StoryBlocks: A Tangible Programming Game to Create Accessible Audio Stories

Varsha Koushik, Darren Guinness, and Shaun K. Kane
Department of Computer Science
University of Colorado Boulder
Boulder, CO, USA
{varsha.koushik, darren.guinness, shaun.kane}@colorado.edu

ABSTRACT

Block-based programming languages can support novice programmers through features such as simplified code syntax and user-friendly libraries. However, most block-based programming languages are highly visual, which makes them inaccessible to blind and visually impaired students. To address the inaccessibility of block-based languages, we introduce StoryBlocks, a tangible block-based game that enables blind programmers to learn basic programming concepts by creating audio stories. In this paper, we document the design of StoryBlocks and report on a series of design activities with groups of teachers, Braille experts, and students. Participants in our design sessions worked together to create accessible stories, and their feedback offers insights for the future development of accessible, tangible programming tools.

CCS CONCEPTS

• Human-centered computing → Accessibility → Accessibility technologies;

KEYWORDS

Computer science education; tangible user interfaces; audio interfaces; storytelling; cross-ability collaboration.

ACM Reference format:
Varsha Koushik, Darren Guinness, and Shaun K. Kane. 2019. StoryBlocks: A Tangible Programming Game to Create Accessible Audio Stories. In *2019 CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland, UK.* ACM, NY, NY, USA. Paper492, 12 pages. https://doi.org/10.1145/3290605.3300722

CHI 2019, May 4-9, 2019, Glasgow, Scotland, UK.

ACM ISBN 978-1-4503-5970-2/19/05...$15.00.
DOI: https://doi.org/10.1145/3290605.3300722

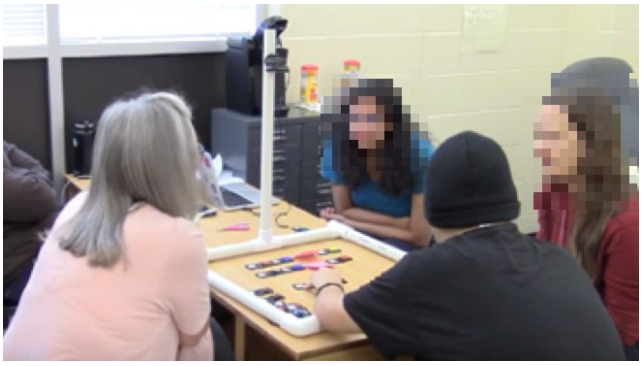

Figure 1. StoryBlocks is a tangible programming game in which users create audio stories by combining code blocks. Here a visually impaired high school student creates a story with assistance from teachers.

1 INTRODUCTION

Computer science is increasingly becoming a part of mainstream K-12 education. Learning how to create computer programs can empower students to pursue work in computer science and other STEM fields. Even for students who do not go on to pursue computing careers, learning computer science and can support computational thinking, which itself can be applied to many different domains [34]. However, computer science students may encounter a variety of barriers in their studies. As a result, computer science educational programs often suffer a high attrition rate [4] and may disproportionately drive out women and other marginalized students [15].

Students with disabilities may experience many barriers in learning computer science due to inaccessible programming tools, unprepared teachers, and other concerns [27]. Supporting students with disabilities in computing careers is crucial, not only to support equal access to education and work, but because developing computational skills can help empower people to use, adapt, and create assistive technologies [27]. A key challenge is enabling students to work collaboratively

with teachers and other students; while group activities can encourage some students to learn computer science [40], people with disabilities who use assistive technologies often encounter barriers when using their assistive technology during collaborative activities [7]. A second challenge is to make block-based programming languages accessible to all students [34]. While block-based programs offer advantages to novice programmers, they generally are not accessible to students with vision impairments [30].

To address the challenge of creating an accessible and educational programming environment, we introduce StoryBlocks (Figure 1), a programming game that uses tangible blocks to represent code. These blocks can be assembled to program simple audio games and stories. In this paper, we describe the design and development of StoryBlocks, the design of the tangible blocks and workspace, and format of the audio story output. To gather feedback about our prototype, we conducted six design sessions with blind and visually impaired students, teachers of the visually impaired (TVIs), Braille experts, and other educational staff. The contributions of this paper include documentation of the StoryBlocks prototype, user feedback from our design sessions, and example stories created by our design teams.

2 RELATED WORK

This work draws from best practices in designing accessible technology for blind and visually impaired people [31], as well as research on designing accessible programming environments, block-based languages, tangible computing environments, and storytelling-based programming tools.

2.1 Accessible Programming Environments

Blind and visually impaired programmers face many challenges while programming, including navigating code structure, identifying bugs in code, and performing visual tasks such as graphical user interface development [1]. For expert coders, tools such as Emacspeak [33] and StructJumper [2] use audio cues and navigation shortcuts to make it easier to read and navigate through code. However, these tools address the needs of expert programmers, and are not suited to novices.

Other approaches to making coding accessible include programming languages that are optimized for both blind and sighted coders. Quorum [38] is a text-based programming language that enables programmers to create complex graphical applications, video and audio games, and other programs. Quorum is increasingly used to support accessibility in K-12 computer science education [28]. APL [36] is an audio programming language that allows blind and visually impaired programmers to create simple games and applications that use recorded sounds and text-to-speech. These tools provide accessible entry points to computer science but require learners to be skilled users of computer software and any necessary assistive technologies, which may leave out some children.

Researchers and educators have developed curricula that support blind and visually impaired children in learning mainstream programming languages, often paired with some application library, such as teaching Java through programming robots [29], chat bots [5], 3D printing [21], and audio games [22]. These tools have the benefit of supporting mainstream programming languages, but may carry the complexities of these languages, which can present challenges for novice learners. Our present work focuses on providing a simplified introduction to programming for blind and visually impaired learners.

