Accessible Tools and Curricula for K-12 Computer Science Education

White paper from the Accessible Computer Science Education Fall Workshop (Draft for Review)

November 17-19, 2020

Earl Huff, Clemson University

Varsha Koushik, University of Colorado

Richard E. Ladner, University of Washington

Stephanie Ludi, University of North Texas

Lauren Milne, Macalester College

Aboubakar Mountapmbeme, University of North Texas

Margaret Perkoff, University of Colorado

Andreas Stefik, University of Nevada, Las Vegas

Introduction

Ladner

Computer science is rapidly becoming an integral part of the K-12 school curriculum in the United States (US) and other countries. This white paper will focus on the US because we understand K-12 education better than in other countries.

Since 1984, computer science has been part of the curriculum in schools that offered AP Computer Science A (CSA). This introduction to programming was originally taught using Pascal, then moved to C++ in 1999, and then Java in 2003. Since 2007, there has been a concerted effort by the National Science Foundation to bring computer science education to a much broader audience and to all levels in K-12. This includes all students regardless of gender, ethnicity, race, or disability. The effort began with Exploring Computer Science (ECS) which was a gentler introduction to computer science at the high school level. Shortly thereafter, the idea of an AP Computer Science Principles (CSP) course arose that was to be a gentler introduction to computer science than AP CSA with a college level equivalent, introductory computer science for non-majors. In 2016, AP CSP became a reality with 43,780 students taking the exam in spring 2017 and two years later that number rose to 94,361. A major objective of ECS and AP CSP is broadening participation in computing, that is, more equitable participation in computer science education among minoritized groups, including students with disabilities. To a modest degree there has been improvement for women, Black, and Hispanic students on participation in AP CSP [Sax 2020]. Unfortunately, the College Board does not release any data about participation in its AP exam by students with disabilities.

Just this year, eleven states reported on the participation of students with disabilities under the Individuals with Disabilities Education Act (IDEA) in computer science courses (ECS, AP CSA, AP CSP, and others) in the 2019-2020 academic year [CSReport2020]. In the eleven states 12.9% of students with disabilities are served under IDEA while at the same time just 7.6% of students taking at least one computer science course were served under IDEA.   Hopefully, in the future we will learn from all the remaining states and the District of Columbia about the participation of students with disabilities in computer science courses. Also, from two to four percent of K-12 students with disabilities are served under Section 504 of the Rehabilitation Act and not under IDEA. We currently have no information regarding their participation in computer science courses.

To be clear, computing skills have been taught in K-12 schools for a very long time, but this is distinct from teaching computer science in general education classes. Teaching of computer skills often occurs at the high school level in Career and Technical Education (CTE) classes that were not considered to be pre-college courses. Computer science, as a discipline in the same family as biology, chemistry, physics, and mathematics, is new to K-12 education and still very much so in the formative stages of its development. Some other newer disciplines taught at the high school level are earth science and environmental science. There is little doubt that computer science belongs among the fundamental disciplines that should be taught in K-12 because of its pivotal importance of computing in almost all facets of science, business, and everyday life.

Until recently, computer science was not taught in K-8 at all, but that is changing quickly with the creation of curricula like Code.org’s CS Fundamentals for grades K-5 and CS Discoveries for grades 6-8, and other curricula. Nonetheless, programming environments for children have become extremely popular starting as far back as 1966 with the Logo programming environment [Solomon2020].   More recently, block-based programming environments have gained popularity with the Scratch programming language and environment and its many variants [Maloney2010, Bau2020]. Although these block-based programming environments have introduced programming and computational thinking to millions of children around the world, none of the popular block-based programming environments are accessible by children who use screen readers or switches for computer access.

The purpose of this white paper is to succinctly describe the fundamental accessibility problems children and young adults with disabilities have in K-12 computer science education, to describe current progress in attacking those problems, and to outline a plan, in research, development, and deployment, in solving those problems. The problems fall into five categories: accessible input, accessible output, accessible infrastructure, curriculum and inclusion.

> **Input** concerns keyboard, mouse, touch, speech, eye-gaze, switch, and reactive interfaces such as augmentative and alternative communication (AAC) devices and gesture sensors found in VR devices.
> **Output** concerns visual (screen), speech, sonification, vibration or other tactile output.
> **Infrastructure** concerns the design, implementation, and deployment of usable accessibility application programming interfaces (APIs).
> **Curriculum** concerns accessible technological approaches to helping students learn computer science.
> **Inclusion** concerns the participation of people with disabilities in the development of accessible tools and curricula in K-12 computer science education.

Accessible Input

Perkoff, Koushik, Milne, Mountapmbeme

In order to make Computer Science education accessible, it is first necessary to consider the manner in which students interact with subject materials and tools. Students may learn to code through an integrated development environment, a text editor, or block-based programming languages. These tools have evolved to make it easier for users to grasp complex coding concepts, however, they have not evolved to accommodate the diverse set of input needs of students. Students with various disabilities may rely on alternative methods for input other than the typical keyboard, mouse, and touch. Alternative modes of input include speech, switch control, eye-tracking, head-tracking, and alternative and augmentative communication (AAC) devices. Speech might be used by a student with limited control of hands. Switch control, eye-tracking, or head-tracking might be used by a student with limited mobility and speech. AAC devices might be used by students with certain cognitive disabilities. Some students may use a combination of these inputs.

AAC

Students with disabilities under IDEA are eligible to be provided any assistive technology device “that is used to increase, maintain, or improve functional capabilities of a child with a disability” [IDEA 2019]. The 2004 Special Education Elementary Longitudinal Study estimated that 9.1% of students surveyed received such assistive technology services [SEELS 2004]. Assistive technology can be used in the classroom to accommodate motor, physical, and cognitive impairments. It can also be used to enhance the ability of students to interact with their teachers and peers. In a national survey evaluating 15,643 students with communication needs, 19.3% of students did not use vocal speech as their primary form of language. 6.9%

used gesture-based language, while 6.5% used pictorial systems and 4.8% used speech generating devices [Andzik et al. 2018]. The latter two methods can be categorized as alternative and augmentative communication (AAC) devices.

There is an extensive range of AAC technology currently available to accommodate the spectrum of complex communication needs. AAC technology includes dedicated devices as well as supplemental applications that can be integrated into a mobile or desktop environment. In addition to the variations in the physical nature of the device, AAC devices can be distinguished by the combination of signal sources that are used for input. Possible signal sources include: images, mechanical input, touch screens, breathing sensors, and brain-computer interfaces [Elsahar et al. 2019]. Imaging methods include systems that track the individual's attention as it shifts through different points in the screen to interact with content either based on eye-gaze or head pointing [Townend et al. 2016]. For individuals with control of their voluntary motor functions, traditional or enhanced keyboards and switches can be used for input. Specialized keyboards allow visually impaired users to read web pages in real-time through automatically refreshing Braille displays and some touchscreen versions [Alnfiai et al. 2017]. Alternatively, voice-to-text services can be a helpful interface for visual users as well as individuals with motor impairments. Visual scene displays based devices allow students to communicate through symbolic language representations instead of selecting text on the screen [Wilkinson et al. 2012]. This breadth of available communication methods has emerged in order to best serve the needs of individuals with different types of communication needs. However, students that are currently making use of such devices and other forms of assistive technology are still limited by the capacity of other educational tools that may be used in Computer Science classrooms.

Text-based programming Environments

Text-based integrated development environments (IDEs), such as Visual Studio, Eclipse and IntelliJ are the standard interfaces for developers to write production code, and as such, they are extensively used in computer science education. These IDEs typically display text-based code in a programming language (e.g., JavaScript, Python, C++) and have a number of integrated tools such as syntax highlighting, code completion and debuggers that allow a programmer to walk through steps of the code as it executes.

There are a number of existing accessibility problems with these environments. For people with motor impairments, text input can be a challenge. Notably many programmers develop repetitive strain injuries because of the large amount of typing that they do. As an alternative, one can use speech to program, which may require some translation as programming syntax is often quite distinct from spoken language syntax [Nowogrodzki2018], and certain languages and environments are less accessible via voice. There are an increasing number of commercial tools available to support coding by voice [Dragon, VoiceCode, Talon]. Alternative input mechanisms include eye-tracking, head-tracking or switch control, which can be used in conjunction with speech control to operate these environments, although these tools are often slower or less accurate than mouse-based control [Hansen2018]. Such methods also often require custom controls to work at all. For example, using the OptiKey eye tracking environment requires an IDE that supports joystick control commands and allows resizing of elements on the screen to increase target sizes for eye tracking accuracy. For people with visual impairments, there is an extended set of accessibility challenges. In particular, it can be difficult to determine context, code structure and syntax errors using a screen reader [Baker2015] [Mealin2012]. Additionally, screen readers do not work well with the integrated tools such as the debuggers and syntax highlighting [Mealin2012]. There have been a few tools created that increase the accessibility of the spoken output for screen readers (e.g. [Raman1996] [Stefik2011b]) and also plug-ins for various environments that increase the ability to understand the context and structure of code [Baker2015] [Stefik2011b]. However, these tend to be piece-meal solutions to improve the accessibility of a particular environment. There is still work to be done on making all environments robustly accessible for people with visual and motor impairments.

