Block-based Abstractions and Expansive Services to Make Advanced Computing Concepts Accessible to Novices

Corey Brady[a], Brian Broll[a], Gordon Stein[a], Devin Jean[a], Shuchi Grover[b], Veronica Cateté[c], Tiffany Barnes[c] and Ákos Lédeczi[a]

[a]*Vanderbilt University, Nashville, Tennessee,*
[b]*Looking Glass Ventures, Palo Alto, California,*
[c]*North Carolina State University, Raleigh, North Carolina,*

ARTICLE INFO

Keywords:
Block Based Programming
NetsBlox
Distributed Computing
Message Passing
Remote Procedure Calls

ABSTRACT

Many block-based programming environments have proven to be effective at engaging novices in learning programming. However, most offer only restricted access to the outside world, limiting learners to commands and computing resources built in to the environment. Some allow learners to drag and drop files, connect to sensors and robots locally or issue HTTP requests. But in a world where most of the applications in our daily lives are *distributed* (i.e., their functionality depends on communicating with other computers or accessing resources and data on the internet), the limited support for beginners to envision and create such distributed programs is a lost opportunity. We argue that it is feasible to create environments with simple yet powerful abstractions that open up distributed computing and other widely-used but advanced computing concepts including networking, the Internet of Things, and cybersecurity to novices. The paper presents the architecture of and design decisions behind NetsBlox, a programming environment that supports these ideas. We show how NetsBlox expands opportunities for learning considerably: programs can access a wealth of online data and web services, and they can communicate with other projects. Moreover, the tool infrastructure enables young learners to collaborate with each other during program construction whether they share their physical location or study remotely. Importantly, providing access to the wider world will also help counter widespread student perceptions that block-based environments are mere toys, and show that they are capable of creating compelling applications. Finally, we hope to show that tools can be designed to democratize access to powerful ideas in computing.

1. Introduction

There are many block-based educational programming environments designed to make programming accessible to novices. With inspiration from Logo [43], block-based environments have been popular tools for introducing programming and computational thinking (CT) to young learners. Moreover, research has shown that important aspects of this enthusiasm are well grounded in empirical effectiveness of block-based environments to support learners in comprehending code [51]; exploring and conceptualizing what is possible [49]; building self-confidence [31, 37, 42, 51]; and developing algorithmic and computational thinking [20, 21].

As the strengths of blocks as a representational infrastructure and medium for learning gather an empirical basis, additional questions emerge about how this innovation might enable a *restructuration* [52] of the conceptual domains that students engage with in their introduction to programming, computational thinking, and computer science. We propose that *distributed computing* is a promising domain, containing a family of powerful ideas [35] that can be made conceptually accessible through the block-based restructuration. We argue that this shift would not only be educationally meaningful, but that it also could offer an increase in the power and social relevance of student projects, which could counteract impressions that students (and teachers) can harbor about block-based programming as inauthentic [13] or otherwise limited [50].

However, a key design limitation of many block-based learning environments impedes the field's ability to explore these conjectures. Specifically, many environments for learning programming keep students and their projects confined within the tool. This paper argues that removing these walls can be highly beneficial—both to be able to teach more advanced concepts and to broaden participation in computing among young learners.

We propose a design for a block-based environment that highlights and leverages analogies between fundamental ideas of distributed computing on one hand (including Remote Procedure Calls and messaging) and sociable human communication on the other. In this approach, mechanisms of distributed computation are expressed analogously across their many manifestations (including facilitating communications between human users, calling encapsulated procedures, managing inter-process communication, sending commands to embedded devices like robots, and receiving data from IoT sensors), in ways that, we argue, both build upon and extend the core strengths of block representations.

Alongside this philosophical approach, we take a distinctive perspective on implementing an extensible, connected, and collaborative environment to promote and illuminate novice programmers' learning of distributed computing. In particular, over the past several years we have explored

ORCID(s): 0000-0002-4086-9638 (C. Brady); 0000-0003-4549-0333 (B. Broll); 0000-0002-1422-2814 (G. Stein); 0000-0001-9549-2324 (D. Jean); 0000-0001-6633-8862 (S. Grover); 0000-0002-7620-7708 (V. Cateté); 0000-0002-6500-9976 (T. Barnes); 0000-0003-2876-585X (Á. Lédeczi)

the learning potential of providing uniform support, in the form of a few intuitive abstractions, to open up block-based programming so that students can create truly distributed applications. We have found that construction environments can capitalize on this opportunity by taking three key design commitments to heart.

A first design commitment is that modern environments need to be easily extensible and afford loosely coupled, easily-discoverable methods of integration with external resources such as web APIs. Adding a new resource should require no code changes or user interface changes on the client (i.e., no new blocks). This not only reduces the implementation effort required but also presents the external resources in a uniform, predictable way to the young learners.

Second, environments should support methods of communication between projects. Distributed computing is ubiquitous both generally and in the applications popular among today's youth. Block-based environments, designed to make computing accessible and engaging, seem to be missing a crucial opportunity when they restrict learners from creating "social" applications that leverage the internet for communications and real data sources for broad engagement.

Finally, collaborating with peers can be fun and engaging, and it can also improve learning [39] and tap into the identity-building value of computational participation [23]. Furthermore, collaboration and teamwork are vital parts of industry applications. Supporting equitable collaboration that goes beyond co-located pair-programming can help to promote engagement and valuable 21st century skills, and also to dispel misconceptions about software being developed in isolation.

For the rest of the paper, an extended version of our conference publication [8], we use the open source NetsBlox tool [33, 9] to demonstrate how advanced distributed computing concepts can be made accessible to novice programmers. We begin with an overview of related work that has aimed to use visual programming environments to make core concepts and practices in computer science accessible to novices (Section 2). We then introduce the innovations central to NetsBlox's approach to provide a conceptual introduction to the powerful ideas of distributed computing: Remote Procedure Calls and Messages (Sections 3 and 4). Next, we show how these ideas allow exciting application areas in educational computing to be reframed in terms of distributed computing, including physical and virtual robotics; Internet-of-Things sensing and location-aware mobile computing; and voice-assistant integration (Sections 5-7). Then, we show how NetsBlox's foregrounding of connectivity also enables novel forms of collaboration at small-group and whole-class levels, supporting teachers in the challenging task of facilitating and coordinating computing activities (Section 8). Next, we show how NetsBlox can maintain its status as a "high-ceiling" [35] environment, through architectural extensibility and a pedagogical commitment to supporting students as they transition from blocks-based, visual programming to text-based languages like Python (Sections 9 and 10). NetsBlox's design thus enables it to be a flexible and accessible construction environment for learners to create personally-meaningful projects that use the lens of distributed computing. In the last three sections, we give examples of extended use and evaluation studies that (a) indicate that NetsBlox has delivered on its design commitments, and (b) lay the groundwork for its ongoing research agenda (Sections 11-13).

2. Related Work

Scratch [30] is arguably one of the most popular tools among block-based programming environments. Although it was not the first visual environment designed for younger learners (Alice [12] and Agentsheets [38] predate it), Scratch owes its popularity in large part to making programming accessible through visual programming, creative effects, and affordances that help novice programmers avoid many pitfalls while also encouraging engagement and creativity. It facilitates the creation of "Scratch extensions" with blocks that bring new capabilities to the environment, including language translation and support for interacting with a number of physical devices, such as Micro:bit [40] and Makey Makey [11]. At the time of this writing, there are 11 supported extensions: 6 for interacting with physical devices, 2 related to language, and 3 providing custom blocks for local capabilities such as drawing or playing music. However, with each of these extensions, Scratch brings in a number of new blocks, which can make it harder to find blocks and may steepen the learning curve. Scratch supports limited distributed data sharing via Cloud Variables that enable instances of the same program to share variables.

