabu Sugiura, Sanshiro Sakai, Language migration in non-CS introductory programming through mutual language translation environment, in: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, ACM Press, 2015, pp. 185–190, http://dx.doi.org/10.1145/2676723.2677230.

[69] David Weintrop, Nathan Holbert, From blocks to text and back: Programming patterns in a dual-modality environment, in: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, ACM, Seattle Washington USA, 2017, pp. 633–638, http://dx.doi.org/10.1145/3017680.3017707.

[70] Austin Cory Bart, Eli Tilevich, Clifford A. Shaffer, Dennis Kafura, Position paper: From interest to usefulness with blockpy, a block-based, educational environment, in: 2015 IEEE Blocks and beyond Workshop, Blocks and beyond, IEEE, Atlanta, GA, USA, 2015, pp. 87–89, http://dx.doi.org/10.1109/BLOCKS.2015.7369009.

[71] Žiga Leber, Matej Črepinek, Tomaž Kosar, Simultaneous multiple representation editing environment for primary school education, in: 2019 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2019, pp. 175–179, http://dx.doi.org/10.1109/VLHCC.2019.8818927.

[72] David Bau, Droplet, a blocks-based editor for text code, J. Comput. Sci. Coll. 30 (6) (2015) 138–144.

[73] Jens Mönig, Yoshiki Ohshima, John Maloney, Blocks at your fingertips: Blurring the line between blocks and text in GP, in: 2015 IEEE Blocks and beyond Workshop, Blocks and Beyond, 2015, pp. 51–53, http://dx.doi.org/10.1109/BLOCKS.2015.7369001.

[74] Michael Kölling, Neil C.C. Brown, Amjad Altadmri, Frame-based editing, J. Vis. Lang. Sentient Syst. 3 (2017) (2017) 40–67, http://dx.doi.org/10.18293/VLSS2017.

[75] Thomas W. Price, Neil C.C. Brown, Dragan Lipovac, Tiffany Barnes, Michael Kölling, Evaluation of a frame-based programming editor, in: Proceedings of the 2016 ACM Conference on International Computing Education Research, ACM, Melbourne VIC Australia, 2016, pp. 33–42, http://dx.doi.org/10.1145/2960310.2960319.

[76] Terrance Liang, Typeblocking : Keyboard Integration with Block Programming in StarLogo Nova (Thesis), Massachusetts Institute of Technology, 2019, Retrieved September 16, 2021 from https://dspace.mit.edu/handle/1721.1/123162.

[77] Daniel Wendel, Position: Meeting the promise of blocks-as-AST-nodes editing with typeblocking, in: 2019 IEEE Blocks and beyond Workshop (B & B), IEEE, Memphis, TN, USA, 2019, pp. 23–26, http://dx.doi.org/10.1109/BB48857.2019.8941210.

[78] David Weintrop, Minding the gap between blocks-based and text-based programming, in: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE '15, ACM, New York, NY, USA, 2015, p. 720, http://dx.doi.org/10.1145/2676723.2693622.

[79] Karishma Chadha, Improving App Inventor Through Conversion Between Blocks and Text (Honors Thesis), Wellesley College, 2014.

[80] David Weintrop, Merijke Coenraad, Jen Palmer, Diana Franklin, The teacher accessibility, equity, and content (TEC) rubric for evaluating computing curricula, ACM Trans. Comput. Educ. 20 (1) (2020) 1–30, http://dx.doi.org/10.1145/3371155.

[81] Curtis Lynch, Bonnie Hansen Grafton, A retrospective view of a study of middle school science curriculum materials: Implementation, scale-up, and sustainability in a changing policy environment, J. Res. Sci. Teach. 49 (3) (2012) 305–332, http://dx.doi.org/10.1002/tea.21000.

[82] Lauren R. Milne, Richard E. Ladner, Position: Accessible block-based programming: Why and how, in: 2019 IEEE Blocks and beyond Workshop (B & B), IEEE, Memphis, TN, USA, 2019, pp. 19–22, http://dx.doi.org/10.1109/BB48857.2019.8941230.

[83] Stephanie Ludi, Mary Spencer, Design considerations to increase block-based language accessibility for blind programmers via blockly, J. Vis. Lang. Sentient Syst. 3 (1) (2017) 119–124.