2.2 Block-Based Programming Languages

Block-based programming languages such as Scratch [34] have become popular means for introducing children to computer science, due in part to beginner-friendly features such as visual code blocks that suggest proper syntax, the ability to easily create games, animations, and other visual media, and the ability to easily share and remix programs online. While Scratch is perhaps the most popular block-based language, with millions of users[1], many other programming environments, such as Blockly[2] and code.org [23], also use blocks. Blockly, an open source library, has been used to create a variety of block-based programming tools, such as BlockyTalky [14], which allows children to program networked mobile devices and musical instruments; ArduBlockly[3], which enables block-based programming of microcontrollers; and OzoBlockly[4], which enables children to control robots with block-based programs. Unfortunately, neither Scratch nor Blockly are accessible to screen reader users, and both rely heavily on visual presentation for both creating code and observing its output. StoryBlocks offers an alternative approach to

[1] scratch.mit.edu/statistics
[2] developers.google.com/blockly
[3] github.com/carlosperate/ardublockly
[4] ozoblockly.com
[4] algobrix.com
[5] playosmo.com

block-based programming that provides equal access to blind and sighted learners.

Some block-based programming environments have begun to address accessibility issues [30]. PseudoBlocks [26] enables blind programmers to navigate block-based programs via keyboard input and speech output. Blocks4All [32] enables blind users to explore block-based programs by touching a touch screen and receiving audio feedback. Project Torino [39] is an accessible programming environment that uses custom hardware "blocks" that can be connected together via cables to create audio-based programs. Like Torino, StoryBlocks uses physical components to create programs; however, StoryBlocks uses low-cost components and computer vision rather than custom electronics. StoryBlocks also focuses on audio stories as an output format. We consider these to be complementary approaches to a similar problem.

2.3 Tangible Programming Toolkits

In addition to supporting accessibility, tangible programming tools may help learners engage with computing concepts and may support collaborative programming [16]. Many current tangible programming toolkits support physical computing projects, as in roBlocks [35], littleBits [3], Project Bloks [6] and Algobrix[4]. These toolkits have physical blocks that may sometimes be accessible to blind and visually impaired learners, but they are rarely designed with accessibility in mind.

Other tangible programming toolkits, such as Tern [15], Strawbies [18], and Osmo[5], have focused on creating programming-like games in which learners compose programs by assembling a list of instructions, which then control an on-screen character or simulation. These systems are usually comprised of passive blocks, made of wood or cardboard, with visual tags on each block. A computer's camera reads the series of blocks and executes the program. Often the shape of the block itself demonstrates its function, such as by having an if-block that demonstrates a branching "Y" shape. While these toolkits feature tangible blocks that might be understandable to a blind learner, they typically create on-screen programs that are inaccessible. StoryBlocks follows a similar approach of using tangible blocks as input, but it has been designed such that both creating and playing back programs is accessible.

2.4 Storytelling in Computer Science Education

A significant challenge in designing accessible programming tools is creating program output that can be accessible and interesting to blind learners. In StoryBlocks, we use *audio stories* as an output format that can be accessible to both blind and sighted users and could even be easily translated into text for deaf learners and others.

Using stories as program output has been employed previously, as many stories can be described as a series of objects and actions that correspond to program code. Strawbies [18] and Alice [25] enable programmers to control on-screen characters through code. Ryokai et al. [35] developed a robotic programming environment in which children created interactive stories by specifying a robot character's behaviors. Storytelling Alice [25] and Looking Glass [24] enable novice programmers to control 3D characters in their code and to create stories that can be played back or saved as a movie. These systems have largely focused on creating visual stories, while StoryBlocks explores how to create engaging and fun audio stories through code.

2.5 Design Methods for Accessible User Interfaces

Designing technology for and with people with vision impairments presents its own set of challenges, as activities such as sketching and storyboarding must be supplemented with a more accessible alternative. Participatory workshops are often used as a way to explore the design non-visual and tangible systems [9, 39, 41]. We created a physical StoryBlocks prototype as a foundation for a series of collaborative design workshops.

Another common approach is to include other stakeholders in the research activity, such as friends and family members, who may have a unique insight into the accessibility barriers encountered by an individual [7, 42]. We conducted our formative evaluation of StoryBlocks with selected groups of students, teachers, and assistive technology specialists.

3 STORYBLOCKS

To explore the design space of tangible programming tools for blind children, we developed the StoryBlocks prototype, which involved building the system software, designing the tactile blocks, and creating demonstration characters and voices for users' programs.

3.1 System Overview

The StoryBlocks system comprises a set of tangible blocks, an augmented workspace, and software for interpreting and playing back users' story programs.

Tangible Blocks. Each block represents one component of an interactive story, including characters, actions, and program control statements. Each block features a tactile symbol and a visual tag. Blocks do not contain any electronics, enabling them to be easily fabricated and modified. Each block is approximately 3.5 × 1 inch in size.

Workspace. Programs are created in a designated workspace area. The workspace is 24 x 24 inches in size. Tactile borders indicate the boundaries of the workspace. The prototype workspace is constructed from 1-inch PVC pipe (Figure 1). Images are captured by an overhead webcam (currently a Logitech C615), connected to a laptop.

Software. StoryBlocks programs are interpreted by a custom Python program on a nearby laptop. Visual tags are recognized using the reacTIVision library [20]. The user scans a program by pressing a key on the laptop's keyboard; the program tags are read, the program is parsed, and the laptop produces audio output that corresponds to the user's code.

3.2 Design of Tactile Features

StoryBlocks uses tangible blocks to represent programs. Because StoryBlocks may be used by blind learners, it was important that the tactile design of the blocks represent their function as clearly as possible. The initial set of blocks used in our prototype is shown in Figure 2.