Snap*!* is a conceptual descendant of Scratch designed to support more advanced features including first class lists and functions, as well as to provide richer support for custom blocks [22]. Snap*!* also allows for extensions (e.g., to physical devices via libraries), and it provides a block for making HTTP requests. However, processing the information returned by such requests is anything but intuitive, making internet-connected applications brittle and adding unnecessary complexity that block-based environments are designed to remove in the first place.

BlockyTalky [24], used largely in research settings, supports the development of distributed applications for devices like the Raspberry Pi [48] and Micro:bit [40]. It facilitates communication between the devices allowing network messages which can be sent to a given IP address and port, but it does not support generic internet access or the creative programming elements present in both Scratch and Snap*!*.

MIT App Inventor is designed for development of mobile applications [36] and consists of "Designer" and "Blocks" editors. The "Designer" editor is used to add components to the app's user interface and the "Blocks" editor is used to program the app's behavior. App Inventor has native support for HTTP requests, and for communicating with Lego Mindstorms and Firebase [32]. Additional capabilities are supported using "extensions" that consist of new types of components and their corresponding blocks, similar to

Scratch. (In contrast, we will see that NetsBlox uses a single self-documenting block, named "call," to provide access to a large number of online services and hardware devices.)

CloudDB, Internet-of-Things, and machine learning capabilities are supported as App Inventor extensions enabling apps to store data in the cloud, to communicate with various devices like Arduino, and to incorporate various ML-based pattern recognition capabilities. After adding a component from one of these extensions to an app, the user has the ability to configure the component accordingly. This may include providing a secret access token or URL for a web-based service. After the component has been added to the app, the corresponding blocks will be available in the palette of the "Blocks" editor. App Inventor also has some support for real-time collaboration and merging projects.

A recent addition to the App Inventor toolbox is support for the creation of Alexa skills, although currently this is available only through a forked version of the environment. This version changes the editor to add new programmable entities (i.e., Alexa skills) and provides a chat dialog for testing it. Creating Alexa skills in App Inventor is exciting, but achieving it by modifying the editor itself is not scalable. As we will see, NetsBlox has been able to add similar capabilities without needing to change the user interface or introduce any new blocks.

Today, web services are becoming a required topic to teach in some high school computer science curricula [5]. And Lim et al. believe that web services should begin to be taught in introductory computer science classes [28]. However, there have been difficulties in teaching web services without proper tools [2]. For instance, instructors Assunção and Osório found that when teaching web services to computer science undergraduates, students focused more on issues involving the configurations of tools for the course instead of the actual material [2]. With its simple implementation of the Remote Procedure Call (RPC) "call" block, NetsBlox allows web services to be taught as an easy-to-comprehend concept. The "call" block eliminates distracting issues of tooling and allows novice programmer students to focus more on the subject matter at hand without being overwhelmed—a key notion in introductory computer science programming classrooms.

While tool support for collaboration is generally lacking in educational programming environments, educators still try to encourage their students to work together. Collaboration in most block-based programming curricula (such as Snap*!* for the popular Beauty and Joy of Computing course), is encouraged through side-by-side, driver-navigator pair programming [16]. This paradigm requires the driver to make edits to a program, while the navigator monitors the progress (e.g., by reading instructions or requirements). While this activity structure is sometimes very effective, greater flexibility in collaboration enables a wider variety of pair programming formats. In particular, built-in tool support in NetsBlox enables pair programming in which partners do not have to be co-located, and it also opens the door for other models of collaboration.

In summary, most existing environments lack 1) a uniform and intuitive way to access resources on the internet, 2) general support for distributed applications, and 3) flexible, synchronous and asynchronous collaboration support. Importantly, these shortcomings correspond to key ideas in distributed computing. Addressing them in a unified and conceptually coherent manner not only delivers the benefits of each of the features; it communicates a vision of distributed computing through the block syntax, the programming interface, and the presentation of distributed services and functions as 'first class' members of the toolkit (as opposed to special-purpose extensions that are peripheral to a core set of functionality centered upon local resources). In the next section, we will introduce NetsBlox's approach to these challenges.

3. Online Data and Web Services

NetsBlox, built on the open source codebase of Snap*!*, introduces a simple abstraction to provide conceptually simple access to online data sources and web services. *Remote Procedure Calls (RPC)* allow users to invoke functions running remotely on the NetsBlox server and provide results. The code on the NetsBlox server invokes public web APIs, but it also does additional work such caching and parsing the data received before returning it to the NetsBlox client as return values that correspond to data types native to the environment (e.g., numbers, strings, images, and lists).

Related RPCs are grouped into *Services*. Examples are Google Maps, Weather, Earthquakes, the Movie Database, and many others. Not all of the Services wrap web APIs to third-party providers. Additional Services that run exclusively on the NetsBlox server and do not require external support include a Gnuplot-based chart service, server-side support for various games, access to WiFi-enabled hardware devices and a hierarchical key-value store called Cloud Variables.

How much end-user complexity does it involve to access this much functionality? Won't users get overwhelmed and confused by this? RPCs use a ***single block*** named "call." Furthermore, the block is self-documenting. It has two pull-down menus, one for the Service and one for the RPC. See a subset of the Services available in Figure 1.

When a Service is selected, the second menu reconfigures itself to show the RPCs available within the selected Service. See Figure 2 for the RPCs of the Google Maps service. When an RPC is selected, slots for the required input arguments appear along with their names. See a few examples in Figure 3, which return values of type text, number, image, list, and multi-dimensional array. In addition, Service- and RPC-specific documentation is available by context-clicking on the call block and selecting help. While most of the remote procedures that can be invoked are provided to the users as is, there is one where the user can supply their own code for execution on the server, the call RPC of the Execute Service (see the top example in Figure 3).

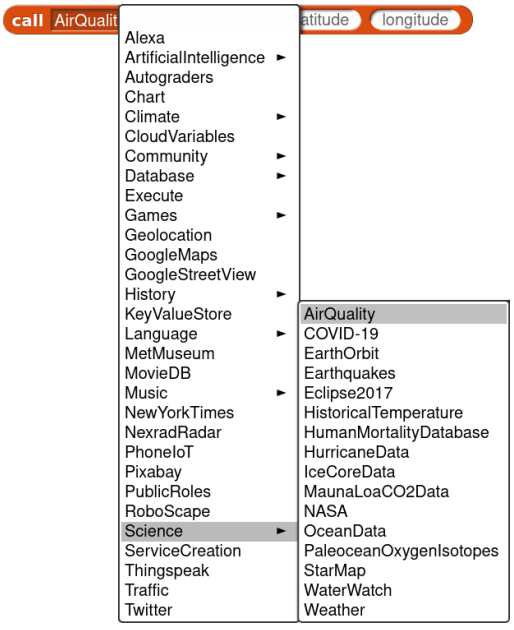


Figure 1: Services at the root menu and science sub-menu

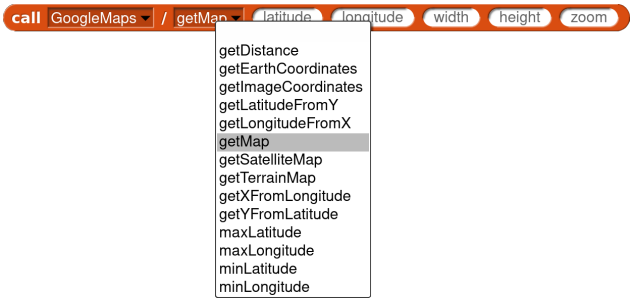


Figure 2: RPCs of the Google Maps service

Presenting Services and RPCs in this way foregrounds the idea that the role of the student's program is analogous in accessing any of these remote resources. Communicating this similarity of role through a single call interface sets the stage for seeing the similarity between RPCs that provide interfaces to cultural heritage databases, live sensor streams, or physical or virtual robots. It integrates the full range of remote resources coherently into the conceptual ecosystem of NetsBlox, an important step for thinking in terms of big ideas of distributed computing.