[84] Emmanuel Schanzer, Sina Bahram, Shriram Krishnamurthi, Building an accessible block environment: multi-language, fully-accessible AST-based editing in the browser, in: ACM SPLASH BLOCKS+ Workshop, 2018, (2018).

[85] Alexandria K. Hansen, Eric R. Hansen, Hilary A. Dwyer, Danielle B. Harlow, Diana Franklin, Differentiating for diversity: Using universal design for learning in elementary computer science education, in: Proceedings of the 47th ACM Technical Symposium on Computing Science Education, SIGCSE '16, Association for Computing Machinery, Memphis, Tennessee, USA, 2016, pp. 376–381, http://dx.doi.org/10.1145/2839509.2844570.

[86] A. Repenning, A. Ioannidou, Agentcubes: Raising the ceiling of end-user development in education through incremental 3D, in: Visual Languages and Human-Centric Computing, VL/HCC'06, 2006, pp. 27–34, http://dx.doi.org/10.1109/VLHCC.2006.7.

[87] Leonel Morales Díaz, Laura S. Gaytán-Lugo, Lissette Fleck, Profiling styles of use in alice: Identifying patterns of use by observing participants in workshops with Alice, in: 2015 IEEE Blocks and beyond Workshop, Blocks and Beyond, 2015, pp. 19–24, http://dx.doi.org/10.1109/BLOCKS.2015.7368994.

[88] Arcadia | proceedings of the 2018 CHI conference on human factors in computing systems, 2018, Retrieved September 15, 2021 from https://dl-acm-org.proxy-um.researchport.umd.edu/doi/abs/10.1145/3173574.3173983?casa_token=4-pnNtIKskMAAAAA:Kxnm4dVDEX6_zFdNbIPvX7iythR8bgqGPr5LBnltwvjv1e6p-vS_nKoil6Nw7kUdTLvE2ZveGzXtmg.

[89] Hiroyuki Dekihara, Toru Ochi, Ryuji Miyazaki, Takuro Ozaki, Proposal of Practice Materials To Learn About Combining AI and IoT Based on Graphical Programming Language using Free and Open Source Software, Association for the Advancement of Computing in Education (AACE), 2019, pp. 1510–1515, Retrieved September 15, 2021 from https://www.learntechlib.org/primary/p/210167/.

[90] Sharon Lynn Chu, Francis Quek, Elizabeth Deuermeyer, Rachel Martin, From classroom-making to functional-making: A study in the development of making literacy, in: Proceedings of the 7th Annual Conference on Creativity and Fabrication in Education, ACM, Stanford CA USA, 2017, pp. 1–8, http://dx.doi.org/10.1145/3141798.3141802.

[91] Adin Baskoro Pratomo, Riza Satria Perdana, Arduviz, a visual programming IDE for arduino, in: 2017 International Conference on Data and Software Engineering, ICoDSE, 2017, pp. 1–6, http://dx.doi.org/10.1109/ICODSE.2017.8285871.

[92] Duks Koschitz, Eric Ramagosa, Beetle blocks: A new visual language for designers and makers, in: ACADIA//2016: POSTHUMAN FRONTIERS: Data, Designers, and Cognitive Machines [Proceedings of the 36th Annual Conference of the Association for Computer Aided Design in Architecture], ACADIA, vol. 2016, Ann Arbor, ISBN: 978-0-692-77095-5, 2016, pp. 130–139, CUMINCAD. Retrieved September 15, 2021 from http://papers.cumincad.org/cgi-bin/works/paper/acadia16_130.

[93] Charalampos Kyfonidis, Nektarios Moumoutzis, Stavros Christodoulakis, Block-C: A block-based programming teaching tool to facilitate introductory C programming courses, in: 2017 IEEE Global Engineering Education Conference, EDUCON, 2017, pp. 570–579, http://dx.doi.org/10.1109/EDUCON.2017.7942903.

[94] Erik Pasternak, Rachel Fenichel, Andrew N. Marshall, Tips for creating a block language with blockly, in: 2017 IEEE Blocks and beyond Workshop (B B), 2017, pp. 21–24, http://dx.doi.org/10.1109/BLOCKS.2017.8120404.