Block-based programming Environments

With the increased use of block-based programming (BBP) in K-12 curricula and other outreach activities, there is a need to enhance accessibility in order to serve all students [Ludi 2015], [Milne2019]. BBP is appealing to many novice learners because its intrinsic design allows students to focus on concepts rather than syntax [Weintrop2015]. In a block-based programming environment [BBPE], blocks are usually pre-designed with special structure (shape) and color that communicate most of the syntax. Constructing a program then simply involves connecting multiple compatible blocks with one another to form a syntactically correct program. This interaction usually happens via drag and drop operations using the mouse. However, the interaction/input mechanism as well as the structure of the blocks make BBPE inaccessible to learners who rely on assistive technologies. Most, if not all mainstream BBPEs such as Scratch [Scratch], MakeCode[MakeCode], AppInventor [AppInv] and Code.org curricular [Code] only support the mouse as the primary input device. Other input modalities such as keyboard, touch, speech, eye-gaze and switch are not supported by these systems. This poses a serious barrier for accessibility of CS Ed for all since these environments are increasingly being used to introduce students to programming and computational thinking activities.

There has been some progress in the past few years to make BBPEs accessible to people with disabilities, notably with the creation of an accessible BBPE from the ground-up called Blocks4All [Milne2018] and with the work by Ludi et al. [Ludi2017] that adds keyboard and screen reader interaction on the Blockly Framework [Blockly]. Blockly is a popular framework for building BBPEs. Most mainstream BBPEs such as MakeCode, AppInventor, Scratch, etc. are built on top of the Blockly framework. Therefore, adding accessibility to the Blockly library will indirectly enable accessibility on these mainstream systems. Google recently released a version of Blockly that accepts keyboard input for interaction. Swift Playgrounds [Swift], a hybrid of text-based and block-based programming works well with screen readers despite some existing challenges [Mountapmbeme2020]. The bulk of these research efforts has mainly been geared towards addressing accessibility barriers for people with visual impairments who primarily interact with computers using screen readers, keyboards and touchscreens. The literature says little about other input modalities such as eye-gaze, speech, and AAC devices, but the same issues that apply to IDEs broadly also apply here (e.g., switch support, target size in eye trackers). More research work needs to be done in order to address accessibility for the wide ranges of input modalities used by diverse students to effectively make block-based programming and CS Ed accessible for all.

Tangible programming Environments

Tangible programming languages are widely popular in K-12 education as an alternative to text-based and block-based programming. Some introductory tangible toolkits focus on creating electronic circuits using sensors like Arduino, Littlebits (Bdier, 2009), ProjectBloks (Blikstein, et.al. 2016), roBlocks (Schweikardt & Gross, 2006), and Algobrix. These toolkits utilize hardware and software components that may be inaccessible to students with visual impairments. The Blind Arduino Project is an effort to make circuit-based programming platforms accessible to students who are blind or visually impaired by providing tutorials and hands-on workshops to develop using Arduino.

Tools like Osmo, Awbies (Hu et.al, 2015), and Tern (Horn & Jacob, 2007) explore tangible games to learn programming concepts. Students assemble physical blocks to interact with a screen-based game. By creating non-electronic blocks, these tools particularly focus on affordability for K-12 classrooms. However, interfacing tangible inputs with visual applications remains inaccessible to students with visual impairments.

To make tangible programming accessible to students with disabilities, research has coupled tangible inputs with audio-based outputs. Code jumper (Thieme et.al, 2017) is an accessible tangible toolkit to learn programming concepts by connecting custom pods to create digital music. Exploring tangible blocks as an alternative to visual block-based languages, StoryBlocks (Koushik et.al, 2019) uses physical blocks with distinct tactile markers to create audio stories. However, these tools largely focus on students with visual impairments. Making tangible programming tools accessible to students with diverse abilities can help them engage with computing concepts and foster collaborative learning in K-12 education.

Next Steps

As the demand for computational skills increases nationally, Computer Science education is becoming more prevalent in K-12 classrooms, but it is not inherently accessible to all students. In order to prevent students with disabilities from being left behind in the technical realm, we need to take into account the different input modalities used by this diverse range of students. Future development in this space can make the field more accessible through a number of options. This includes modifying existing programming interfaces as well as applications meant to simplify Computer Science education. These modifications can either be in terms of the physical way that a student is interacting with a tool (through voice control, eye-gaze, or another AAC device) as well as the representation of the code itself. For example, it is currently unknown if the current structure block-based code is as beneficial for students interacting with the program through voice. By navigating the tools with a different input method, it is possible that students’ perception of the information will change as well. Tangible programming tools have significant potential for students with visual disabilities, but for those that may have multiple disabilities including motor impairments they may be difficult to interact with. Extensions of this type of educational tool could consider how to design pieces that are easier to navigate with complex motor needs. One theme that is present across all these tools is a general lack of information. Additional data on the current usage of assistive technologies in classrooms would be extremely beneficial to help guide the direction that developers should take to enhance their applications.

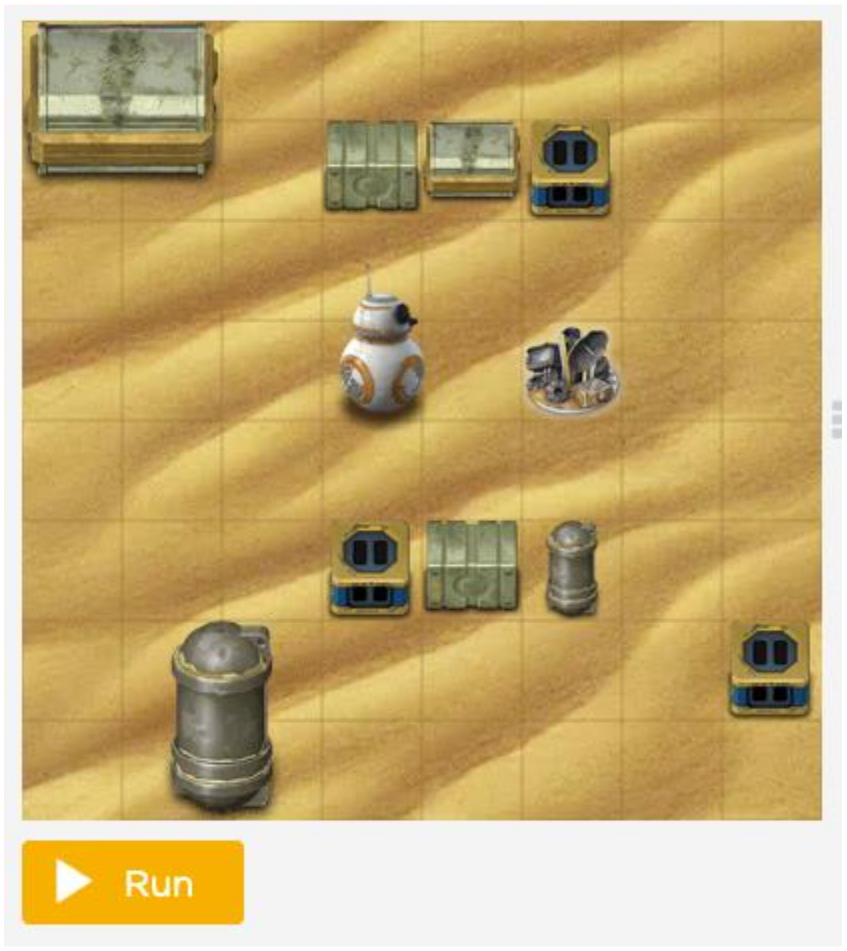
Accessible Output

Stefik, Ladner

Typical outputs from computers are typically visual or auditory, or a combination of both. These outputs can be problematic for students who are blind or visually impaired, or are deaf or hard of hearing. The key problem is to replace an output that cannot be perceived, to an output that can be perceived so that the information obtained is as equivalent as possible. Replacing visual content is most challenging as we shall see below. An easier part of visual replacement or text on a screen with speech output, which is the purpose of screen reader technology. Much more difficult is the replacement of non-text visual output which is a virtually infinite space. A standard way to replace non-text visual output is with audio description. Another issue is navigating through visual content to identify what would be spoken. Replacing audio content that is speech can be done with visual captions. Non-speech output is again a virtual infinite space which may or may not even need replacement depending on its importance to the output. One example might be replacing a “beep” sound with light flashing.