At the same time, it should be noted that the dynamically-reconfiguring nature of the call block confronts a potential tension with the block metaphor. Specifically, the number and nature of arguments to different RPC calls, as well as their return values, are highly variable, in contrast with the structural fixity of blocks under the standard metaphor of traditional visual programming environments. Here, NetsBlox stretches the block metaphor as a *syntactic* object, in order to emphasize *semantic* analogies. On one hand, two different call blocks cannot necessarily be swapped into the same "rounded-rectangle" hole in a given program for processing

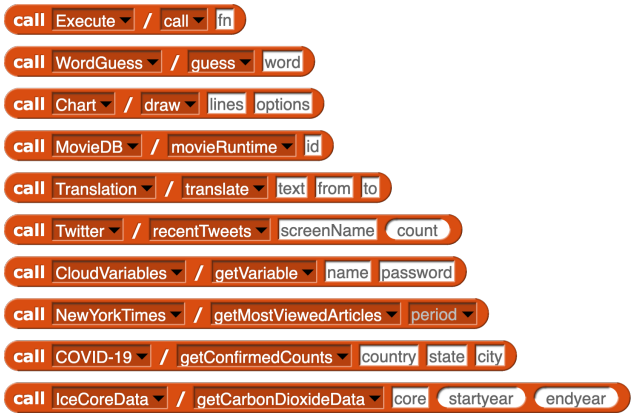


Figure 3: Example RPC calls

return values. But on the other hand, the similarity of all of the examples of Figure 3 communicates the parallelism in making remote requests and receiving responses. And the "lively" nature of the environment enables learners to click on a completed call block and see the response (See Figure 4). This enables the communicative aspects of a program to be constructed and iteratively explored before RPC responses are processed and integrated into program flow.

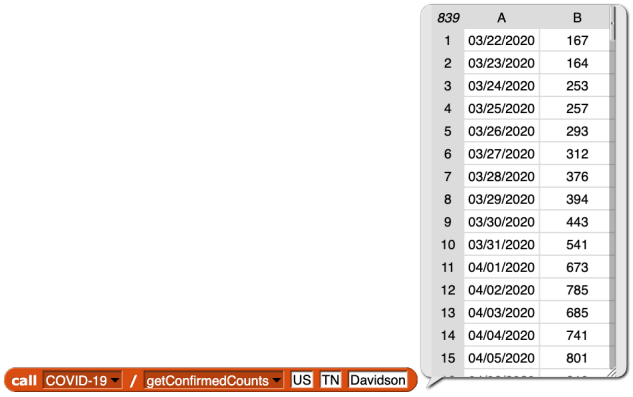


Figure 4: Exploring an RPC's return values by clicking.

To illustrate the simplicity and intuitive nature of the resulting semantic abstraction from the perspective of reading and understanding code, consider a 14-block program that shows a map of the local area of the user and displays the Air Quality Index (AQI) anywhere the user clicks (Figure 5 and 6).

It is not necessary to know anything about NetsBlox or read comments to understand what the code does and how it works. (To make the background into a pan-able and zoom-able fully interactive map of the world requires only 20 additional blocks.)

While the primary purpose of RPCs is to provide access to resources on the internet for student programs, one can also view them as a way to extend the built-in capabilities of

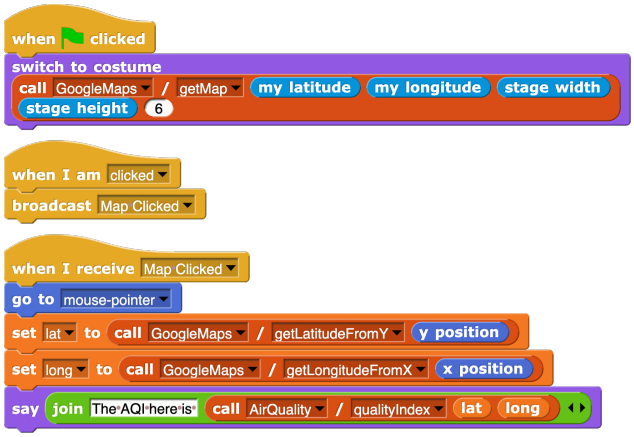


Figure 5: Current air quality project. Top two scripts: stage. Bottom script: sprite.

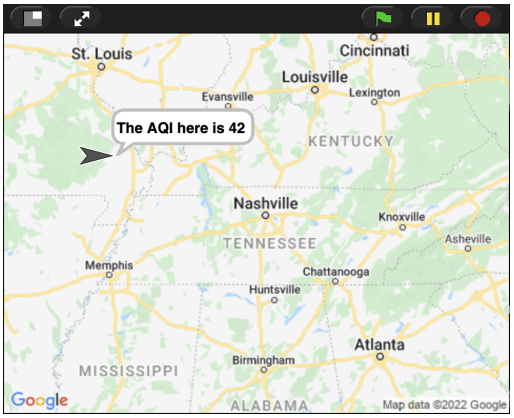


Figure 6: Running the current air quality project.

NetsBlox. In that, they are similar to extensions in Scratch and libraries in Snap*!*. The "call" block is also a truly powerful abstraction for writing code. Using a single generic block that configures itself according to context removes the cognitive load of learning a new set of blocks for every Service (extension or library). It also eliminates palettes full of new and unfamiliar blocks that would require searching for just the right one. The menu based interface employs hierarchical decomposition to arrange RPCs making them quickly discoverable and enabling experimentation. Services are grouped by categories, such as Science or Games, so users can quickly explore what's available.

Contrast that with Snap! libraries: users need to use the file menu, libraries option to import a given library. The action loads a potentially large number of blocks into one or more palettes (e.g., motion, control, etc.). The user would have to inspect the various palettes to see what new blocks became available. Some of these blocks are commands, others are variables or reporters. Input slots do not have names but they may show default values. While some libraries and their blocks are obvious how to use, others are more complicated. If the user decides not to use the library or only needs a block or two, they have to use the "Unused blocks" menu command and corresponding pop-up dialog to remove the clutter from their command palettes.

Finally, the call block and its menu based abstraction is "backward" compatible with the library approach. One can create libraries of custom blocks that wrap RPC calls and extend their functionality if desired. For example, the MovieDB service, providing access to information on tens of thousands of movies, has over 60 RPCs. We created a library that has 21 custom blocks for the most important functionality.

Another factor that makes the RPC concept familiar to students is that it closely resembles custom blocks. Both of them have multiple inputs and a single output, and are blocking calls that cause the program to wait for the result. The only difference is that RPCs run on the server.

As mentioned above, RPCs return data in the form of numbers (e.g., temperature), text (e.g., city name), lists (e.g., movie IDs), multi-dimensional arrays (e.g., geolocation search results), or images (charts, maps, movie posters, etc.). These are built-in data types and students are already familiar with them. Users do not need to de-serialize the data, parse text, or process a JSON data structure to extract useful information from results, unlike with HTTP calls available in some other tools.

Services allow students to create projects that utilize a wide array of information freely available on the internet. Many of these are sources that students already use in their daily lives (e.g., maps, weather, movie ratings, etc.). Others are related to topics they may care about, such as climate change or sports. Helping students create projects tied to their interests and related to real world issues will increase their motivation to learn to program and make programming and computer science more relevant to them [14].