[95] Alexander G. Tamilias, Theodoros J. Themelis, Theodoros Karvounidis, Zacharenia Garofalaki, Dimitrios Kallergis, B@SE: Blocks for @rduino in the Students' educational process, in: 2017 IEEE Global Engineering Education Conference, EDUCON, 2017, pp. 910–915, http://dx.doi.org/10.1109/EDUCON.2017.7942956.

[96] Austin Cory Bart, Javier Tibau, Eli Tilevich, Clifford A. Shaffer, Dennis Kafura, Blockpy: An open access data-science environment for introductory programmers, Austin Cory Bart Javier Tibau Eli Tilevich Clifford A. Shaffer Dennis Kafura Computer 50 (5) (2017) 18–26, http://dx.doi.org/10.1109/MC.2017.132.

[97] Pedro Guillermo Feijóo-García, Sishun Wang, Ju Cai, Naga Polavarapu, Christina Gardner-McCune, Eric D. Ragan, Design and evaluation of a scaffolded block-based learning environment for hierarchical data structures, in: 2019 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2019, pp. 145–149, http://dx.doi.org/10.1109/VLHCC.2019.8818759.

[98] R. Benjamin Shapiro, Annie Kelly, Matthew Ahrens, Rebecca Fiebrink, Blockytalky: A physical and distributed computer music toolkit for kids, in: NIME, 2016, Retrieved September 15, 2021 from http://www.nime.org/proceedings/2016/nime2016_paper0084.pdf.

[99] F. Edward, Bots & amp (main) frames: Exploring the impact of tangible blocks and collaborative play in an educational programming game, in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, (2021) Association for Computing Machinery, New York, NY, USA, 2018, pp. 1–14, http://dx.doi.org/10.1145/3173574.3173840, Retrieved September 15, 2021 from.

[100] Michelle Friend, Michael Matthews, Victor Winter, Betty Love, Deanna Moisset, Ian Goodwin, Bricklayer: Elementary students learn math through programming and art, in: Proceedings of the 49th ACM Technical Symposium on Computer Science Education, SIGCSE '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 628–633, http://dx.doi.org/10.1145/3159450.3159515.

[101] David S. Touretzky, Computational thinking and mental models: From kodu to calypso, in: 2017 IEEE Blocks and beyond Workshop (B B), 2017, pp. 71–78, http://dx.doi.org/10.1109/BLOCKS.2017.8120416.

[102] Wolfgang Slany, A mobile visual programming system for Android smartphones and tablets, in: 2012 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2012, pp. 265–266, http://dx.doi.org/10.1109/VLHCC.2012.6344546.

[103] Nicholas Lytle, Yihuan Dong, Veronica Cateté, Alex Milliken, Amy Isvik, Tiffany Barnes, Position: Scaffolded coding activities afforded by block-based environments, in: 2019 IEEE Blocks and beyond Workshop (B B), 2019, pp. 5–7, http://dx.doi.org/10.1109/BB48857.2019.8941203.