Visual Output to Speech

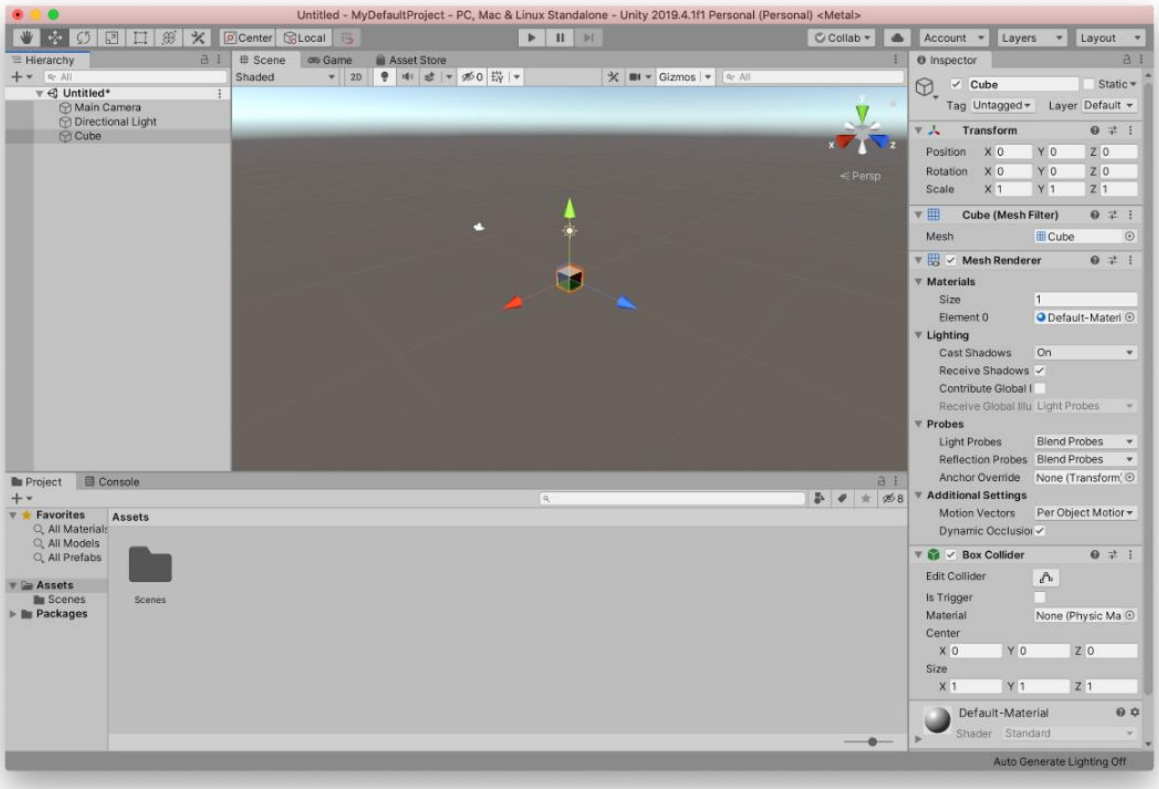
The output of programming environments is equally important in computer science education. Consider, for example, the output of Code.org's Star Wars hour of code tutorial, used by predominantly younger students learning computer science. These tutorials have rich media, are inventive, and are designed to be easy for children to use.



Many tutorials exist like this one and, for young students, they have often defining characteristics. First, typically there exists a graphical overlay on a 2D graphical grid that students visually explore. Second, these tutorials typically involve movement of a character, in the above example bb8, across the computer screen. The input for this movement, either using text or blocks, is literal, like moving the character up, down, left, or right. Success for a particular part of a project is determined if the movements match a pre-set answer (e.g., go right, then down, then rotate left).

While the above is designed for young students, computer programming in the 21st century also has complex output. For example, the environment Unity3D allows for the creation of complex 2D or 3D worlds. Such environments are considerably more advanced than those for children, but are important to consider in K-12 education. Students with disabilities wanting to learn should not have to "stop" learning after high school because the professional environments are not accessible. In the screen capture below, the defining features include 3D scene generation, properties windows that provide details on the scene, assets windows, and a 3D grid.

Environments for children, while rarer, also sometimes provide a 3D grid. While these items are visual, and are not currently accessible, all of them could be, including through screen readers.



Inspector
Cube — Static
Tag: Untagged — Layer: Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

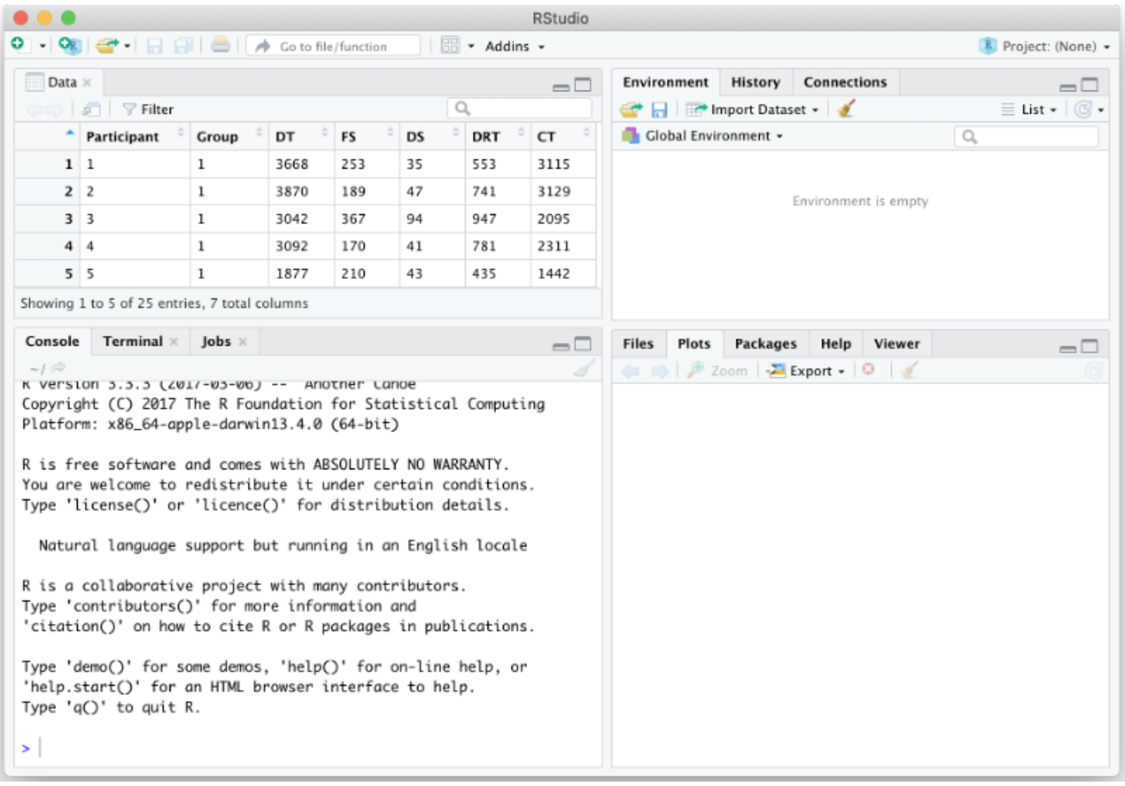
Cube (Mesh Filter)
Mesh: Cube

Mesh Renderer
Materials
Size: 1
Element 0: Default-Materi
Lighting
Cast Shadows: On
Receive Shadows: ✓
Contribute Global I
Receive Global Illu: Light Probes
Probes
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Additional Settings
Motion Vectors: Per Object Motior
Dynamic Occlusioi: ✓

Box Collider
Edit Collider
Is Trigger
Material: None (Physic Ma
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1

Default-Material
Shader: Standard

Accessible output is not limited to Turtle graphics or computer gaming. Similarly, the area of data science increasingly generates complex visualizations. A good example is in the environment RStudio, a professional development environment for data scientists. These environments contain data tables, console-like output, and often generate charts inside them as flat images. These images today are not accessible, but this does not have to be so. Further, while RStudio uses text, it is not guaranteed that because an environment contains text that its input or output is accessible. Both code.org's Hour of Code, using JavaScript, and RStudio's text environments, have significant problems with accessibility, despite being text-based. Point being, the fact that such environments have visual output is not the problem. The key issue is that such environments must have operating system hooks in order to make them connect to accessibility technologies.