4. Communication

Teenagers spend a lot of time on social media and with online multi-player games. What kind of support would a programming environment need to let them ***create*** such applications as opposed to just consuming them? Message passing is probably the most important abstraction in distributed computing. We incorporated it into the fundamental design of NetsBlox to enable projects running anywhere on the internet to be able to communicate with each other.

Messages in NetsBlox are very similar to events in Scratch and Snap*!*. However, messages are more powerful, as they can carry data, and they do not have to stay within the project; they can travel to any other NetsBlox project that is running anywhere on the internet at the time of sending. Messages have types, defined by a name and the data the message is to carry (i.e., a set of input slot names). Message type definition is done similarly to how one defines a custom block header in Snap*!*. See Figure 7 for an example.

Only two blocks are needed for message passing: one for sending and one for receiving. Selecting a message type in the "send" block pull down menu reconfigures it to show the corresponding input slots with their names provided.

Figure 7: Message type definition

Similarly, selecting a message type in the "when I receive" receiver hat block shows the same fields as variables, just like a custom block definition does (Figure 8).

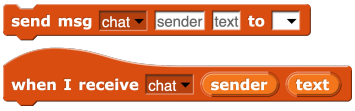


Figure 8: Message passing blocks

Message data can be built out of any of the data types supported by the environment—even scripts that the receiver can later run! The data is not strongly typed, so the sender and receiver must agree on what the message means. In particular, if the two projects need a prescribed interaction pattern, then their authors need to agree on a sequence of different messages of typically different types. In other words, they have to design a protocol. However, for simple applications, a single message type suffices, and this provides a powerful entry point. As their applications and communications gain in complexity, the message construct scales with the students' needs. Consider Figure 9, showing a six-block chat application (using the 'chat' message type defined in Figure 7). Two or more students can run instances of the project shown to chat with each other.

In embodying key concepts of distributed computing, defining and using messages plays a complementary role to using the call block to invoke RPCs. With messages, learners are put in the position of designing APIs and protocols, and as they develop experience with NetsBlox projects that both send and receive messages, they gain a more symmetric perspective on request-response relations among distributed networked processes.

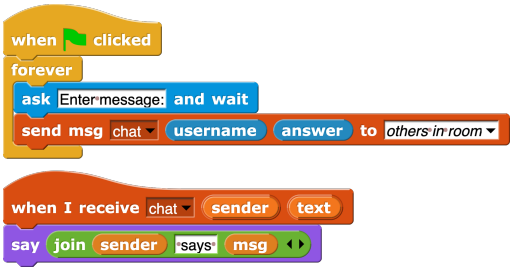


Figure 9: Simple chat app

Another important concept in message passing is addressing. The sender must specify where to send the message. NetsBlox supports local addressing. A NetsBlox project can consist of multiple subprojects, each of which plays a *Role* in the project, while the project itself is referred to as the *Room*. (This naming convention comes from the fact that we anticipate many students will use message passing for creating multi-player games.) Subprojects can be assigned to different users to run on different machines, e.g., play a game against each other. In turn, messages can be sent to any role or group of roles within the same project. For example, in the chat application in Figure 9, the chat messages are sent by any one role to all the other roles, i.e., to *others in room*.

Messages can also be sent to any running application. A globally-unique address is constructed by the tuple (username, project name, role name), since each of these is guaranteed to be unique within its context. (The role name is optional. If a role is not specified, messages to the project are delivered to all its Roles.) We call this global address a public role ID, since local addressing is done using role names. There is a Service and an RPC that returns the public role ID of the currently running project, so that students do not have to manually type it in. This approach also allows sharing and running the same project by different users without having to update the address manually. Global addressing is useful when one wants to support a dynamic number of participants in a distributed application. For example, the simple chat application of Figure 9 can be extended to be an actual chatroom that users can dynamically join and leave.

In this case, there are two separate NetsBlox projects: a chatroom server and a client. The protocol requires two message types: one, called 'connect,' for the client to register with the server by sending its own address; and another, called 'chat,' for sending actual chat messages. The server maintains a list of client addresses by handling the connect messages and simply rebroadcasts any chat messages to the list of addresses it has (Figure 10).

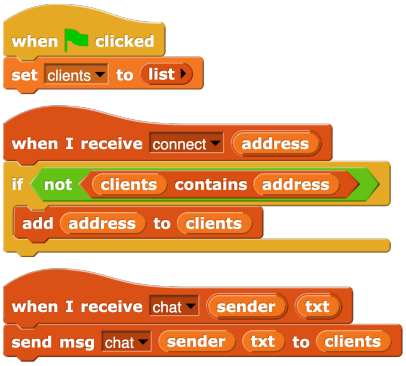


Figure 10: Chatroom server

The client sends its own address to the server and then, in a forever loop, it asks its user to type in a message, which it then sends over to the server. When it receives a chat message (routed from the server), it simply displays the original sender and the message text. See Figure 11. For simplicity, we omitted a more involved refinement, to display the last several messages as opposed to just the last one.

In the classroom, we typically explain the task, describe the client-server approach and introduce global addressing. Then we show the chatroom server program and run it on

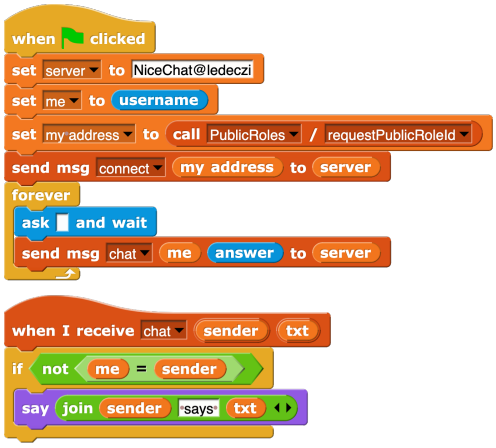


Figure 11: Chatroom client

the teacher's computer showing on the projector screen. The students are then asked to implement the client. This has proven to be an engaging activity for young learners.

Other examples of illustrative distributed computing applications using message passing would be peer-to-peer networking or volunteer computing–a citizen science project where a server (also a NetsBlox project) divides up a parallel complex problem into small tasks and distributes them to volunteer workers (running the same NetsBlox "client" project). The famous NASA SETI@home project [15, 25] is a real-world example of this kind of distributed computing.

These examples illustrate that the message passing abstraction in NetsBlox has a low threshold, enabling students to write non-trivial applications with just a few blocks. But it also has a high ceiling, allowing students to create complex distributed applications in a block-based environment. The abstraction hides a lot of the *accidental* complexity associated with message passing and networking, but it exposes the most important concepts of distributed computing, including message types, protocols, latency, and addressing.

Note that Services are not restricted to provide synchronous replies and can include message passing as well. For example, a call to the Earthquake Service may need to provide data on thousands of earthquakes in the requested geographic area. Instead of returning a huge multi-dimensional array in a single reply, the Service sends one message per earthquake, with each message containing the date, magnitude, and location of the earthquake in separate fields. The N-Player game Service that provides generic support for turn-based games also utilizes messages. For example, upon receiving an "end of turn" RPC call from one player, the Service sends a message to the player whose turn is next.

5. Robotics Reimagined

The traditional approach to educational robot programming requires a local connection to the device via USB or Bluetooth to download programs that can later be executed on the robot. NetsBlox takes a different approach, with the goal of allowing experiences of programming and interacting with robots to enhance and benefit from learners' emerging understanding of distributed computing. Under our approach, WiFi-enabled robots can connect to and register with the NetsBlox server directly via the internet. In turn, a Service called RoboScape allows NetsBlox programs to send commands to registered robots. The NetsBlox server handles routing commands to the robots' wireless connections. Each robot runs a command interpreter that executes these commands and can respond by sending messages back to through the NetsBlox server to the user's program. Such messages might contain, for example, requested sensor values.