[104] Marianthi Grizioti, Chronis Kynigos, Game modding for computational thinking: an integrated design approach, in: Proceedings of the 17th ACM Conference on Interaction Design and Children, IDC '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 687–692, http://dx.doi.org/10.1145/3202185.3210800.
[105] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C. Shepherd, Diana Franklin, Evaluating coblox: A comparative study of robotics programming environments for adult novices, in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1–12, http://dx.doi.org/10.1145/3173574.3173940.
[106] Cameron Wilson, What's up next for code.org? Computer 46 (8) (2013) 95–97, http://dx.doi.org/10.1109/MC.2013.292.
[107] Blayde Dill, Developing a blocked based language for the adafruit circuit playground: (abstract only), in: Proceedings of the 49th ACM Technical Symposium on Computer Science Education, SIGCSE '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 272, http://dx.doi.org/10.1145/3159450.3162335.
[108] Nath Tumlin, Teacher configurable coding challenges for block languages, in: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 783–784, http://dx.doi.org/10.1145/3017680.3022467.
[109] R. Vinayakumar, K.P. Soman, Pradeep Menon, CT-Blocks: Learning computational thinking by snapping blocks, in: 2018 9th International Conference on Computing, Communication and Networking Technologies, ICCCNT, 2018, http://dx.doi.org/10.1109/ICCCNT.2018.8493669.
[110] Juraj Vincur, Martin Konopka, Jozef Tvarozek, Martin Hoang, Pavol Navrat, Cubely: virtual reality block-based programming environment, in: Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology, VRST '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 1–2, http://dx.doi.org/10.1145/3139131.3141785.
[111] R. Vinayakumar, K.P. Soman, Pradeep Menon, DB-learn: Studying relational algebra concepts by snapping blocks, in: 2018 9th International Conference on Computing, Communication and Networking Technologies, ICCCNT, 2018, http://dx.doi.org/10.1109/ICCCNT.2018.8494181.
[112] Yasin N. Silva, Jaime Chon, DBsnap: Learning database queries by snapping blocks, in: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 179–184, http://dx.doi.org/10.1145/2676723.2677220.
[113] Yasin N. Silva, Anthony Nieuwenhuyse, Thomas G. Schenk, Alaura Symons, DBSnap++: creating data-driven programs by snapping blocks, in: Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2018, Association for Computing Machinery, New York, NY, USA, 2018, pp. 170–175, http://dx.doi.org/10.1145/3197091.3197114.
[114] Aaron Bauer, Eric Butler, Zoran Popović, Approaches for teaching computational thinking strategies in an educational game: A position paper, in: 2015 IEEE Blocks and beyond Workshop, Blocks and Beyond, 2015, pp. 121–123, http://dx.doi.org/10.1109/BLOCKS.2015.7369019.
[115] Eric Tilley, Jeff Gray, Dronely: A visual block programming language for the control of drones, in: Proceedings of the SouthEast Conference, ACM SE '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 208–211, http://dx.doi.org/10.1145/3077286.3077307.
[116] Francis Wyffels, Karel Bruneel, Peter Bertels, Michiel D'Haene, Wim Heirman, Tim Waegeman, A human-friendly way of programming robots, in: 5th International Workshop on Human-Friendly Robotics IEEE, 2012, (2012).
[117] Inseong Jeon, Ki-Sang Song, The effect of learning analytics system towards learner's computational thinking capabilities, in: Proceedings of the 2019 11th International Conference on Computer and Automation Engineering, ICCAE 2019, Association for Computing Machinery, New York, NY, USA, 2019, pp. 12–16, http://dx.doi.org/10.1145/3313991.3314017.
[118] Anke Brocker, Simon Voelker, Tony Zelun Zhang, Mathis Müller, Jan Borchers, Flowboard: A visual flow-based programming environment for embedded coding, in: Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems, CHI EA '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1–4, http://dx.doi.org/10.1145/3290607.3313247.
[119] Robert Holwerda, The freecoffee editor: Using natural language sentence structure to make blocks more readable, in: 2017 IEEE Blocks and beyond Workshop (B B), 2017, pp. 107–108, http://dx.doi.org/10.1109/BLOCKS.2017.8120425.
[120] Dragan Cvetković, Simulation and Gaming. BoD – Books on Demand, 2018.
[121] Rebecca Krosnick, Creating interactive user interfaces by demonstration using crowdsourcing, in: 2018 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2018, pp. 277–278, http://dx.doi.org/10.1109/VLHCC.2018.8506507.
[122] Caitlin Kelleher, John Maloney, Paul Medlock-Walton, Evan Patton, Daniel Wendel, Invited panel: The future of blocks programming, in: 2017 IEEE Blocks and beyond Workshop (B & B), IEEE, Raleigh, NC, 2017, pp. 99–101, http://dx.doi.org/10.1109/BLOCKS.2017.8120421.
[123] Yana Malysheva, An AST-based interface for composing and editing javascript on the phone, in: 2017 IEEE Blocks and beyond Workshop (B B), 2017, pp. 9–16, http://dx.doi.org/10.1109/BLOCKS.2017.8120402.