	Participant	Group	DT	FS	DS	DRT	CT
1	1	1	3668	253	35	553	3115
2	2	1	3870	189	47	741	3129
3	3	1	3042	367	94	947	2095
4	4	1	3092	170	41	781	2311
5	5	1	1877	210	43	435	1442

Showing 1 to 5 of 25 entries, 7 total columns

```
R version 3.3.3 (2017-03-06) -- "Another Canoe"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

The crucial point to understand is that while many of the backend systems are similar between accessible input and output, input is defining how the user is programming in an environment and output is defining what comes out from that programming effort. Such environments matter in K-12 education and are often graphical, but ultimately need to be accessible past high school as well. Crucially, any initiative on accessibility in high-school thus has to remember that accessibility does not stop at this age range. Students with disabilities need environments that cross the learning and professional spectrum in order to be successful.

Audio Output to Visual Output

As mentioned above, speech output can be converted to captions. While speech and visual output can be consumed and understood simultaneously, a person who is deaf only has one, vision, of the two output modalities. The replacement of speech with caption means that the deaf student has to switch visual attention often between the main visual content and the captions. There is some research on supporting this divided attention problem in the academic classroom setting [Cavender et al. 2009, Kushalnagar et al. 2010] and the on-screen setting [Ouzts et al. 2013]. In terms of computer science education, audio output of most concern is found in educational videos, which can be easily captioned.

Next Steps

Computer science is a rapidly changing field, even at the K-12 level. Just this past year a new data science unit was added to the AP CSP curriculum. As mentioned earlier this curriculum is highly visual using graphs, charts, animations, and interactive graphics. A good next step is to identify and prioritize the visual components of K-12 CS education that do not now have good alternative non-visual access.

Accessible Infrastructure

Stefik

This section addresses the core technologies that are needed to make applications accessible, including tools that are commonly used in K-12 computer science education.   Online curricula such as the Code.org AP CSP curricula use a multitude of tools such as network simulators, data compression simulators, and encryption simulators that are interactive and highly visual. Making accessible versions of these can be very challenging.

In order to make the input and output accessible, operating systems have long contained infrastructures so that application programmers do not have to write code for accessibility components. This is crucial, as writing for accessibility can be very difficult. For example, applications written for the blind not only must connect to screen readers, they must also use Braille. Requiring every application developer to learn Braille would be an unreasonable non-starter. For this reason and others, operating system architectures almost always include an accessibility infrastructure and these must work correctly, otherwise the entire stack of cards collapses for accessibility.

There are several major infrastructures for accessibility. On Windows, there is Microsoft UIA Automation. On Mac and iOS, there is NSAccessibility. On Android, there is the android.view.accessibility package. These systems, even for experts like some of the authors of this white paper, have a ***truly extraordinary learning curve***. As an example, while we know of no citations on the topic, from personal experience the authors have observed the following: 1) Some of UIA's documentation is often wrong, 2) Even the main page for NSAccessibility on mac says "No Overview Available" as the primary source of documentation for the API, and 3) some programming environments for accessibility do not work or poorly work, and 4) to make applications accessible across multiple platforms, accessibility must be written in a variety of incompatible programming languages. All of these barriers, and others, make programming accessibility today incredibly and unnecessarily difficult.

For example, consider the JavaFX accessibility model. In JavaFX, we program for accessibility in Java by sending down accessibility events and this is automated by the user interface libraries in the language. Essentially, a programmer using a "menu" does not write the accessibility layer, nor should they, for the reasons mentioned above. In theory, this simplifies development, as one does not need to write custom accessibility wrappers on each operating system, in multiple programming languages, for multiple user interface controls. However, on Windows, JavaFX sends the wrong properties to Microsoft UIA, which basically means accessibility technologies will not work correctly. On screen readers, for example, they will not "read" at the correct times. To the end user, even knowing what programming language a tool is written in is a large barrier to entry, so they would just be aware that a technology does not work for them. On Mac and other operating systems, the situation is similar and JavaFX is hardly alone.

Further, accessibility infrastructures from manufacturers essentially use a push and query style model. Each operating system calls it something different, like in UIA it is the "provider" model, but the idea is similar. Applications push events to the operating system and AT devices can optionally choose to query for these events and react to them. For example, if a user creates a "text box" but does not create an appropriate provider on Windows, then any and all devices, for any disability, will be broken out of the box for people with disabilities. However, implementing such providers is a herculean task. These providers are old, complex, and require a deep understanding of how specific kinds of AT devices interact with them. For example, even between screen readers like Windows Narrator, JAWS, and NVDA, there are considerable differences in how the device may interact with a text box provider and this matters to users. As a practical example, consider that implementing the provider for progress bars in Microsoft's UIA works differently between NVDA and Microsoft Narrator and does not work in JAWS at all. This is true even if an application developer knows providers exist, knows how to test with screen readers, and implements their provider correctly.

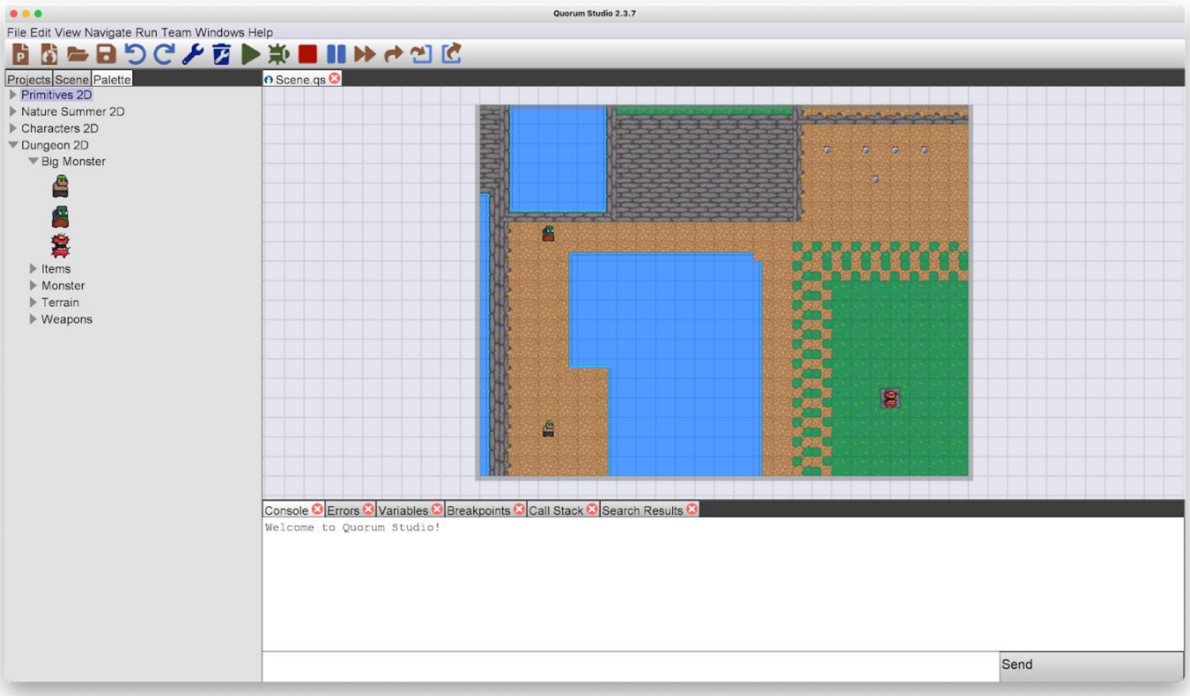
This observation alone is explanatory as a theoretical reason why users of AT technology have such an inconsistent user experience. If an application is programmed in Java, which sends down the wrong events, the team that programmed it may have no idea it is not accessible, nor do they have an easy mechanism to fix it if they do not control the programming language. If the application uses raw UIA, they may not have the expertise to understand why "JAWS users cannot access the progress bar." Or, for those using Unity3D or RStudio, the application teams may not even know what a provider is, let alone have the expertise to program them correctly on all platforms and in multiple programming languages.

Thus, one key issue that needs resolution is improvements to the underlying accessibility infrastructure in operating systems and inside of programming languages. Operating systems and programming languages ultimately are the lynchpin for application developers wanting to make their applications accessible. We think several changes are needed:

> NSAccessibility, UIA, and other similar technologies are extremely difficult to use as APIs and they do not have to be. Many have complex memory management, and inter-process communication requirements (e.g., BSTR values, complex unions) that require additional expertise beyond accessibility and limiting work to experts. These APIs can be both industry scale and dramatically easier to use.

A cross-platform, and correct, API for accessibility infrastructure is sorely needed, across all industry programming languages and products. Application developers should not have to worry that the programming language they are using may be inaccessible on a technical level and they should not have to write custom accessibility implementations everywhere. Given that many programming languages have poor, or barely working, accessibility implementations, it is not surprising that applications themselves lack accessibility support.
Accessibility APIs should be built into the core graphics toolkits, like OpenGL, Direct3D, Metal, and otherwise. Graphics developers today have a herculean task to make their applications accessible, but this need not be so with improvements to accessibility infrastructures at the operating system and programming language level.

The only existing infrastructure that approaches these goals is an academic project embedded into the [Quorum programming language]() and used in the Quorum Studio environment. For example, consider the image below. This image is of Quorum Studio presenting 2D grid-like information, similar to scene editors in Unity3D. For this to work accessibly, Quorum implements a customized accessibility manager, which connects to appropriate providers (currently on Windows only).