This approach has several advantages. First, from the perspective of supporting distributed computing education, it connects robotics to the RPC and message-passing paradigm used elsewhere in the NetsBlox platform. Second, from the perspective of lowering barriers to educational robotics, the student's code is written and run from the NetsBlox project inside the browser, making it much easier to test and debug. Third, from the perspective of expanding robots' functionality, since student programs (and consequently all programs controlling robots), can communicate with each other, collaborative robotics becomes feasible.

Figure 12 shows a very simple remote control program that uses the keyboard to make a robot spin by making the two wheels rotate in opposite directions (s key); stop (space key) or beep at 400 Hz for 1 second (b key). The initialization code sets the robot ID to the last 4 digits of the MAC address and makes the robot beep, so that the user can check that the robot is now registered and 'connected' at this address. Note that the RoboScape service uses a generic 'send' RPC to issue text-based commands to the robot. The two input arguments are the robot ID and the desired command. Why not have separate 'set speed,' 'beep' and similar, more specific, RPCs? RoboScape intentionally makes it possible to "eavesdrop" on other students' communication with their robots and inject new commands. Specifically, when a robot receives a command, it sends an acknowledgement in the form of a NetsBlox message. Any NetsBlox project can subscribe to receive a robot's acknowledgement messages, not just the sender of the initial message. Moreover, the robots do not check who sent a command to them; they execute all valid commands received. These were intentional design decisions, intended to motivate the need for cybersecurity and to make the subject much more tangible and fun to learn. For example, students can encrypt the actual textual commands, to try to prevent adversaries from observing the commands they are sending to their robots or sending their own. The robots support a number of encryption schemes, but the NetsBlox program and the robot need to agree on the key first. Simply sharing the key sent in clear text makes it possible for others to intercept it when it is initially sent (an adversary will see a, for example, "set key 1 2 3 4" message acknowledgement from the robot) and break the encryption easily. That motivates the need for secure key exchange, which we provide through a hardware feature on

the robots. The cybersecurity functionality is largely implemented through the Service on the NetsBlox server, which allows robots to receive cybersecurity 'updates' without requiring actual firmware changes, reducing maintenance requirements for use with new curricula as they are created. There are a number of other cybersecurity concepts that the RoboScape Service helps to motivate and teach. We have carried out multiple successful and popular high school cybercamps built around wireless robotics and cybersecurity [26].

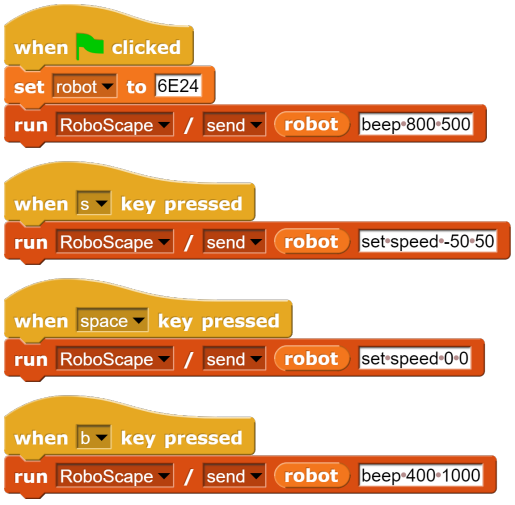


Figure 12: Simple robot control program using keyboard events

Since the student's running program and the robot do not have to be co-located, remote robotics also becomes possible. All one needs is a webcam streaming a video of the robot "arena" and multiple students can use the robots from their own homes.

NetsBlox's networking features can be combined with more traditional Bluetooth-connected robots to create additional remote robot programming configurations. For example, in the Spring of 2020, after the pandemic started, Birdbrain Technologies set up multiple Hummingbird robots and connected them to a laptop, where they run a NetsBlox program that controls the local robots using messages they receive from remote NetsBlox projects. Students are provided with various remote template projects that have the required message types already predefined and encapsulated in custom blocks. They can then use these blocks in their programs to control the robots remotely. The first command sends the "reserve" message that assigns the robot to the given user for a fixed amount of time, if it is available. The robot action is live-streamed in another window [4].

Finally, other configurations and collaborations between students and robots are possible; for example, using communications among NetsBlox projects running on students' computers to enable them to coordinate the actions of their Bluetooth-connected robots as they move about in a shared physical space.

Virtual Robotics

Real physical robots are fun, and students love hands-on activities. However, there is a paradox: robots are either inexpensive but simple, or powerful but expensive. Moreover, a collection of robots of any kind can be hard to maintain, especially in a school setting. Furthermore, connecting wireless devices to a school network can be problematic, as IT departments are often hesitant to provide access or support. Our work-in-progress RoboScape Online project responds to these challenges by providing a flexible, modular, and immersive virtual robotics environment. A group of students can share an instance of a virtual world with each student having their own virtual robots to program [44]. The virtual robots are programmed and controlled through the RoboScape Service, and students use the same blocks and commands to control virtual or physical robots. However, because the virtual robots' capabilities are simulated, there is no physical limit to what they can do. Yet, virtual robots are still tangible for today's youth, accustomed as they are to virtual experiences.

The simulated nature of the environment and the robots' functionality can make virtual robotics an ideal setting for introducing advanced computing concepts. This is because many of the constraints of physical devices can be controlled to serve pedagogical goals. In some learning settings it can be useful to remove physical limitations (such as noise in sensor signals or variability in actuators); while in other settings it can be useful to *amplify* them (e.g., when students are designing algorithms to be robust to these issues) [53]. Furthermore, the physical robots and their low-cost sensors are inherently unreliable for many autonomous tasks, creating extra difficulty for students. The simulated sensors and actuators are more reliable and have reduced latency for commands and responses sent over the network, further enabling lessons on autonomous tasks.

Virtual robots can offer a wide range of capabilities in terms of locomotion, sensors, and actuators. For example, they can 'contain' the most advanced hardware, from GPS to Lidar, without requiring any additional costs or configuration to make them available to programmers at any level. These sensors or actuators are exposed to students in the NetsBlox environment as additional Services, providing a simple interface familiar to students already working with the robots. For example, when students access an environment where their robots have a Lidar sensor enabled, they simply use a 'getRange' method on a 'LidarSensor' Service (see Figure 13). These devices are specified through code, so their functionalities can vary for different challenges, and new Services can be easily created. Virtual worlds can range from urban environments to deserts; from the open ocean to space. Without the physical limitations on what conditions can be represented, students can program robots in environments which would otherwise be impossible for them to access at all (e.g., a reactor with a radiation leak, or a space station orbiting a distant star). Visual content in environments can also be used to provide additional

Figure 13: Simulated sensors available as services.

context to an activity, such as to integrate cultural artifacts or historical information.

The virtual robotics capability was originally created as a standalone program made in the Unity game engine. While this provided a simple editor for allowing students to create their own environments and higher graphical quality, it was found that this approach lead to many of the same issues previously experienced with school IT departments and physical robots. Not only did the software require approval and support for installation, but it continued to require special assistance to allow it through school firewalls, and some schools provided hardware such as Chromebooks which were not necessarily compatible. Switching between the browser window with their NetsBlox code and a full screen graphical representation of the virtual world posed some frustration to students, who struggled to manage their windows to gain a clear view of both during activities. In addition, the server software, in order to take full advantage of the Unity editor, was required to run a graphics-less instance of the client software, which created significant costs to host the students' simulations. To alleviate these issues, the client software was rewritten to not only run in a web browser, but to be inserted into the NetsBlox interface itself, so students would be able to interact with the virtual world without having to move away from their code. Additionally, while Unity provided support to build the software for Windows, macOS, Linux, and mobile platforms, a browser-based system is compatible with all these platforms and more, while not requiring students to find the right version and install it. Most platforms capable of running NetsBlox should be able to add the virtual robots on top of it. WebSocket-based networking was found to generally be allowed by school IT departments, providing a method for real time updating of the simulation environment on students' browsers. The server software was rewritten to be significantly lighter-weight than the Unity version, allowing entire classrooms worth of students to share a small cloud server. The updated software has been tested in high school classrooms, with students appearing to enjoy the improved ease-of-use.