[124] David Weintrop, Uri Wilensky, Between a block and a typeface: Designing and evaluating hybrid programming environments, in: Proceedings of the 2017 Conference on Interaction Design and Children, IDC '17, ACM, New York, NY, USA, 2017, pp. 183–192, http://dx.doi.org/10.1145/3078072.3079715.
[125] Sandra Taylor, Wookhee Min, Bradford Mott, Andrew Emerson, Andy Smith, Eric Wiebe, James Lester, Position: IntelliBlox: A toolkit for integrating block-based programming into game-based learning environments, in: 2019 IEEE Blocks and beyond Workshop (B B), 2019, pp. 55–58, http://dx.doi.org/10.1109/BB48857.2019.8941222.
[126] Juan Haladjian, Katharina Bredies, Bernd Brügge, Interactex: an integrated development environment for smart textiles, in: Proceedings of the 2016 ACM International Symposium on Wearable Computers, ISWC '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 8–15, http://dx.doi.org/10.1145/2971763.2971776.
[127] Thomas W. Price, Tiffany Barnes, Showpiece: Isnap demonstration, in: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2017, pp. 339–340, http://dx.doi.org/10.1109/VLHCC.2017.8103499.
[128] Daniel Rough, Aaron Quigley, Jeeves - A visual programming environment for mobile experience sampling, in: 2015 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2015, pp. 121–129, http://dx.doi.org/10.1109/VLHCC.2015.7357206.
[129] David S. Touretzky, Teaching Kodu with physical manipulatives, ACM Inroads 5 (4) (2014) 44–51, http://dx.doi.org/10.1145/2684721.2684732.
[130] Rob Thompson, Teaching coding to learning-disabled children with Kokopelli's world, in: 2016 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2016, pp. 258–259, http://dx.doi.org/10.1109/VLHCC.2016.7739705.
[131] Charlotte Hill, Hilary A. Dwyer, Tim Martinez, Danielle Harlow, Diana Franklin, Floors and flexibility: Designing a programming environment for 4th-6th grade classrooms, in: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, ACM, Kansas City Missouri USA, 2015, pp. 546–551, http://dx.doi.org/10.1145/2676723.2677275.
[132] Mark Rollins, Beginning LEGO MINDSTORMS EV3, A Press, Berkeley, CA, 2014, http://dx.doi.org/10.1007/978-1-4302-6437-8.
[133] Andrew Begel, Mitchel Resnick, Logoblocks: A graphical programming language for interacting with the world, 2000, Retrieved September 16, 2021 from https://www.semanticscholar.org/paper/LogoBlocks%3A-A-Graphical-Programming-Language-for-Begel-Resnick/78de01596353f8f34ba5ebedfea17e51f3ea3fad.
[134] Caitlin Kelleher, Looking glass, in: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 271, http://dx.doi.org/10.1145/2676723.2691873.
[135] Chris Johnson, Peter Bui, Blocks in, blocks out: A language for 3D models, in: 2015 IEEE Blocks and beyond Workshop, Blocks and Beyond, 2015, pp. 77–82, http://dx.doi.org/10.1109/BLOCKS.2015.7369007.
[136] Thomas Ball, Abhijith Chatra, Peli de Halleux, Steve Hodges, Michał Moskal, Jacqueline Russell, Microsoft MakeCode: embedded programming for education, in blocks and TypeScript, in: Proceedings of the 2019 ACM SIGPLAN Symposium on SPLASH-E, SPLASH-E 2019, Association for Computing Machinery, New York, NY, USA, 2019, pp. 7–12, http://dx.doi.org/10.1145/3358711.3361630.
[137] R. Vinayakumar, K.P. Soman, Pradeep Menon, Map-blocks: Playing with online data and infuse to think in a computational way, in: 2018 9th International Conference on Computing, Communication and Networking Technologies, ICCCNT, 2018, http://dx.doi.org/10.1109/ICCCNT.2018.8493700.
[138] Dave Krebs, Alexander Conrad, Jingtao Wang, Combining visual block programming and graph manipulation for clinical alert rule building, in: CHI '12 Extended Abstracts on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA, 2012, pp. 2453–2458, http://dx.doi.org/10.1145/2212776.2223818.
[139] L.H. Peng, M.-H. Bai, I. Siswanto, A study of learning motivation of senior high schools by applying unity and mblock on programming languages courses, J. Phys.: Conf. Ser 1456 (2020) (2020) 012037, http://dx.doi.org/10.1088/1742-6596/1456/1/012037.
[140] Arjun Rao, Ayush Bihani, Mydhili Nair, Milo: A visual programming environment for data science education, in: 2018 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2018, pp. 211–215, http://dx.doi.org/10.1109/VLHCC.2018.8506504.
[141] Matthew Bajzek, Heather Bort, Omokolade Hunpatin, Luke Mivshek, Tyler Much, Casey O'Hare, Dennis Brylow, MUzECS: Embedded blocks for exploring computer science, in: 2015 IEEE Blocks and beyond Workshop, Blocks and Beyond, 2015, pp. 127–132, http://dx.doi.org/10.1109/BLOCKS.2015.7369021.
[142] Brian Broll, Akos Lédeczi, Peter Volgyesi, Janos Sallai, Miklos Maroti, Alexia Carrillo, Stephanie L. Weeden-Wright, Chris Vanags, Joshua D. Swartz, Melvin Lu, A visual programming environment for learning distributed programming, in: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 81–86, http://dx.doi.org/10.1145/3017680.3017741.
[143] Izabel C. Olson, Michael S. Horn, Modeling on the table: agent-based modeling in elementary school with NetTango, in: Proceedings of the 10th International Conference on Interaction Design and Children, IDC '11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 189–192, http://dx.doi.org/10.1145/1999030.1999058.