Figure 2. Blocks used in the initial StoryBlocks prototype include characters (mouse, turtle, cat, snake, cheese), actions (explode, talk, run, dance, eat), and control structures (repeat, branch).

In designing the tangible blocks, we followed the following design criteria, derived from prior work in designing laser-cut and 3D-printed tactile graphics [9,12,13,25]:

Communicate Type and Allowable Connections. The physical design of each block needs to communicate its overall type (*e.g.,* character, action, or command), and which blocks can be connected to other blocks. In StoryBlocks, the shape of the program block communicates the block's type—character blocks have square edges, actions have rounded edges, and control blocks have diamond-shaped edges—and the design of the notches and tabs along the edge of each block communicates valid connections between blocks (Figure 3). For blocks that connect directly to other blocks, specifically the *if-else* blocks, we use a physical string to indicate the connection between the original block and the two end points.

Figure 3. Blocks represent the phrase "The mouse eats cheese." The tabs at the top and bottom indicate that more lines of code may be added to the program.

Communicate Block Identity by Touch. Blocks should be differentiable using tactile information only, and without the need for Braille labels, as not all blind children can read Braille and because this toolkit should also be usable by sighted collaborators. A user should be able to quickly identify a block in their hand, find a specific block from within a pile of unsorted blocks, and count how many of each block are available. In StoryBlocks, each block features a tactile symbol with a distinctive shape. For example, the cat character features pointy ears, while the mouse character features round ears.

Our initial set of tactile symbols was based on the open source EmojiOne[5] characters. We chose a set of symbols with distinct shapes, flattened their images to a single color, and removed some unnecessary visual features, such as the holes in the cheese symbol. This approach enabled us to quickly create distinct tactile symbols and enables us to easily add more symbols in the future.

Use Widely Available Materials. While including electronics inside blocks can enable rich interactions, as in Torino [39], we designed the components of StoryBlocks to be inexpensive and easy to fabricate [17]. The current prototype uses blocks that were laser cut and hand painted, although they could also be hand-cut from wood or cardboard. A full set of blocks can be produced on a laser cutter in about 15 minutes and requires about US$30

[5] www.emojione.com/emoji/v2

in materials. As StoryBlocks programs run on a centralized computer, the system can scale to support multiple student groups in a classroom simply by fabricating more blocks.

Support Customization and Extensibility. As the StoryBlocks software tracks a visual tag, users can create blocks of different sizes, colors, or materials without disrupting the program functions. While the set of StoryBlocks characters and actions is currently fixed, future versions could enable the addition of new characters by end users, similar to how Bonk [22] enabled programmers to create custom voice characters in their audio games.

Support Diverse Stories and Characters. We intend StoryBlocks to be most appropriate for learners from age 8 to 18, but we designed it with the hope that it would be welcoming to the widest possible group. As we developed one set of starter blocks for all users, we chose characters and actions that would be of interest to learners of all ages and genders. We initially explored characters from fairy tale stories, but later focused on a series of animal characters that could represent different cultural perspectives, gender roles, and personality types.

3.3 Story Playback

A key design goal for StoryBlocks is to enable learners to create audio stories that are interesting and fun to listen to so that they are rewarded for creating programs and motivated to share their creations with others.

In the current prototype, the user initiates a story by pressing a key on the laptop keyboard. The story is read from start to finish, line by line. StoryBlocks system reads each block from left to right, sorts the blocks based on their x and y coordinates, and generates a word associated with each block. StoryBlocks then combines all the words into a sentence and adds different voices to characters and sound effects to actions. For each line, the system narrates a story-like description of the blocks. For example, the blocks *Cat* → *Run* → *Mouse* might be read as "the cat runs after the mouse." In addition, the specific characters and actions customize the voice used when reading this sentence (in this case, it might be read in the cat's high-pitched voice) and the accompanying sound effects (in this case, the sound of a cartoon character running).

Currently, user input during playback is limited to *if-else* blocks; for these blocks, the system reads the options and the player presses a key on the keyboard to choose a path.

4 TESTING WITH TEACHERS AND STUDENTS

To explore the StoryBlocks concept and our initial design prototype, we conducted a formative study with several groups of students, teachers, and educational aides working together. We chose to test StoryBlocks in a collaborative school environment as we imagine this to be a common use of StoryBlocks in the future.

Our study was motivated by the following research questions:

- *RQ1:* How can students and teachers use StoryBlocks to create audio stories?
- *RQ2:* How do students and teacher groups collaborate to create audio stories using StoryBlocks?
- *RQ3:* What kinds of stories do people want to create using StoryBlocks?
- *RQ4:* How can we improve future versions of StoryBlocks?

4.1 Participants

We conducted six design sessions. Each session included a group that contained a combination of students, teachers, and other educational staff. We recruited study participants via our state's coordinator of teachers of the visually impaired (TVIs), and through our ongoing partnerships with local schools.

We recruited 16 participants total (12 female), ranging in age from 11 to 65 years old. Our participants included five blind and low vision students (middle school and high school), eight TVIs, and three staff members—two Braillists (Braille transcriptionists) and one preschool staff member. Groups were made up of individuals who regularly worked with each other at school. One participant, *e*, participated in two sessions; everyone else participated in one session only. Table 1 describes the participants.

Most participants had no prior computer science experience, except for *f*, who previously earned a computer science degree; *j*, who once took a robotics course; and *n* and *o*, who had participated in computer literacy training.

4.2 Apparatus

We used the prototype version of StoryBlocks described previously. We fabricated a set of 31 program blocks, which were placed in a box at the start of the session. The blocks were as follows: cat and mouse (5 each); dragon and turtle (3 each); cheese, dance, eat, explode, run, say (2 each); branch (2); loop (1).