Through the RoboScape Online software, students can view the entire virtual world on the classroom projector or on their own computers/devices (Figure 14). They can also put on a VR headset to get a first-person view and a truly immersive experience (Figure 15). The RoboScape Online environment is designed to be accessible remotely, so students can share worlds regardless of their geographic location. All information of the robots' state is stored on the server, so a consistent virtual world is provided to all students. The environment looks just like a video game, except that students create and program it, rather than just

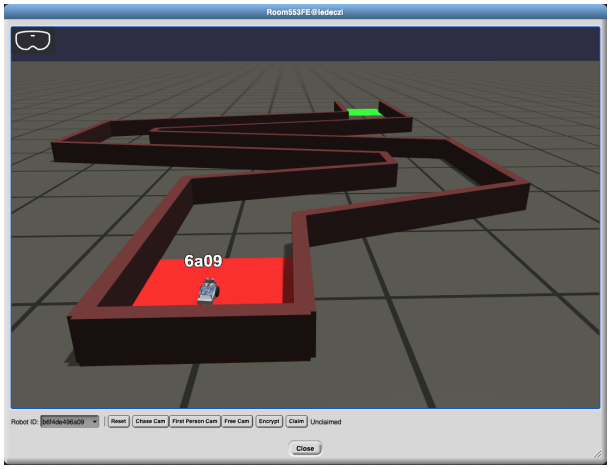


Figure 14: Bird's eye view of an environment where the students have to implement autonomous driving to navigate from the red to the green area using a simplified Lidar sensor.

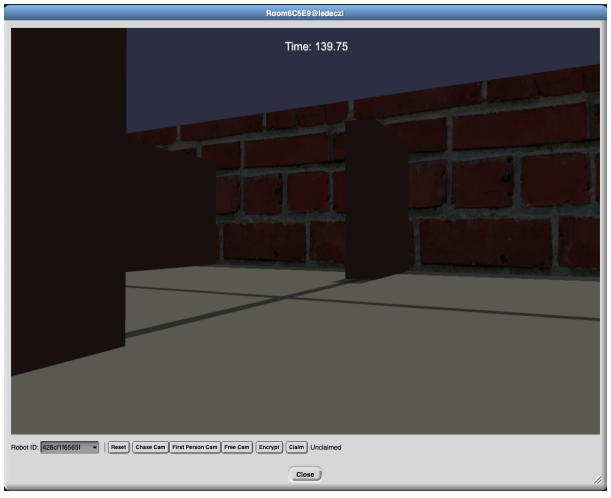


Figure 15: First person view of a maze environment from one robot.

playing within it. We have begun implementing RoboScape Online activities with high school students and in summer camps, where students work on robotics and cybersecurity challenges.

6. Mobile Device Integration

Many schools now offer makerspaces and other opportunities for students to get their hands on simple embedded computers, sensors, and educational robots. However, most schools still do not have access to these luxuries, and even those that do are typically limited by cost as to the number of and complexity of sensors and devices. Even when this hardware is available to students, schools typically cannot allow it to leave the classroom due to its expense; thus, these kinds of activities are necessarily in-person, and restricted to the school where the lab is located. However, almost every student in developed and even developing countries has a smartphone (or other mobile device) that contains a rich

collection of sensors connected to the internet out of the box. This presents an opportunity to teach concepts related to the Internet of Things (IoT), networking, and distributed computing in a manner that is not only accessible to novices but also highly engaging and motivating.

To make this approach a reality, we have created a mobile app called PhoneIoT, which allows the built-in sensors of the device to be accessed remotely from NetsBlox projects. Sensor data is made accessible through two popular networked sensor paradigms: polling, via RPC return values and streaming, via message passing. The list of RPCs providing sensor values is shown in Figure 16.

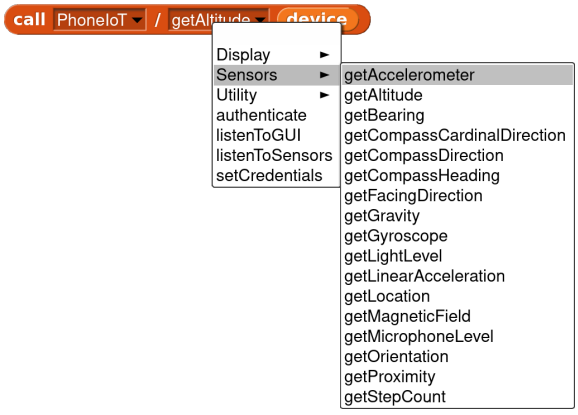


Figure 16: Sensors available through PhoneIoT

If a sensor is not present on the given device, is disabled, or is otherwise blocked by app permissions, it is simply logically disabled as a target for interactions, and calls to it from NetsBlox will return an error message explaining the problem. Streaming can be turned on by the 'listenToSensors' RPC, and the data arrives via various messages depending on the sensor requested. Figure 17 shows how to request acceleration data at 10 samples per second (i.e., with a 100 ms update interval), and the corresponding message handler hat block.

Figure 17: Streaming data from the 3-axis accelerometer

Note that in Figure 17 the 'listenToSensors' RPC is invoked through a puzzle-shaped **run** block (a command block), as opposed to the rounded-rectangle **call** block (a reporter block). NetsBlox's run block offers access to the same Services and RPCs as the call block, but it discards the return value. All RPCs give a return value (even remote commands reply with an "Ok" or error message), but as students write more complex programs, it is sometimes convenient for them to use the run block because it can be inserted directly into the program flow.

PhoneIoT also allows access to the touchscreen through a collection of customizable widgets that can send messages to a user's NetsBlox project when they interact with them. This makes it possible to configure a Graphical User Interface (GUI) on the phone from the very same NetsBlox program that processes the sensor data and handles asynchronous events from the mobile device by implementing message handlers. Hence, students can build truly distributed applications that run on two or more computers/devices connected via the internet and that interact with the physical world via sensors. To keep the interface to these features as simple as possible, controls are created and modified through individual RPCs such as "addButton" or "clearControls." The RPCs for creating and configuring the GUI elements are shown in Figure 18.

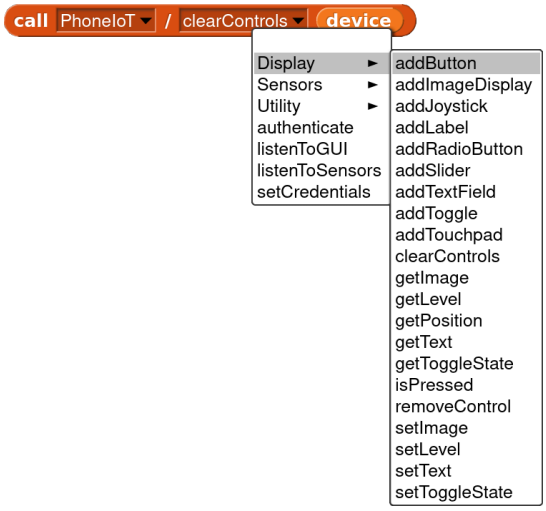


Figure 18: GUI widget creation and configurations RPCs

There are a few different approaches to creating native graphical applications that run on different operating systems. One is to logically replace the widgets with native equivalents and build a communication bridge for applying changes and receiving interaction events. Although this would make PhoneIoT apps look like native iOS and Android apps, it would also mean that PhoneIoT is restricted to only the intersection of widget and style options available to all platforms. Instead, PhoneIoT includes a simple, custom rendering engine that draws logical widgets on a canvas. This allows for total control over the layout and appearance of the controls, making it possible to support a broad range of style options, as well as interactive animated controls (e.g., joysticks and touchpads). This also allows PhoneIoT to look exactly the same on all devices, which could help reduce student confusion when working in team projects where group members have different types of phones.