[144] Chris S. Crawford, Juan E. Gilbert, NeuroBlock: A block-based programming approach to neurofeedback application development, in: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2017, pp. 303–307, http://dx.doi.org/10.1109/VLHCC.2017.8103483.

[145] Ricarose Vallarta Roque, OpenBlocks: An Extendable Framework for Graphical Block Programming Systems (Master's Thesis), Ricarose Vallarta Roque Massachusetts Institute of Technology, 2007.

[146] Pao-Nan Chou, Little engineers: Young children's learning patterns in an educational robotics project, in: 2018 World Engineering Education Forum - Global Engineering Deans Council, WEEF-GEDC, 2018, pp. 1–5, http://dx.doi.org/10.1109/WEEF-GEDC.2018.8629609.

[147] William Robinson, From scratch to patch: Easing the blocks-text transition, in: Proceedings of the 11th Workshop in Primary and Secondary Computing Education, ACM, Münster Germany, 2016, pp. 96–99, http://dx.doi.org/10.1145/2978249.2978265.

[148] Franklyn Turbak, Smaranda Sandu, Olivia Kotsopoulos, Emily Erdman, Erin Davis, Karishma Chadha, Blocks languages for creating tangible artifacts, in: 2012 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2012, pp. 137–144, http://dx.doi.org/10.1109/VLHCC.2012.6344500.

[149] Andrew Emerson, Andy Smith, Fernando J. Rodriguez, Eric N. Wiebe, Bradford W. Mott, Kristy Elizabeth Boyer, James C. Lester, Cluster-based analysis of novice coding misconceptions in block-based programming, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, SIGCSE '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 825–831, http://dx.doi.org/10.1145/3328778.3366924.

[150] Peeratham Techapalokul, Eli Tilevich, QIS: Automated refactoring for scratch, in: 2019 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2019, pp. 265–266, http://dx.doi.org/10.1109/VLHCC.2019.8818906.

[151] Ian Arawjo, Cheng-Yao Wang, Andrew C. Myers, Erik Andersen, Teaching programming with gamified semantics, in: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA, 2017, pp. 4911–4923, http://dx.doi.org/10.1145/3025453.3025711, Retrieved September 16, 2021 from.

[152] Nicholas Lytle, Jennifer Echavarria, Joshua Sosa, Thomas W. Price, Resource rush: Towards an open-ended programming game, in: 2019 IEEE Blocks and beyond Workshop (B B), 2019, pp. 91–93, http://dx.doi.org/10.1109/BB48857.2019.8941215.

[153] Tomáš Effenberger, Radek Pelánek, Towards making block-based programming activities adaptive, in: Proceedings of the Fifth Annual ACM Conference on Learning At Scale, L@S '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1–4, http://dx.doi.org/10.1145/3231644.3231670.

[154] Sayamindu Dasgupta, Block-based programming with Scratch community data: A position paper, in: 2015 IEEE Blocks and beyond Workshop, Blocks and Beyond, 2015, pp. 97–98, http://dx.doi.org/10.1109/BLOCKS.2015.7369011.

[155] Louise P. Flannery, Brian Silverman, Elizabeth R. Kazakoff, Marina Umaschi Bers, Paula Bontá, Mitchel Resnick, Designing scratchjr: support for early childhood learning through computer programming, in: Proceedings of the 12th International Conference on Interaction Design and Children, IDC '13, Association for Computing Machinery, New York, NY, USA, 2013, pp. 1–10, http://dx.doi.org/10.1145/2485760.2485785.