```
Welcome to Quorum Studio!
```

As there is no provider, on any platform, for computer graphics, creating 2D or 3D scenes such as this requires creative use of focus events and customized providers inside of Quorum. Using this or technologies like it allow any graphical technology to be accessible, although such an infrastructure is sorely needed at industry scale and plausibly built into the core graphical toolkits.

Curriculum

Huff, Ludi

As Computer Science (CS) continues to make its way into K-12 schools across the U.S., the accessibility of the curricula used remains in question. It’s not enough to introduce CS to students early in their learning; there needs to be a greater effort in ensuring the curriculum used in the classroom is inclusive and accessible to all students, including those with disabilities. There is a sense of urgency to provide access to CS education in K-12 to as many schools as possible; however, those efforts primarily focused on increased participation of students from disadvantaged and underrepresented populations such as Black, LatinX, and Native Americans. However, with this approach, the design of resulting CS curricula does not consider the needs and preferences of students with disabilities, putting them at a disadvantage in their learning progression. While not impossible, many students with disabilities persevered and achieved success in computing; they face a steeper hill to climb than their sighted peers.

The struggles encountered during their education due to the curriculum may have a lasting impact on integrating and contributing as developers at the professional level. The exploratory study of Mealin and Murphy-Hill [Mealin2012] suggest that the challenges faced by the blind or visually impaired programmers are due to their inexperience in using the developer tools because of their lack of education. At the K-12 level, the curriculum may often decide what tools or environments teachers use; however, some curricula offer flexibility in using different environments or languages. For example, CodeHS provides various pathways to 6th-12th and K-12 curriculum and provides courses using JavaScript, Java, and Python [CodeHS]. However, when accessibility for students with disabilities becomes a factor, the tool and language of choice may depend on how it works with their assistive technologies. In a 2019 study, Baker et al. interviewed how teachers of the visually impaired (TVI) determined what factors influenced their choice of technologies to use in their classrooms [Baker2019]. The most significant factors included a) if the technology matches what their peers use, b) the ability of the student(s), and c) what is best suited for the environment and task at hand. Similarly, in a 2020 study, Huff et al. interviewed TVIs regarding their teaching experience with blind or visually impaired students in CS courses [Huff2020b]. Among the findings, the study revealed teachers were often modifying the curriculum. The kind of tools and materials used differed for some teachers based on if the student was blind or had low vision. TVIs expressed the need to redesign current curricula to provide more accessible formats of their materials and improve their learning tools to work with screen readers. The current challenges in existing curricula affect students in K-12 and the teachers and how they prepare to teach CS. The following section details recent attempts to improve CS curricula and introduce alternative educational approaches for introducing CS to students with disabilities.

Current Efforts in Improving CS Curricula

In response to the growing need to improve accessibility in CS education, several initiatives, organizations, and solutions, both technological and educational, are designed to mitigate or eliminate barriers to the user experience for students with disabilities in learning programming. The following subsections introduce examples of current efforts in the form of educational workshops, curricula, developer tools, and environments for which enhances access to learning CS for people with disabilities.

Educational Workshops & Curricula

A side effect of current K-12 CS curricula designed without full consideration of people with disabilities is how learning materials and editors do not work well with assistive technologies such as screen readers. This side effect can limit how people with visual impairments participate using screen readers to access online content. To address this barrier, AccessCSforAll researched approaches toward making K-12 CS curricula more accessible, even for screen reader users. The team of Stefik et al. created a version of Code.org's Computer Science Principles (CSP) course that is screen-reader accessible [Stefik2019]. The modifications included unplugged alternatives to CSP's activities and making versions of specific activities doable using the Quorum programming language [Quorum]. Bootstrap provides a series of curricular modules and a tool for teaching computing, math, physics, and data science to 6-12 grade students. Currently, work is underway to make their tool accessible for people with disabilities [Schanzer2019, Schanzer2020]. Ludi et al.'s work saw enhancements to the Robotics unit of the Exploring Computer Science curriculum to improve accessibility in robotics programming for visually impaired students using JBrick, a programming environment for Lego Mindstorms NXT development [Ludi2018, Ludi2014].

In addition to redesigning K-12 CS curricula for accessibility, other approaches include workshops using more accessible devices, environments, and tools to expose students with disabilities to computer science. In a 2008 study, Bigham et al. conducted a workshop for blind high school students developing an instant messaging chatbot using software that was deemed accessible for blind persons [Bigham2008]. In 2014, Kane and Bigham conducted a workshop over four days with blind high school students, analyzing Twitter data, and creating interactive visualizations [Kane2014]. They used Ruby to create the Twitter analysis scripts and then create 3D tactile graphics, printable using a 3D printer. Stefik et al. conducted a summer workshop with several blind or visually impaired students using Sodbeans, a development environment based on the Netbeans integrated development environment (IDE), and the Hop programming language. The authors evaluated the IDE and programming language using a multisensory CS curriculum designed to be accessible for students with visual impairments [Stefik2011].

Educational Tools & Environments

To complement efforts in curricula, developing development tools and environments that are inclusive to learners with disabilities is critical. The Quorum language, with the associated development environment, is an example of development tools that support computer science education for students who are blind and visually impaired [Stefik2013]. Quorum supports programming that includes games and robotics, as well as general programming uses. There are other tools developed that support specific domains. For example, The JBrick development tool enables persons with or without sight to program LEGO Mindstorms robots via a subset of the C language [Ludi2010]. Both JBrick and Quorom's development tools focused on providing access to programming and compatibility with assistive technology such as screen readers and refreshable Braille displays. These tools focus on traditional, text-based programming languages, while other work focuses on block-based tools that target novice programmers. Both Milne and Ludi's work is pursuing the addition of features to support persons with visual impairments in Google's Blockly framework via touchscreen and keyboard use, respectively [Milne2019, Ludi2015]. Key areas of concern include both the basics of input and output within the programming tasks (discussed earlier in this paper) while also investigating specific features such as audio, search, and code understanding, especially in computer science education itself. Other work studied a narrow aspect of programming, such as a concept or the input method itself. For example, the PLUMB EXTRA3 tool teaches the fundamentals of data structures to students who are blind [Calder2007]. While traditionally, the input of programming is via the device (such as the computer or tablet), Morrison et al. created Torino, a tangible programming tool for teaching programming concepts to children ages 7-11 who are visually impaired [Morrison2018]. Torina has become Code Jumper as it has moved out of research and into an actual product (Thieme et.al, 2017).

While companies developed some accessibility features and plug-ins to support users of mainstream development tools such as Microsoft's Visual Studio and Eclipse, these environments are used by people who are either more advanced in their education or are professionals. The tools noted above focus on people, often students, who are learning to program. The goal is to increase access to Computer Science, even for students in elementary school.

Potential Resource: Accessible Tutorials