4.3 Procedure

Study sessions took place in classrooms at our participants' schools or at our university research lab. The first author facilitated all of the study sessions. Each participant completed a consent or assent process before participating in the research and each participant was compensated for their time. Each session lasted approximately 90 minutes. All sessions were audio and video recorded. Each session proceeded as follows:

Introduction (15 minutes). The researcher introduced the StoryBlocks concept and the prototype hardware. Participants touched the workspace frame. The researcher introduced and passed around each program block. To demonstrate the physical connections between blocks, the researcher asked the group to choose a character and action block, and to connect them to create their first program.

Next, the researcher constructed and demonstrated three simple stories, one using only characters and actions and the other two introducing loops and branches, respectively. During this demonstration, participants were encouraged to touch and interact with the program blocks.

Recreating a Story (5 minutes). The researcher described a simple story, which was either "Cat chases mouse and cat explodes," or "Dragon says hi and dragon dances with turtle," and asked the group to work together to recreate the story using StoryBlocks. The group could play back their audio story to debug their program. The group was instructed to tell the researcher when they had completed the task.

Creating a New Story (15 minutes). The researcher instructed the group to work together to program a new story of their choice. The group was allowed to use any of the provided blocks and could also create new blocks using pieces of paper; we chose this open-ended approach to enable the groups to express their creativity and to identify gaps in our initial set of blocks.

After creating their story, the group was instructed to act it out as they would like it to be performed by the system; we chose this approach so that groups could demonstrate the types of story playback they would want to hear.

Group Discussion (30 minutes). The researcher interviewed the group members about their experience using StoryBlocks, their feedback about the current prototype, and their thoughts on future directions for StoryBlocks.

5 FINDINGS

We describe how our participants created stories using StoryBlocks and summarize participants' feedback about the prototype system.

5.1 Creating and Assembling Programs

In general, participant groups were able to use StoryBlocks to construct story programs without much confusion. For the *Recreating a Story* activity, five out of six groups were able to correctly reproduce the two-line example program; the sixth group misidentified a tangible block and added the wrong block to their program. While we did not define success or failure in the *Creating a New Story* activity, as participants were allowed to bend the rules to add new blocks and new functionality, we found that each group was able to plan, implement, and test a story.

Each group chose their own approach to creating and programming their original stories. Two of the six groups discussed the details of the story plot and characters before assembling their code, while the other groups followed a more iterative approach, creating one line of the program at a time, and then discussing what to do next.

We observed three recurring problems that affected several of the participant groups. First, blind and visually impaired participants were sometimes unable to identify individual blocks based on the tactile symbol alone. In these cases, their sighted partners would verify the identity of the block.

Second, our participant groups sometimes had difficulty using some of the more advanced programming features, particularly conditional branching. Two of the six groups had difficulty using branching in their story and required extra help from the research team to do so.

Third, participant groups sometimes created programs that did not follow our initial syntactic rules. For example, one group wrote their entire program as a continuing line, rather than placing each statement on its own line (Figure 4D). In that specific case, we might expect that their program would result in valid output, much as a C program can be written on one line so long as each instruction is delimited with a semicolon, but the divergence in the spatial layout of groups' program suggests some confusion about the rules of StoryBlocks programs.

Table 1. Groups in the study and tasks performed by each group member.

Group	Size	Participants	Location	Task: Planning	Creating	Organizing	Assembling	Performing
A	4	TVIs (a, b, c); Visually impaired preschool staff (d)	School	TVI (a), Visually impaired preschool staff (d)	TVIs (b, c)	—	TVIs (b, c)	TVIs (a, b, c); Visually impaired preschool staff (d)
B	3	TVI (e), Braillist (f), Blind high school student (g)	University lab	Braillist (f), Blind high school student (g)	TVI (e)	TVI (e)	Blind high school student (g)	Blind high school student (g)
C	3	TVI (h), Braillist (i), Low vision high school student (j)	School	TVI (h), low vision high school student (j)	Braillist (i)	Braillist (i)	Low vision high school student (j)	TVI (h)
D	2	TVIs (k, l)	School	TVIs (k, l)	TVI (k)	TVI (k)	TVI (l)	TVIs (k, l)
E	3	TVI (m), Blind middle school student (n), Low vision middle school student (o)	University lab	Blind middle school student (n), Low vision middle school student (o), TVI (m)	TVI (m)	Blind middle school student (n), Low vision middle school student (o)	Blind middle school student (n), Low vision middle school student (o)	TVI (m), Low vision middle school student (o), Blind middle school student (n)
F	2	TVI (e), Low vision middle school student (p)	School	TVI (e), Low vision middle school student (p)	TVI (e)	TVI (e)	Low vision middle school student (p)	TVI (e), Low vision middle school student (p)

Despite these challenges, our participant groups were able to complete the study tasks (sometimes with assistance) and remained engaged throughout the process.

5.2 Working in Groups

In general, each group seemed to enjoy the process of creating programs using StoryBlocks: participants talked with one another throughout the process and laughed at the amusing stories that they created. We observed that participants often smiled when they heard their first audio story. After successfully completing the *Recreating a Story* activity, some groups clapped or cheered.

We were interested to learn how participant would divide up tasks within their groups. While we asked our participant groups to work together during the study, we did not specify how they should divide up their tasks. We were therefore able to observe how groups delegated tasks among themselves.

In reviewing the activities after the study sessions, we identified a set of tasks that were common across the different groups: *planning*, which involved discussing ideas and making suggestions for programs (but not manipulating blocks); *creating* new blocks via sticky notes; *organizing* and sorting blocks; *assembling* code blocks; and *performing* the completed story. Table 1 shows the tasks performed by each group member.