[156] Nayeon Bak, Byeong-Mo Chang, Kwanghoon Choi, Smart block: A visual block language and its programming environment for IoT, J. Comput. Lang. 60 (2020) (2020) 100999, http://dx.doi.org/10.1016/j.cola.2020.100999.

[157] Annette Feng, Eli Tilevich, Wu-chun Feng, Block-based programming abstractions for explicit parallel computing, in: 2015 IEEE Blocks and beyond Workshop, Blocks and Beyond, 2015, pp. 71–75, http://dx.doi.org/10.1109/BLOCKS.2015.7369006.

[158] Jack Atherton, Paulo Blikstein, Sonification blocks: A block-based programming environment for embodied data sonification, in: Proceedings of the 2017 Conference on Interaction Design and Children, IDC '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 733–736, http://dx.doi.org/10.1145/3078072.3091992.

[159] Alan Kay, Squeak etoys, children & learning, 2005, Retrieved October 17, 2014 from http://www.squeakland.org/content/articles/attach/etoys_n_learning.pdf.

[160] Andrew Begel, Eric Klopfer, Starlogo TNG: An introduction to game development, J. E-Learning (2007) (2007).

[161] Eric Klopfer, Ricarose Roque, Wendy Huang, Daniel Wendel, Hal Scheintaub, The simulation cycle: combining games, simulations, engineering and science using StarLogo TNG, E-Learning 6 (1) (2009) 71, http://dx.doi.org/10.2304/elea.2009.6.1.71.

[162] Varsha. Koushik, Darren. Guinness, Shaun K. Kane, Storyblocks: A tangible programming game to create accessible audio stories, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA, 2019, http://dx.doi.org/10.1145/3290605.3300722, Retrieved September 16, 2021 from.

[163] Terrell Glenn, Ananya Ipsita, Caleb Carithers, Kylie Peppler, Karthik Ramani, Storymakar: Bringing stories to life with an augmented reality & amp physical prototyping toolkit for youth, in: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–14, Retrieved September 16, 2021 from.

[164] Karishma Chadha, Franklyn Turbak, Improving app inventor usability via conversion between blocks and text.

[165] Leonel Morales Díaz, Carlos Morales Hernández, Alex Viau Ortiz, Tinkercad and codeblocks in a summer course: an attempt to explain observed engagement and enthusiasm, in: 2019 IEEE Blocks and beyond Workshop (B B), 2019, pp. 43–47, http://dx.doi.org/10.1109/BB48857.2019.8941211.

[166] Thomas Ball, Sebastian Burckhardt, Jonathan de Halleux, Michal Moskal, Jonathan Protzenko, Nikolai Tillmann, Beyond open source: The touch develop cloud-based integrated development environment, in: 2015 2nd ACM International Conference on Mobile Software Engineering and Systems, 2015, pp. 83–93, http://dx.doi.org/10.1109/MobileSoft.2015.20.

[167] Chonnuttida Koracharkornradt, Tuk Tuk: A block-based programming game, in: Proceedings of the 2017 Conference on Interaction Design and Children, IDC '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 725–728, http://dx.doi.org/10.1145/3078072.3091990.

[168] Amon Millner, Allison Busa, Bryanne Leeming, Promoting unruly programming with random blocks and physical play, in: 2017 IEEE Blocks and beyond Workshop (B B), 2017, pp. 113–114, http://dx.doi.org/10.1109/BLOCKS.2017.8120428.

[169] Ruanqianqian Huang, Franklyn Turbak, A design for bidirectional conversion between blocks and text for app inventor, in: 2019 IEEE Blocks and beyond Workshop (B B), 2019, pp. 87–89, http://dx.doi.org/10.1109/BB48857.2019.8941197.

[170] Innovation First Int., Vexcode VR, 2021, Retrieved September 23, 2021 from https://vr.vex.com/.

[171] Bas Jansen, Felienne Hermans, XLBlocks: a block-based formula editor for spreadsheet formulas, in: 2019 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, 2019, pp. 55–63, http://dx.doi.org/10.1109/VLHCC.2019.8818748.