When adding controls to the PhoneIoT display, the system must know the location and size of the new control; this is typically passed as four separate RPC input values for "x," "y," "width," and "height." These concepts already exist under the same names in NetsBlox, however the coordinate system is different. In NetsBlox, the center of the screen is

(0, 0), with increasing values going up and to the right, and a sprite's location is the position of its center point. However, PhoneIoT uses the more common graphical unit system where (0, 0) is the top left corner of the display, increasing values go down and to the right, and the location of a control is its top left corner, rather than its center. Although this coordinate system is new to students, it introduces a more real-world graphical coordinate system which could be applied to many other graphical environments outside of NetsBlox.

Once an app allows access to the device from the internet, security and privacy become important considerations. Each mobile device is identified by a unique 12-digit hexadecimal ID: their MAC address (or a random, persistent value if the MAC address is inaccessible). The PhoneIoT app generates an 8-digit hexadecimal password which expires every 24 hours. Both of these need to be provided by the NetsBlox project in order for PhoneIoT to accept connections.

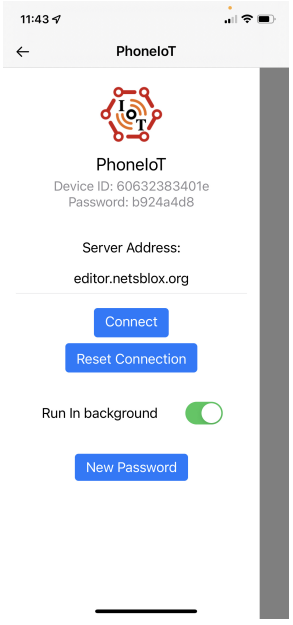


Figure 19: PhoneIoT configuration screen

Additionally, some sensors are intentionally limited by the app. For instance, only the current volume level from the microphone is provided (rather than the actual waveform). In addition, the app does not allow a network request to take a new picture from the camera without user confirmation on the phone itself. An image display widget when clicked, asks the user whether they want to take a picture. If they do, this and only this picture will become available to NetsBlox through the 'getImage' RPC.

Moreover, it would not be acceptable for a user to be unknowingly tracked or spied upon through the NetsBlox interface due to forgetting to close the app. Because of this, unless explicitly requested with the "run in background" setting in the menu, the app ceases all communication with the server and rejects all incoming requests upon being put into the background or turning off the screen. We believe these safeguards are sufficient to allow K–12 audiences to use the app while still affording them reasonable internet privacy.

It is worth emphasizing that integrating mobile devices into the NetsBlox framework with PhoneIoT required no changes on the NetsBlox client whatsoever. PhoneIoT introduces no new programming abstractions, no new interaction primitives, and no new blocks that might present barriers to students. Any user who is already accustomed to NetsBlox RPCs and message passing should quickly feel comfortable using PhoneIoT through its familiar interfaces. It is also important to note that PhoneIoT with NetsBlox is not a mobile app development environment. It simply treats mobile devices as intelligent, remotely controllable IoT devices, making it possible to create engaging distributed applications. Users can create a simple compass that displays the current heading on both the computer and phone screens. Or they can turn the phone into a remote controller for games. They can use the accelerometer to control a sprite by tilting the mobile device, or they can use buttons, joysticks and sliders to make a complex game controller as shown in Figure 20. Multiple devices, across multiple Services, can be linked to the same NetsBlox program, so it is also perfectly feasible to create a robot remote controller using PhoneIoT.

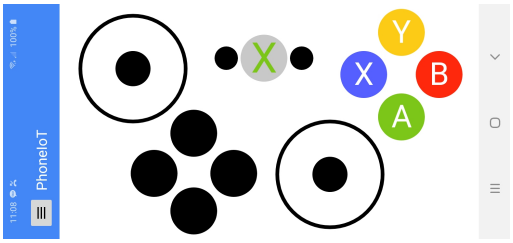


Figure 20: An XBox-like game controller made of buttons and joystick controls (rotated to landscape).

Exercise Tracker

To illustrate how PhoneIoT can be used to create powerful and engaging projects, we present a simple exercise tracker which plots the user's route on top of a Google Map displayed in the NetsBlox client, streams the updated display back to the mobile device, and prints the total distance covered as well. To illustrate the use of PhoneIoT's custom GUI controls, we also include start/stop buttons.

Figure 21 shows a portion of the initialization code. As mentioned above, for security the device ID and password displayed in the PhoneIoT menu must manually entered in the NetsBlox client code to establish the connection with the phone. If the correct credentials are provided, PhoneIoT will accept configuration RPC calls such as the ones in the figure, which clear the screen and add controls at certain coordinates and dimensions. The last two blocks in the figure enable GUI event messaging from the phone and request location data updates every 2 seconds (2000 milliseconds), respectively.

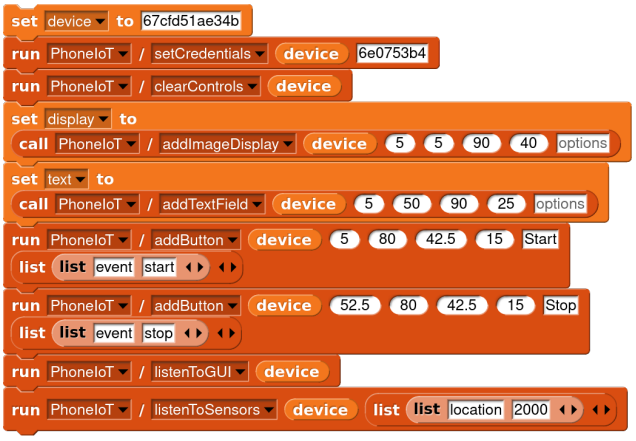


Figure 21: NetsBlox code to initialize communication with the PhoneIoT app and add GUI widgets on a mobile device

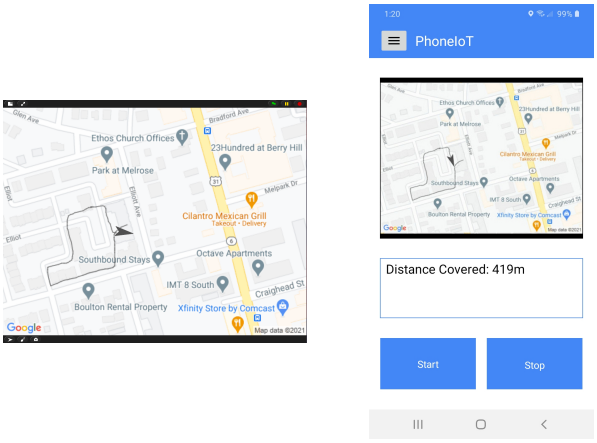


Figure 23: NetsBlox client stage (left) and phone display (right) of the exercise tracker app at slightly different moments

After initialization, the NetsBlox program receives location updates via the "location" message type and begins plotting the course. Each message provides the latitude and longitude (along with heading and altitude), which can be converted into screen coordinates with the GoogleMaps coordinate translation RPCs. See Figure 22. The "getDistance" RPC of the GoogleMaps Service provides the distance between two map locations, though one needs to perform some averaging to reduce errors due to the variation in reported location caused by GPS inaccuracy [3]. This is done inside the "add point" custom block (i.e., function) in Figure 22. The only other logic required is handling the stop and start events from the custom buttons on the phone. See Figure 23 for the final app screens in NetsBlox and PhoneIoT.