Group members often shared tasks by taking turns or by dividing a task into smaller parts. For example, the two students in Group E took turns assembling code blocks, and later divided the work of performing the story so that one student read dialog and the other made sound effects. Groups also divided tasks by ability. For example, as sorting and organizing through blocks was sometimes aided by visual search, this task was often performed by a sighted teacher or staff member, although blind and visually impaired students were usually able to perform this task.

True to their roles at school, the teachers and staff often took on the task of keeping students engaged in the activity. Teachers and staff often checked in with the students throughout the activity, asking what the group should do next and verifying that the students were following along. Teachers also sometimes made adaptations to the work to increase accessibility; for example, the teacher in Group B noted that the sticky notes they were using were not accessible to the whole group and began adding Braille labels to those notes (Figure 4C).

Despite the groups' efforts to work together, in two of the groups, A and E, we observed that the visually impaired students sometimes lost track of the ongoing conversation. In both groups, this occurred when the sighted team members began discussing some visual aspect of the task without taking the time to include the other members. However, in each case, this problem was noticed, and the students were brought back into the conversation.

5.3 Creating Original Stories

Each group created at least one story based on a topic of their choice. Most of the groups created stories that built upon the existing set of character and action blocks, although one group, A, created a story about events at their school. Although the processes by which each group created their story varied, all of the groups engaged in computational thinking practices like incremental design of their program, testing the program to see whether it matched their expectations, and debugging when the program differed from their expectation [34].

During the *Creating a New* Story activity, groups were given the option of creating new blocks. This approach allowed us to identify what blocks the groups might wish to create, and to explore how closely the group adhered to the existing program syntax. Each group created at least one new block during the process, writing its name on a sticky note (and, in one case, adding Braille labels) and placing it in the workspace.

The most common types of new blocks were additional characters and actions, including *teacher*, *donut*, *coffee*, *revenge*, and *dance competition* (Figure 4A and 4B). However, some groups created blocks that represented new language constructs entirely. Five of the six groups created new blocks that represented some new language feature, including adding numerical constants, string constants, and sound effects. Finally, some groups added blocks that altered the structure of the language itself, adding linguistic features such as prepositions ("into the lake", "with the cheese"). Group E restructured their program to look like a prose paragraph, with a constant row of phrases, rather than as a stacked sequence of actions (Figure 4D).

These user-created blocks provide insight about how to extend the StoryBlocks language in both simple ways (*e.g.*, adding new characters) and complex ways (*e.g.*, adding string operations). These possible changes are explored further below.

5.4 Participant Feedback

Following the programming activities, we asked the participant groups to offer feedback on the current version of StoryBlocks and suggestions for future versions.

Overall, our participants were enthusiastic about their experience with StoryBlocks. During the post-activity interview, 15 out of 16 participants expressed positive feelings about the StoryBlocks prototype and its potential for the future. Participants spoke positively about the benefits of tangible blocks for diverse learners and noted that StoryBlocks might be useful to both blind and sighted children. Regarding the connection between StoryBlocks and programming, one teacher, who had some previous computer science experience, said that the storytelling approach helped her to understand programming, and that StoryBlocks "*makes sense to me more than my CS class 20 years ago.*"

Participants offered a number of suggestions for extending the StoryBlocks system. Participants requested additional blocks to represent new characters and actions, as well as more voices and sound effects. Participants also requested a manual or other training materials to help teachers, parents, and students learn how to use StoryBlocks. One common request, voiced by multiple participants, was to incorporate text strings and Braille characters into StoryBlocks; doing so would expand the possibilities of what could be created using StoryBlocks, and might even encourage students to learn Braille.

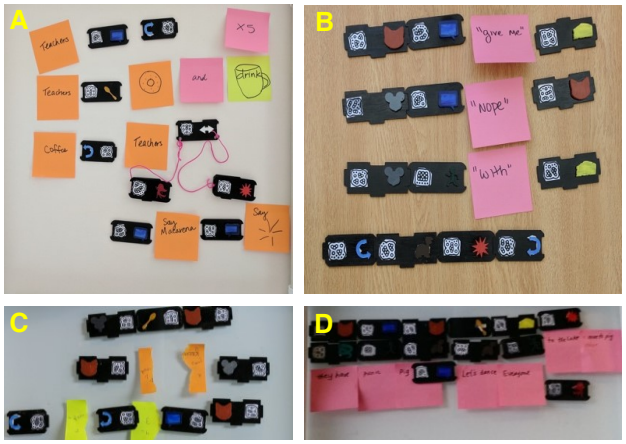

Figure 4. Participants introduced new components to their sample programs: A) new characters and actions; B) strings, represented via text written on sticky notes; C) strings as sticky notes with both printed text and Braille labels; D) Program structured in paragraph form, rather than as a series of stacked lines that represent story actions.

6 DISCUSSION

Our initial evaluation of StoryBlocks demonstrated that the overall approach of combining tangible programming and audio stories can be a compelling way to teach introductory programming concepts in an accessible and collaborative manner. At the same time, we found that participants were eager to do more with StoryBlocks. Here we discuss some of the design decisions that guided

the development of StoryBlocks, the limitations of the current implementation, and the role of StoryBlocks in supporting computational thinking practices.

6.1 Design Tensions and Trade-Offs

Developing StoryBlocks involved navigating numerous design trade-offs. When designing StoryBlocks, we often considered questions such as whether to create a programming language that is large and expressive or small and conceptually simple, or to create tactile blocks that are complex and expensive (with moving parts and embedded electronics) or simple and easy to fabricate. We do not claim that the choices we made are better than other choices; making different choices might result in solutions that are quite different but equally promising. However, we do feel that documenting the choices that we made, and examining these trade-offs, may increase our understanding of how to create accessible, engaging, and fun programming tools.