Tutorials may be one area to explore in improving the educational experience in CS. Current developer tools may be inaccessible for people with disabilities [Albusays2016, Huff2020a] and, therefore, serves as a barrier to their learning. While research continues to improve tool accessibility, developing tutorials or instructional media for using developer tools by a person with a disability may help mitigate the challenges in the onboarding process for students. For example, developers may look for a video tutorial to learn to use Visual Studio Code on YouTube. As another example, developers may want to take a course to learn a web framework, and thus they may take an online class on a website such as Udacity or Coursera. These tutorials may be useful as supplemental materials for learning to write code, use developer tools, and learn CS concepts. However, many tutorials come in the form of videos and may not be designed initially for people with visual impairments. Videos can be difficult for people with visual impairments to follow because of the lack of context and screen readers' inability to interpret the videos correctly [Voykinksa2016]. More importantly, the demonstration of developer tools such as text editors or integrated development environments (IDE) comes from the perspective of someone without a visual impairment. Therefore, such tutorials lack consideration for using such tools with a screen reader or keyboard shortcuts to navigate the user interface (UI) or access specific functions. Another consideration is the availability of existing tutorials made by screen reader users. It is currently unknown how much media of using software developer tools from the perspective of screen reader users exists on the Web. Although previous work examined the kinds of videos published on YouTube by people with visual impairments, the focus was on video blogs (vlogs) [Seo2017].

To that end, video tutorials for people with disabilities, especially visual disabilities, can be viable and valuable for learning to use educational developer tools and thus improve learning outcomes. The question then is, “What would it look like?” One possible outcome could be a repository of tutorial videos, resembling the structure of the popular platform YouTube, consisting of content generated by people with disabilities showing how to use technology. To a more considerable extent, we may consider developing a website hosting massive open online courses (MOOCs); these MOOCs would comprise lectures providing incremental knowledge towards achieving the overarching goal of learning a new technology, language, or task. Platforms such as Udacity, LinkedIn Learning, Coursera, and Udemy are examples of such a model. However, then, the question becomes, “Why not just upload such content to those platforms?” While using existing platforms would reduce the time, effort, and resources required to achieve the outlined goals, there is the question of accessibility on existing platforms. Previous work identified accessibility issues in some of the more popular MOOC platforms [Al-Mouh2014, Bohnsack2014, Sanchez-Gordon2016].

The next question to pose would be, “What will it take to realize accessible tutorials for people with visual impairments?” The reality is that such a task will require significant research, human resources, stakeholder involvement, and cost. The following subsections outline steps toward developing an accessible video tutorial.

Infrastructure/Content Development

Critical considerations must include 1) who will produce the tutorials, 2) what resources are required to produce the tutorials, 3) what format(s) are the most accessible for the intended audience, and 4) what will the infrastructure look like for the system. For the benefit of blind or low vision users, the videos’ development would require audio descriptions to inform viewers of the objects on the screen and the action(s) taking place. However, audio descriptions would require substantial time, effort, and cost to generate for existing video tutorials. However, there are alternative approaches to providing audio descriptions to tutorials. For example, YouDescribe [YouDescribe], a website and mobile application run by the Smith-Kettlewell Eye Research Institute, crowdsources audio descriptions for YouTube videos. Users can sign in using a current Google account to create audio descriptions of existing videos within the repository. Another approach is hiring visually impaired content creators or developers to produce videos. From their perspective, developing the videos to cater to the needs of other blind or visually impaired people will demonstrate and articulate their lessons in a comprehensible manner and, therefore, would not require audio descriptions. Of course, the drawback in such an approach would be the cost to hire their services.

Funding

Funding sources are always a consideration when discussing the design, development, and deployment of a web-based service. Cost considerations include hiring content developers, production specialists, web development and publication, and other key components.

Stakeholders and Expertise

We identify potential stakeholders for a possible video repository portal, beginning with people with disabilities, in this case, blind or visually impaired people. In considering an inclusive design approach, their input would be crucial to ensuring the final product meets their needs. That is not to say the portal is strictly for the blind or visually impaired; however, designing it to meet their needs indirectly addresses people's needs with other forms of impairment (e.g., people with learning disabilities such as dyslexia). A supplement to existing CS curricula for K-12 schools, the portal may benefit students and teachers. The potential barrier for teachers integrating the tutorials in their curriculum may include the regulations from their district on using videos in their lectures. Many school districts impose restrictions on the use of YouTube videos in the classroom because students may use it as an opportunity to browse for non-related content on the platform. A potential workaround would be teachers integrating specific videos into a slide show or online in a learning content management system (LCMS). This approach would serve as a benefit for both in-school and out-of-school learning for students. There is an increase of people forgoing formal CS education and instead pursue their education through coding boot camps or self-learning through videos, making them beneficiaries and stakeholders for the portal.

The expertise required for developing the videos and platform would ideally include (this is not an exhaustive list) the following:

- Content creation/editing/production
- Marking
- Product/tool developers at mainstream companies
- Collaborations with disability organizations (e.g., National Federation for the Blind)
- Research in HCI, Accessibility, and Computing Education
- Leadership/project management can build and maintain connections with organizations, stakeholders, and funders.

Next Steps

The first step in designing and developing a prototype accessible portal for hosting video tutorials for people with visual impairments involves a user needs analysis to identify potential end-users' needs and requirements. The analysis may include a survey of and interview with teachers of visually impaired (TVI) students in CS and the students learning CS. The teacher/student studies' goal will be to identify the most challenging CS topics and development tools for blind or visually impaired students. While prior work identified tools chosen by TVIs for their CS courses [Baker2019] and challenging and easy topics for blind or visually impaired students [Huff2020b], additional exploration with more participants will help produce more generalized recommendations for the broader population. We can analyze the pool of topics to select the most challenging ones (2-3). The next step will be to create a small collection of video tutorials with the aid of recruited blind or visually impaired developers and production specialists. An evaluation study will assess the videos' effectiveness and the effect on students' efficacy in learning challenging topics. A viable approach would be a diary study where students use the videos as part of their learning over time and submit entries on their experience and their attitude and perceived usefulness with follow-up interviews. Findings from the study will provide feedback on the tutorials' strengths and weaknesses to reiterate their development. The aforementioned is a suggested series of steps toward using an inclusive design approach to design and develop video tutorials as an accessible supplement or alternative for learning CS for K12 students with disabilities.

Inclusion

Ladner

Generally, tools and curricula for computer science education at the K-12 level are not developed or tested by people without disabilities. In addition, most college level computer science programs do not teach about how to make applications and websites accessible. For these reasons and others, most tools and curricula are most often not accessible to screen reader users, switch users, and users who require other access technologies. As mentioned earlier, the well-known Scratch programming language and environment that is used by millions of children, both in and out of the classroom, is not accessible. The ECS curriculum that uses Scratch is not fully accessible. All the endorsed AP CSP curricula are not fully accessible. For example, Code.org AP CSP curriculum uses App Lab, a block-based programming environment that is not screen reader or switch access.

The solution to this problem is to increase the number of people with disabilities in the enterprise of creating tools and curricula for K-12 computer science education. This will help embed people who understand disability in companies and organizations that are creating tools and curricula. These people with disabilities should have the power to influence the accessibility of the tools and curricula as they are being developed.   Some of them could even be programmers or curriculum designers.

In the subsequent subsections we will look at the progress so far on increasing the number of people with disabilities in creating tools and curricula for K-12 computer science education, what future research is needed, and the next steps that are needed to move forward on inclusion.

Progress on Inclusion