Many of our design decisions involved trade-offs between simplicity and expressivity. This type of design tension occurs in many programming tools, especially those programming tools that target novice coders. However, these tensions may be magnified when considering tangible programming tools, as adding additional components results in more physical objects and, potentially, physical clutter. For each additional feature, tangible blocks representing that feature must be carried around in sufficient quantity so that they can be used and must be physically searched through whenever the programmer is seeking a specific block. To date, we have focused on creating a "core" set of blocks that can be used to explore tangible programming concepts. In the future, we intend to explore ways to quickly add or repurpose blocks without creating too many additional objects.

Our goal of providing equal access to both blind and sighted users also introduced some unique design challenges. For example, when designing a visual (and non-accessible) programming language, a designer can communicate code types and features using a wide array of visual characteristics such as size, shape, aspect ratio, and visual texture, and can rely upon text labels to emphasize how things work. In a non-visual tangible interface, the user may be unable to perceive more subtle differences. Also, while it is easy to simultaneously compare multiple on-screen objects visually, comparing tactile landmarks is limited by the user's number of free hands. Furthermore, tangible blocks are limited in their ability to show labels, as the blocks may not be large enough to contain Braille, or the user may be unable to read Braille. A major concern here is that if the visual components of the system are more efficient than the tactile components, blind and sighted partners may have difficulty working together, as we saw in some of our study sessions.

6.2 StoryBlocks and Computational Thinking

StoryBlocks aims to support novices in learning about computer science concepts. While the design of StoryBlocks departs from that of traditional programming languages in many ways, we identified a number of places in which StoryBlocks supports aspects of computational thinking [43]. Applying Brennan and Resnick's [8] framework of computational thinking concepts, practices, and perspectives, we find that StoryBlocks supports the concept of performing activities through *sequences* and, to a limited extent, *loops* and *conditionals*, but provides little or no support for *events*, *parallelism*, *operators*, or *data*. StoryBlocks supports the practice of *being incremental and iterative* in developing and playing back stories, and provides opportunities for *testing and debugging*, but currently does not support practices of *reusing and remixing* or *abstracting and modularizing*. Finally, StoryBlocks supports the perspective of *expressing* through coding stories, and *connecting* with others by co-creating and sharing stories, and may potentially support *questioning* by encouraging students to learn about and understand the technologies around them.

7 FUTURE WORK

We have identified a number of future directions for this work, including extending the existing StoryBlocks system, further supporting creative work and problem solving, and applying this approach to new domains.

7.1 Improving StoryBlocks

Our user study revealed a number of possible improvements to the StoryBlocks prototype. We can improve the accessibility of StoryBlocks's code blocks by enriching the blocks' tactile markers, such as by adding textures or Braille labels. We can extend the multimodal aspects of StoryBlocks by adding other forms of feedback such as moving mechanical parts [29], different materials, or even multi-sensory components such as blocks with smell and taste [11].

We may also be able to extend the expressiveness of program output in several ways, such as adding additional

text-to-speech voices and sound effects, or by enabling end users to record their own voices and sound effects.

Developing tutorials, sample activities, and other curricular materials would make it easier for parents, teachers, and children to adopt StoryBlocks. Future versions of StoryBlocks could incorporate a "tutorial mode" in which users recreate simple stories as they did in our study.

During our user evaluation, multiple participants requested that we add mathematical expressions and text strings to StoryBlocks. One approach to managing the additional complexity of these features would be to consider modular sets of program blocks, including a starter set with basic characters and actions, along with add-on sets of blocks representing mathematics, text manipulation, variables, and other advanced features.

7.2 Supporting Creativity and Problem Solving

Beyond improving the usability and increasing the functionality of StoryBlocks, we are interested in improving StoryBlocks's ability to represent creative work and its ability to promote problem solving and computational thinking. To support this goal, we could extend StoryBlocks's character set by adding new characters or enabling users to customize characters; advanced users could create their own physical blocks and specify how the new content is rendered by the system, such as by choosing voice parameters or sound effects. We could also support sharing of users' stories over the Internet, as Scratch enables for its animations and games.

In addition to supporting user creativity, future versions of StoryBlocks could promote problem solving and computational thinking. One way to incorporate problem solving into StoryBlocks would be to develop "puzzle" stories in which part of a story is missing so that the user is required to write code to fill in the missing pieces.

In the future, we may also explore how learners can transfer their knowledge of StoryBlocks to more traditional programming languages and explore how the design of StoryBlocks itself can help or hinder such transitions.

7.3 Enriching Gameplay

Our vision of StoryBlocks is to engage novice programmers in playful activities such as writing and sharing audio stories. We consider this aspect of StoryBlocks to be a storytelling game similar to charades or Pictionary. However, the current prototype focuses mainly on creating stories, and provides very little explicit support for gameplay. Future versions of StoryBlocks could incorporate more game-like activities, such as allowing a player to guess the outcome of a story as it is being created, letting multiple players compete to create the "best" story, or allowing players to create interconnecting, round-robin-style stories.

7.4 New Domains for StoryBlocks

While we have designed StoryBlocks to support computer science education, our underlying approach might also be useful in supporting learning in other domains. We are especially excited about the potential for StoryBlocks to promote learning Braille. Braille literacy rates have declined in recent years, and some researchers believe that this decline may cause problems for blind children when they become adults [19]. Incorporating Braille strings into StoryBlocks would both increase the expressivity of the system and offer a new way to learn Braille.

In addition to supporting Braille education, future versions of StoryBlocks could be used to promote learning in other domains through the use of add-on block sets. For example, a physics add-on could include a set of blocks representing objects in a physical simulation, such as weights on a seesaw, and actions that allow the user to manipulate the simulation. A physics student could then use these blocks to run simulated experiments, listening to the outcome of those experiments as an audio story. StoryBlocks's fundamental model of tangible nouns, verbs, and commands, plus audio feedback, could be applied to many educational domains, including music, mathematics, or chemistry.

8 CONCLUSION

As society increasingly recognizes the importance of computer science education, we must continue to identify ways in which computer science education can exclude learners with disabilities. By identifying the core benefits of block-based languages and representing them in an accessible format, StoryBlocks offers an accessible and engaging learning environment for novice learners. Our formative evaluation with students, teachers, and educational staff shows that StoryBlocks's tangible code blocks and programmable audio stories provide a foundation for mixed-ability groups to discuss and explore programming concepts.

ACKNOWLEDGMENTS

We thank our participants for taking part in our study. We also thank Mike Horn, Richard Ladner, Clayton Lewis,

Lauren Milne, and Kyle Reinholt for their feedback. This work was supported by the National Science Foundation under grants IIS-1619384 and IIS-1652907. Any opinions, findings, conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect those of the National Science Foundation.

REFERENCES

[1] Khaled Albusays and Stephanie Ludi. 2016. Eliciting Programming Challenges Faced by Developers with Visual Impairments: Exploratory Study. In Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '16), 82–85. DOI: https://doi.org/10.1145/2897586.2897616

[2] Catherine M. Baker, Lauren R. Milne, and Richard E. Ladner. 2015. StructJumper: A Tool to Help Blind Programmers Navigate and Understand the Structure of Code. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15), 3043–3052. DOI: https://doi.org/10.1145/2702123.2702589

[3] Ayah Bdeir. 2009. Electronics as material: littleBits. In Proceedings of the 3rd International Conference on Tangible and Embedded Interaction (TEI '09). ACM, New York, NY, USA, 397-400. DOI: https://doi.org/10.1145/1517664.1517743

[4] Theresa Beaubouef and John Mason. 2005. Why the high attrition rate for computer science students: some thoughts and observations. SIGCSE Bull. 37, 2 (June 2005), 103-106. DOI: http://dx.doi.org/10.1145/1083431.1083474

[5] Jeffrey P. Bigham, Maxwell B. Aller, Jeremy T. Brudvik, Jessica O. Leung, Lindsay A. Yazzolino, and Richard E. Ladner. 2008. Inspiring Blind High School Students to Pursue Computer Science with Instant Messaging Chatbots. In Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '08), 449–453. DOI: https://doi.org/10.1145/1352135.1352287

[6] Paulo Blikstein, Arnan Sipitakiat, Jayme Goldstein, João Wilbert, Maggie Johnson, Steve Vranakis, Zebedee Pedersen, and W. Carey. 2016. Project Bloks: designing a development platform for tangible programming for children. http://projectbloks.withgoogle.com/static/Project_Bloks_position_paper_June_2016.pdf

[7] Stacy M. Branham and Shaun K. Kane. 2015. Collaborative Accessibility: How Blind and Sighted Companions Co-Create Accessible Home Spaces. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15). ACM, New York, NY, USA, 2373-2382. DOI: https://doi.org/10.1145/2702123.2702511

[8] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In (AERA 2012), 1–25. Retrieved September 18, 2017 from http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

[9] Robin N. Brewer. 2018. Facilitating discussion and shared meaning: Rethinking co-design sessions with people with vision impairments. In Proceedings of the 12th EAI International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth '18). ACM, New York, NY, USA, 258-262. DOI: https://doi.org/10.1145/3240925.3240981

[10] Craig Brown and Amy Hurst. 2012. VizTouch: automatically generated tactile visualizations of coordinate spaces. In Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction (TEI '12), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, 131-138. DOI: https://doi.org/10.1145/2148131.2148160

[11] MapSense: Multi-Sensory Interactive Maps for Children Living with Visual Impairments. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16). ACM, New York, NY, USA, 445-457. DOI: https://doi.org/10.1145/2858036.2858375

[12] Erin Buehler, Stacy Branham, Abdullah Ali, Jeremy J. Chang, Megan Kelly Hofmann, Amy Hurst, and Shaun K. Kane. 2015. Sharing is Caring: Assistive Technology Designs on Thingiverse. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15). ACM, New York, NY, USA, 525-534. DOI: https://doi.org/10.1145/2702123.2702525

[13] Christian, C. A., Nota, A., Grice, N. A., Sabbi, E., Shaheen, N., Greenfield, P., ... & de Mink, S. E. (2014, January). You Can Touch This! Bringing HST images to life as 3-D models. In *American Astronomical Society Meeting Abstracts# 223* (Vol. 223).

[14] Elise Deitrick, Joseph Sanford, and R. Benjamin Shapiro. 2014. BlockyTalky: A low-cost, extensible, open source, programmable, networked toolkit for tangible creation. *Proceedings of Conference on Interaction Design for Children, Aarhus, Denmark*.

[15] Michael S. Horn and Robert J. K. Jacob. 2007. Tangible programming in the classroom with Tern. In CHI '07 Extended Abstracts on Human Factors in Computing Systems (CHI EA '07). ACM, New York, NY, USA, 1965-1970. DOI: https://doi.org/10.1145/1240866.1240933

[16] Michael S. Horn, Erin Treacy Solovey, R. Jordan Crouser, and Robert J.K. Jacob. 2009. Comparing the use of tangible and graphical programming languages for informal science education. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '09). ACM, New York, NY, USA, 975-984. DOI: https://doi.org/10.1145/1518701.1518851

[17] Amy Hurst and Jasmine Tobias. 2011. Empowering individuals with do-it-yourself assistive technology. In *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility* (ASSETS '11). ACM, New York, NY, USA, 11-18. DOI: http://dx.doi.org/10.1145/2049536.2049541

[18] Felix Hu, Ariel Zekelman, Michael Horn, and Frances Judd. 2015. Strawbies: explorations in tangible programming. In *Proceedings of the 14th International Conference on Interaction Design and Children* (IDC '15). ACM, New York, NY, USA, 410-413. DOI: http://dx.doi.org/10.1145/2771839.2771866

[19] L Johnson. (1996). The Braille Literacy Crisis for Children. *Journal of Visual Impairment & Blindness*, *90*(3), 276-78.