Until recently data on the participation of K-12 students with disabilities in computing courses did not exist. As mentioned earlier, 11 states reported on students under IDEA that took at least one computer science course [CSRepor2020]. In those 11 states 12.9% of K-12 students were served under IDEA, but only 7.6% of students who took at least one computer science course were served under IDEA. Hopefully, in the coming years all 50 states and the District of Columbia will report data of students with disabilities taking computer science courses. The AccessCSforAll project has been developing accessible tools and curricula for AP CSP and training teachers of deaf, blind, and learning-disabled students to teach the course [ACSA]. The Quorum Language and programming environment has been developed to be screen reader accessible from the outset [Q, Stefik2011] which allows the participation of blind students and others who use screen readers to participate in computer science more widely. Blocks4All has been developed and tested for making block-based programming accessible for iOS VoiceOver users [Milne2018]. Work has also been done to help make the block-based programming language framework called Blockly more accessible to screen reader users [Ludi2017]. There has also been work on making the popular programming environment for the Bootstrap program accessible [Schanzer2019]. There is also considerable work being done to help teachers learn the principles and practices of Universal Design for Learning (UDL) that can make learning computer science easier for students with learning differences [Israel2020]. All this work is allowing more K-12 students with disabilities to participate in computer science.

Research on Inclusion

There is scant human studies research on inclusion for people with disabilities in computing fields. One major impediment is the inconsistency of data about the participation of people with disabilities in the computer science education pipeline [LadnerCRA, Blaser2020]. Without reliable data about people with disabilities, it is difficult to track progress on inclusion. This problem about data on disability is an international problem that cannot be addressed only by the computing field.

One of the major resources for research on diversity, equity, and inclusion in computing fields is the annual IEEE Conference on Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT) that has been held every year since 2015. Another source of information is the Data Buddies that does quantitative research about broadening participation in computing for the Computing Research Association [DB1]. One striking result from the project is that 32% of undergraduate majors in computer science who have a disability feel like an outsider in the computing field and even more (45-46%) for women or underrepresented minorities with disabilities [DB2]. This compares to 17% for majority men with disabilities who feel like outsiders. With both RESPECT and Data Buddies, the focus has been mostly on inclusion at the college level, not at the K-12 level.

Next steps on Inclusion

A critical next step is some sort of survey article that summarizes the state of inclusion of students with disabilities in computer science at the K-12 level. We are beginning to get data on participation of students with disabilities in computing courses, but it is incomplete. We have no idea how they are doing on the AP CSA and CSP exams. Working with the College Board to get that data would be very helpful. Once we have the data on disability in K-12 CS education it needs to be disaggregated to see which students with disabilities are not participating. Is it blind students, deaf students, or other groups? From a more local perspective, it would make sense to examine some larger school districts to see how they are including students with disabilities in CS. How does New York City compare with Chicago in inclusion?

Conclusion

This white paper examines the accessibility barriers in K-12 CS education, what work has been done so far to mitigate those barriers, and what needs to be done next. Five areas were covered: input, output, infrastructure, curriculum, and inclusion. No new problems have been solved in this paper, but hopefully the paper will inspire researchers and developers to improve the accessibility of K-12 computer science education for the approximately 9 million students with disabilities in the United States, and the millions more around the world.

Acknowledgments

We would like to thank the organizers of the Accessible Computer Science Education Fall Workshop, especially Roy Zimmermann, Jinoos Safavian, and the others on the Microsoft Research Outreach team, for their support in making the workshop happen. We acknowledge the help of other members of our breakout group, Tom Ball and Colin Clark, in the development of this white paper.

References

[AC] AccessComputing. https://www.washington.edu/accesscomputing/

[ACSA] AccessCSforAll. https://www.washington.edu/accesscomputing/accesscsforall

[Albusays2016] Albusays, K., & Ludi, S. (2016, May). 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 (pp. 82-85).

[Al-Mouh2014] Al-Mouh, N. A., Al-Khalifa, A. S., & Al-Khalifa, H. S. (2014, July). A first look into MOOCs accessibility. In International Conference on Computers for Handicapped Persons (pp. 145-152). Springer, Cham.

[Alnfiai et al. 2017] Mrim Alnfiai and Srinivas Sampalli. 2017. BrailleEnter: A Touch Screen Braille Text Entry Method for the Blind. *Procedia Comput. Sci.* 109, (January 2017), 257–264.

[Andzik et al. 2018] Natalie R. Andzik, John M. Schaefer, Robert T. Nichols, and Yun-Ching Chung. 2018. National survey describing and quantifying students with communication needs. *Dev. Neurorehabil.* 21, 1 (January 2018), 40–47.

[AppInv] “MIT App Inventor | Explore MIT App Inventor.” https://appinventor.mit.edu/ (accessed Sep. 10, 2020).

[Baker2019] Baker, C. M., Milne, L. R., & Ladner, R. E. (2019). Understanding the Impact of TVIs on Technology Use and Selection by Children with Visual Impairments. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–13. https://doi.org/10.1145/3290605.3300654

[Baker2015] 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). Association for Computing Machinery, New York, NY, USA, 3043–3052. DOI:https://doi.org/10.1145/2702123.2702589

[Bau2020] David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn Turbak. 2017. Learnable programming: blocks and beyond. Commun. ACM 60, 6 (June 2017), 72–80. DOI:https://doi.org/10.1145/3015455

[Bdeir, 2009] 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

[Bigham2008] Bigham, J. P., Aller, M. B., Brudvik, J. T., Leung, J. O., Yazzolino, L. A., & Ladner, R. E. (2008). Inspiring blind high school students to pursue computer science with instant messaging chatbots. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, 449–453. https://doi.org/10.1145/1352135.1352287

[Blaser2020] Briana Blaser, Richard E. Ladner. 2020. Why is Data on Disability so Hard to Collect and Understand? In proceedings of the 2020 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT). 8 pages. https://www.washington.edu/doit/sites/default/files/atoms/files/RESPECT_2020_DisabilityData.pdf

[Blikstein, et.al. 2016] Paulo Blikstein, et.al. 2016. Project Bloks: designing a development platform for tangible programming for children. Position paper, retrieved online on, 06-30

[Blockly] "Blockly | Google Developers." https://developers.google.com/blockly (accessed Apr. 27, 2020). [Bohnsack2014] Bohnsack, M., & Puhl, S. (2014, July). Accessibility of MOOCs. In International Conference on Computers for Handicapped Persons (pp. 141-144). Springer, Cham.

[Cavender et al. 2009] Anna C. Cavender, Jeffrey P. Bigham, and Richard E. Ladner. 2009. ClassInFocus: enabling improved visual attention strategies for deaf and hard of hearing students. In Proceedings of the 11th international ACM SIGACCESS conference on Computers and accessibility (Assets '09). Association for Computing Machinery, New York, NY, USA, 67–74. DOI:https://doi.org/10.1145/1639642.1639656

[Code] "Learn | Code.org." https://code.org/learn (accessed Sep. 19, 2020).

[CodeHS] *CodeHS - Teach Coding and Computer Science at Your School | CodeHS*. (n.d.). Retrieved August 28, 2020, from https://codehs.com/

[CSReport2020] 2020 State of Computer Science Education: Illuminating Disparities. Published by Code.org Advocacy Coalition, Computer Science Teachers Association, Expanding Computing Education Pathways. https://advocacy.code.org/stateofcs

[DB1] Data Buddies Research Findings. https://cra.org/cerp/research-findings/

[DB2] Burcin Tamer. Feeling like an Outsider in Computing? You are not Alone. https://cra.org/cra-wp/feeling-like-an-outsider-in-computing-you-are-not-alone/

[Calder2007] M. Calder, R. F. Cohen, J. Lanzoni, N. Landry, and J. Skaff, (2007). “Teaching Data Structures to Students who are Blind,” ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science educationJune 2007 Pages 87–90. https://doi.org/10.1145/1268784.1268811

[Elsahar et al. 2019]]Yasmin Elsahar, Sijung Hu, Kaddour Bouazza-Marouf, David Kerr, and Annysa Mansor. 2019. Augmentative and Alternative Communication (AAC) Advances: A Review of Configurations for Individuals with a Speech Disability. *Sensors* 19, 8 (April 2019). DOI:https://doi.org/10.3390/s19081911

[Hansen2018] John Paulin Hansen, Vijay Rajanna, I. Scott MacKenzie, and Per Bækgaard. 2018. A Fitts' law study of click and dwell interaction by gaze, head and mouse with a head-mounted display. In Proceedings of the Workshop on Communication by Gaze Interaction (COGAIN '18). Association for Computing Machinery, New York, NY, USA, Article 7, 1–5. DOI:https://doi.org/10.1145/3206343.3206344

[Horn & Jacob, 2007]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

[Hu et.al, 2015] 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

[Huff2020a] Huff, E. W., Boateng, K., Moster, M., Rodeghero, P., & Brinkley, J. (2020, September). Examining The Work Experience of Programmers with Visual Impairments. In 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 707-711). IEEE.