[20] Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. 2007. The reacTable: exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st international conference on Tangible and embedded interaction* (TEI '07). ACM, New York, NY, USA, 139-146. DOI: http://dx.doi.org/10.1145/1226969.1226998

[21] Shaun K. Kane and Jeffrey P. Bigham. 2014. Tracking @Stemxcomet: Teaching Programming to Blind Students via 3D Printing, Crisis Management, and All Twitter. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (SIGCSE '14), 247–252.DOI: https://doi.org/10.1145/2538862.2538975

[22] Shaun K. Kane, Varsha Koushik, and Annika Muehlbradt. 2018. Bonk: accessible programming for accessible audio games. In Proceedings of the 17th ACM Conference on Interaction Design and Children (IDC '18). ACM, New York, NY, USA, 132-142. DOI: https://doi.org/10.1145/3202185.3202754

[23] Filiz Kalelioğlu. 2015. A new way of teaching programming skills to K-12 students: Code. org. *Computers in Human Behavior*, *52*, 200-210.

[24] Caitlin Kelleher. 2015. Looking Glass. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15). ACM, New York, NY, USA, 271-271. DOI: https://doi.org/10.1145/2676723.2691873

[25] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.* 37, 2 (June 2005), 83-137. DOI: http://dx.doi.org/10.1145/1089733.1089734

[26] Varsha Koushik and Clayton Lewis. 2016. An Accessible Blocks Language: Work in Progress. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility* (ASSETS '16), 317–318. DOI: https://doi.org/10.1145/2982142.2982150

[27] Richard E. Ladner. 2015. Design for User Empowerment. *interactions* 22, 2: 24–29. DOI: https://doi.org/10.1145/2723869

[28] Richard E. Ladner and Andreas Stefik. 2017. AccessCSforall: Making Computer Science Accessible to K-12 Students in the United States. *SIGACCESS Access. Comput.*, 118: 3–8. DOI: https://doi.org/10.1145/3124144.3124145

[29] Stephanie Ludi and Tom Reichlmayr. 2011. The use of robotics to promote computing to pre-college students with visual impairments. *ACM Transactions on Computing Education (TOCE)* 11, 3: 20.

[30] Stephanie Ludi. 2015. Position paper: Towards making block-based programming accessible to blind users. *IEEE Blocks and Beyond Workshop*, 67-69.

[31] Roberto Manduchi, Sri Kurniawan, Editors. 2012.*Assistive technology for blindness and low vision.* CRC Press.

[32] Lauren R. Milne. 2017. Blocks4All: making block programming languages accessible for blind children. *ACM SIGACCESS Accessibility and Computing*, 117: 26–29.

[33] T. V. Raman. 1996. Emacspeak—a speech interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '96), Michael J. Tauber (Ed.). ACM, New York, NY, USA, 66-71. DOI: http://dx.doi.org/10.1145/238386.238405

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

[35] Kimiko Ryokai, Michael Jongseon Lee, and Jonathan Micah Breitbart. 2009. Children's storytelling and programming with robotic characters. In *Proceedings of the seventh ACM conference on Creativity and cognition* (C&C '09). ACM, New York, NY, USA, 19-28. DOI: http://dx.doi.org/10.1145/1640233.1640240

[36] Jaime Sánchez and Fernando Aguayo. 2005. Blind Learners Programming Through Audio. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems* (CHI EA '05), 1769–1772. DOI: https://doi.org/10.1145/1056808.1057018

[37] Eric Schweikardt and Mark D. Gross. 2006. roBlocks: a robotic construction kit for mathematics and science education. In *Proceedings of the 8th international conference on Multimodal interfaces* (ICMI '06). ACM, New York, NY, USA, 72-75. DOI: http://dx.doi.org/10.1145/1180995.1181010

[38] Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. *Trans. Comput. Educ.* 13, 4: 19:1–19:40. DOI: https://doi.org/10.1145/2534973

[39] Anja Thieme, Cecily Morrison, Nicolas Villar, Martin Grayson, and Siân Lindley. 2017. Enabling Collaboration in Learning Computer Programing Inclusive of Children with Vision Impairments. In *Proceedings of the 2017 Conference on Designing Interactive Systems* (DIS '17). ACM, New York, NY, USA, 739-752. DOI: https://doi.org/10.1145/3064663.3064689

[40] Linda L. Werner, Brian Hanks, and Charlie McDowell. 2004. Pair-programming helps female computer science students. *J. Educ. Resour. Comput.* 4, 1, Article 4 (March 2004). DOI: http://dx.doi.org/10.1145/1060071.1060075

[41] Michele A. Williams, Erin Buehler, Amy Hurst, and Shaun K. Kane. 2015. What not to wearable: using participatory workshops to explore wearable device form factors for blind users. In Proceedings of the 12th Web for Conference (W4A '15). ACM, New York, NY, USA, Article 31, 4 pages. DOI: https://doi.org/10.1145/2745555.2746664

[42] Michele A. Williams, Amy Hurst, and Shaun K. Kane. 2013. "Pray before you step out": describing personal and situational blind navigation behaviors. In Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '13). ACM, New York, NY, USA, , Article 28 , 8 pages. DOI: http://dx.doi.org/10.1145/2513383.2513449

[43] Jeannette M. Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (March 2006), 33-35. DOI: https://doi.org/10.1145/1118178.1118215