[Huff2020b] Huff Jr, E. W**.**, Boateng, K., Moster, M., Rodeghero, P., & Brinkley, J. (2020). Exploring the Perspectives of Teachers of the Visually Impaired Regarding Accessible K12 Computing Education. *In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE’21), March 13- March 20, 2021, Virtual Event*. ACM, New York, NY, USA. [To Appear]

[IDEA 2019]]2019. *Individuals with Disabilities Education Act*. Retrieved December 4, 2020 from https://sites.ed.gov/idea/statute-chapter-33/subchapter-i/1401

[Israel2020] Maya Israel, Gakyung Jeong, Meg Ray, and Todd Lash. 2020. Teaching Elementary Computer Science through Universal Design for Learning. In Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE ’20). Association for Computing Machinery, New York, NY, USA, 1220–1226. DOI:https://doi.org/10.1145/3328778.3366823

[Kane2014] Kane, S. K., & Bigham, J. P. (2014). Tracking @stemxcomet: Teaching programming to blind students via 3D printing, crisis management, and twitter. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, 247–252. https://doi.org/10.1145/2538862.2538975

[Koushik et.al, 2019] Varsha Koushik, Darren Guinness, and Shaun K. Kane. 2019. StoryBlocks: A Tangible Programming Game To Create Accessible Audio Stories. In <i>Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems</i> (<i>CHI '19</i>). Association for Computing Machinery, New York, NY, USA, Paper 492, 1–12. DOI:https://doi.org/10.1145/3290605.3300722

[Kushalnagar et al. 2010] Raja S. Kushalnagar, Anna C. Cavender, and Jehan-François Pâris. 2010. Multiple view perspectives: improving inclusiveness and video compression in mainstream classroom recordings. In Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility (ASSETS '10). Association for Computing Machinery, New York, NY, USA, 123–130. DOI:https://doi.org/10.1145/1878803.1878827

[LadnerCRA] Richard E. Ladner. Expanding the Pipeline: The Status of Persons with Disabilities in the Computer Science Pipeline. Computing Research News, November, 2020. https://cra.org/crn/2020/11/expanding-the-pipeline-the-status-of-persons-with-disabilities-in-the-computer-science-pipeline/

[Ludi2010] Stephanie Ludi, Mohammed Abadi, Yuji Fujiki, Priya Sankaran, and Spencer Herzberg. 2010. JBrick: accessible lego mindstorm programming tool for users who are visually impaired. In <i>Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility</i> (<i>ASSETS '10</i>). Association for Computing Machinery, New York, NY, USA, 271–272. DOI:https://doi-org.libproxy.library.unt.edu/10.1145/1878803.1878866

[Ludi2014] Ludi, S., Ellis, L., & Jordan, S. (2014). An accessible robotics programming environment for visually impaired users. *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility*, 237–238. https://doi.org/10.1145/2661334.2661385

[Ludi2015] S. Ludi, “Position paper: Towards making block-based programming accessible for blind users,” in *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, 2015, pp. 67–69.

[Ludi2017] Stephanie Ludi and Mary Spencer. 2017. Design Considerations to Increase Block-based Language Accessibility for Blind Programmers Via Blockly. J. Vis. Lang. Sentient Syst. 3 (2017): 119-124.

[Ludi2018] Ludi, S., Bernstein, D., & Mutch-Jones, K. (2018). Enhanced Robotics! Improving Building and Programming Learning Experiences for Students with Visual Impairments. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 372–377. https://doi.org/10.1145/3159450.3159501

[MakeCode] “Microsoft MakeCode.” https://www.microsoft.com/en-us/makecode (accessed Apr. 27, 2020).

[Maloney2010] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. ACM Trans. Comput. Educ. 10, 4, Article 16 (November 2010), 15 pages. DOI:https://doi.org/10.1145/1868358.1868363

[Mealin2012] Mealin, S., & Murphy-Hill, E. (2012). An exploratory study of blind software developers. *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 71–74. https://doi.org/10.1109/VLHCC.2012.6344485

[Milne2018] Milne, L. R., & Ladner, R. E. (2018). Blocks4All: Overcoming Accessibility Barriers to Blocks Programming for Children with Visual Impairments. CHI Conference on Human Factors in Computing Systems. ACM. https://dl.acm.org/doi/10.1145/3173574.3173643

[Milne2019] L. R. Milne and R. E. Ladner, “Position: Accessible Block-Based Programming: Why and How,” in *Proceedings - 2019 IEEE Blocks and Beyond Workshop, B and B 2019*, Oct. 2019, pp. 19–22, doi: 10.1109/BB48857.2019.8941230.

[Morrison 2018] Morrison, C., Villar, N., Thieme, A., Ashktorab, Z., Taysom, E., Salandin, O., . . . Zhang, H. (2018). Torino: A Tangible Programming Language Inclusive of Children with Visual

Disabilities. Human-Computer Interaction, 1-49.
https://www.tandfonline.com/doi/abs/10.1080/07370024.2018.1512413?journalCode=hhci20

[Mountapmbeme2020] A. Mountapmbeme and S. Ludi, “Investigating Challenges Faced by Learners with Visual Impairments using Block-Based Programming / Hybrid Environments,” in *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS ’20)*, 2020.

[Nowogrodzki2018] Nowogrodzki, Anna. "Speaking in code: how to program by voice." Nature 559.7712 (2018): 141-143.

[Ong2019] J. S. Y. Ong, N. A. O. Amoah, A. E. Garrett-Engele, M. I. Page, K. R. McCarthy, and L. R. Milne, “Expanding Blocks4All with Variables and Functions,” in *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, 2019, pp. 645–647.

[Ouzts et al. 2013] Andrew D. Ouzts, Nicole E. Snell, Prabudh Maini, and Andrew T. Duchowski. 2013. Determining optimal caption placement using eye tracking. In Proceedings of the 31st ACM international conference on Design of communication (SIGDOC '13). Association for Computing Machinery, New York, NY, USA, 189–190.
DOI:https://doi.org/10.1145/2507065.2507100

[Q] Quorum. https://quorumlanguage.com/

[Raman1996] T. V. Raman. 1996. Emacspeak—direct speech access. In Proceedings of the second annual ACM conference on Assistive technologies (Assets '96). Association for Computing Machinery, New York, NY, USA, 32–36. DOI:https://doi.org/10.1145/228347.228354

[Sanchez-Gordon2016] Sanchez-Gordon, S., & Luján-Mora, S. (2016). How could MOOCs become accessible? The case of edX and the future of inclusive online learning. Journal of Universal Computer Science, 22(1).

[Sax2020] Linda J. Sax, Kaitlin N.S. Newhouse, Joanna Goode, Max Skorodinsky, Tomoko M. Nakajima, and Michelle Sendowski. 2020. Does AP CS Principles Broaden Participation in Computing? An Analysis of APCSA and APCSP Participants. In Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20). Association for Computing Machinery, New York, NY, USA, 542–548.
DOI:https://doi.org/10.1145/3328778.3366826

[Schanzer2019] Schanzer, E., Bahram, S., & Krishnamurthi, S. (2019). Accessible AST-Based

Programming for Visually-Impaired Programmers. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 773–779. https://doi.org/10.1145/3287324.3287499

[Schanzer2020] Schanzer, E., Bahram, S., & Krishnamurthi, S. (2020). Adapting Student IDEs for Blind Programmers. *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, 1–5. https://doi.org/10.1145/3428029.3428051

[Schweikardt & Gross, 2006]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

[Scratch] "Scratch - Imagine, Program, Share." https://scratch.mit.edu/ (accessed Apr. 27, 2020).

[Seo2017] Seo, W., & Jung, H. (2017). Exploring the Community of Blind or Visually Impaired People on YouTube. Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility, 371–372. https://doi.org/10.1145/3132525.3134801

[SEELS 2014]]Special Education Elementary Longitudinal Study. 2004. SEELS Wave 3 Student School Program Questionnaire Educational services and supports Table 64 Estimates. *SEELS data tables*. Retrieved December 4, 2020 from https://www.seels.net/search/tables/21/sp3b5bfrm.html

[Soloman2020] Cynthia Solomon, Brian Harvey, Ken Kahn, Henry Lieberman, Mark L. Miller, Margaret Minsky, Artemis Papert, and Brian Silverman. 2020. History of Logo. Proc. ACM Program. Lang. 4, HOPL, Article 79 (June 2020), 66 pages. DOI:https://doi.org/10.1145/3386329

[Stefik2011a] Andreas Stefik, Susanna Siebert, Melissa Stefik, Kim Slattery. An Empirical Comparison of the Accuracy Rates of Novices using the Quorum, Perl, and Randomo Programming Languages. Workshop on the Evaluation and Usability of Programming Languages and Tools (PLATEAU 2011). Portland, OR, October 24th, 2011. https://dl.acm.org/doi/10.1145/2089155.2089159

[Stefik2011b] Stefik, A. M., Hundhausen, C., & Smith, D. (2011). On the Design of an Educational Infrastructure for the Blind and Visually Impaired in Computer Science. *Proceedings* *of the 42Nd ACM Technical Symposium on Computer Science Education*, 571–576. https://doi.org/10.1145/1953163.1953323

[Stefik2013] Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. ACM Transactions on Computing Education 13, 4, Article 19 (November 2013), 40 pages.

[Swift] “Swift Playgrounds - Apple.” https://www.apple.com/swift/playgrounds/ (accessed Jul. 01, 2020).

[Thieme et.al, 2017] 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

[Townend et al. 2016]] Gillian S. Townend, Peter B. Marschik, Eric Smeets, Raymond van de Berg, Mariëlle van den Berg, and Leopold M. G. Curfs. 2016. Eye Gaze Technology as a Form of Augmentative and Alternative Communication for Individuals with Rett Syndrome: Experiences of Families in The Netherlands. *J. Dev. Phys. Disabil.* 28, (2016), 101–112.

[Voykinska2016] Voykinska, V., Azenkot, S., Wu, S., & Leshed, G. (2016). How Blind People Interact with Visual Content on Social Networking Services. Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing - CSCW ’16, 1582–1593. https://doi.org/10.1145/2818048.2820013

[Weintrop2015] D. Weintrop and U. 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*, 2015, pp. 199–208.

[Wilkinson et al 2012] Krista M. Wilkinson, Janice Light, and Kathryn Drager. 2012. Considerations for the composition of visual scene displays: potential contributions of information from visual and cognitive sciences. *Augment. Altern. Commun.* 28, 3 (September 2012), 137–147.

[YouDescribe] *YouDescribe—Audio Description for YouTube Videos*. (n.d.). Retrieved December 4, 2020, from https://youdescribe.org/