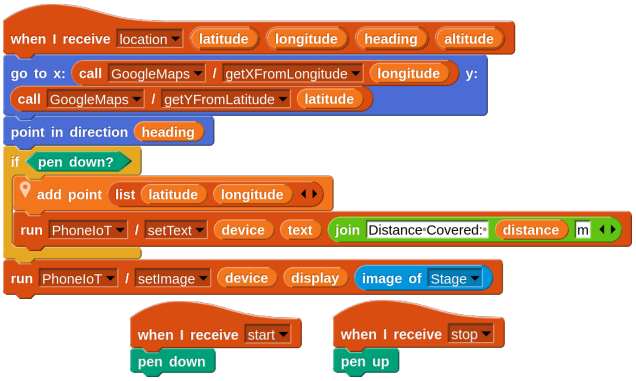


Figure 22: Exercise tracker code. The "add point" custom block maintains the distance covered.

7. Voice Assistant Integration

Enabling students to integrate voice assistants, like the Amazon Echo, into their distributed programs, is yet another way to make programming more compelling and meaningful for young learners. Especially when combined with the other NetsBlox capabilities, this creates many exciting opportunities for students. Students could make games where players control their characters with voice commands or even control network-enabled robots using the RoboScape Service RPCs described in Section 5. Even if they simply want to create a standalone skill, they are able to utilize many of the standard Services such as Weather, MovieDB, and Translation. Furthermore, this integration is possible using only the RPC abstraction (and first class functions) following the NetsBlox design commitment to loosely-coupled, easily discoverable methods of integration.

Before discussing integration with NetsBlox, it is important to provide a high-level overview of how spoken requests are handled by Alexa. Like many other voice assistants, it works by recognizing specific *intents* of the user. An intent is a verbal structure defined by a number of example utterances. These utterances can contain empty "slots" which are placeholders for values like names, locations, etc. The spoken words used for each slot are then passed as arguments to the handler for the specific intent.

NetsBlox provides an Alexa Service, a collection of RPCs for creating a skill from a configuration (defined as a 2D list), along with additional helper RPCs. Using this Service, users can define intents, give example utterances, and provide intent handlers as anonymous functions. When the 'createSkill' RPC is called, the NetsBlox server creates the skill for the given user and stores the handlers in its database. When a command is spoken to the Alexa skill, the request is handled entirely by the NetsBlox server using the appropriate user-defined block-based intent handler. That is, when an intent is received by NetsBlox, the user-defined intent handler is retrieved from the database, compiled to JavaScript, and executed with the received values for each slot. As the intent handlers can utilize the message passing blocks, they can be used to forward messages to student projects, such as games where the players are controlled via Alexa.

Not only does this allow young learners to develop Alexa skills but it also facilitates first-hand experience with

serverless computing, a contemporary distributed computing paradigm made popular through services like Amazon Lambda. The user-defined intent handlers are stateless functions compiled and executed by the server on demand. If the function requires shared state, cloud storage can be used explicitly via NetsBlox cloud variables. The ability to provide young learners with hands-on experience using contemporary computing paradigms in a manner that also simplifies the programming of devices like the Amazon Echo is an exciting opportunity for introductory computer science education.

As an example, consider Figure 24, showing the definition of a simple skill that provides information about atmospheric carbon dioxide concentrations measured on Hawaii by the NOAA. In addition to the Alexa service and its RPCs, a few custom blocks are provided to ease the cumbersome parts of defining a skill. The 'Alexa skill' block creates a data structure, a multi-dimensional array that contains required parts of an Alexa skill, such as its name, the various phrases that it accepts to perform its various actions, or intents. The 'when I hear one of' custom block specifies a similar data structure for one intent. These custom blocks shield users from having to assemble the necessary data structures and instead put the focus on the interesting part: the code that runs when the user invokes the intent via an Echo device.

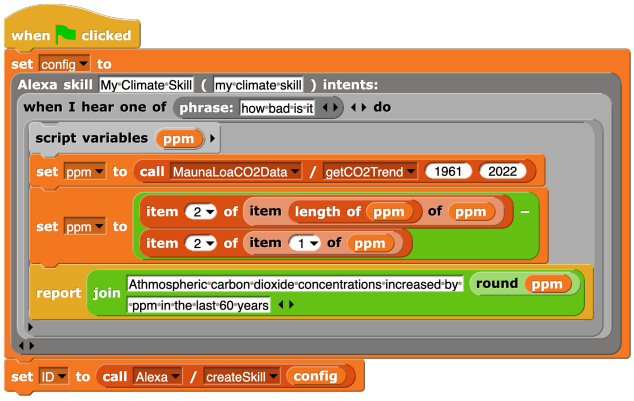


Figure 24: Climate skill

In this example, we use one of the climate change related Services, specifically calling the 'getCO2Trend' RPC to return a two-dimensional array containing the atmospheric carbon dioxide concentrations read at the Mauna Loa observatory between 1961 and 2022. We compute the difference of the first and the last elements of the second column and instruct Alexa to say the result in a sentence. The only Alexa RPC we need to call is 'createSkill.' This RPC configures the skill both on the Alexa and the NetsBlox servers. If we want to tweak the skill later, we can use the 'updateSkill' RPC, and we can even test it from within NetsBlox by using the 'invokeSkill' RPC.

Once ready, the user still needs to enable the skill via the Alexa app. To invoke this intent, the user needs to say: "Alexa, ask my climate skill how bad is it." In turn, Alexa will reply "Atmospheric carbon dioxide concentrations increased by 101 ppm in the last 60 years." It is important to note that the NetsBlox project that creates a skill does not need to continue running once a skill is deployed. After the initial creation, everything that the Alexa app needs is on the NetsBlox server.

In addition to RPCs, NetsBlox Alexa skills can use message passing as well. The example in Figure 25 shows a skill that can control a robot remotely, replicating the keyboard-based controls of Figure 12 with voice commands to trigger three message types: spin, stop, and beep. In this implementation, messages come to the NetsBlox project, and so this project must remain running, to receive messages from the Alexa skill and send the required commands on to the robot. We *could* have removed the NetsBlox intermediary and called the 'send' RPC of RoboScape directly, to turn, stop the wheels, or beep, but the implementation shown here has the purpose of illustrating message passing.

The skill definition in Figure 25 has three corresponding intents; one for spinning, one for stopping, and one for beeping. To respond to the intents, it accesses a cloud variable that stores the global address (public role ID) of the NetsBlox project actually driving the robot, and it sends the appropriate messages to this project. The code for the 'beep' intent illustrates how to include parameters in intent invocations. Another custom block allows the user to specify the data type and a name for the parameter, which the user can then place in the correct position within the phrase: "to beep at **pitch** hertz."

Figure 25: Robot driver skill

8. Collaboration

Once we remove the walls around our programming environments, it becomes possible to support collaborative programming on projects in flexible ways. In particular, NetsBlox allows users to issue and accept invitations to collaborate on a project. Collaborators at any location can then work on the same project simultaneously. Concurrent editing operations show up on everyone's screen. The

server resolves conflicting changes by approving the first one received, and rejecting subsequent ones. However, since the typical latency is under 100 milliseconds, this rarely happens.

Since NetsBlox stores only a single shared copy of the project, students can also work asynchronously. This is similar to how popular collaborative editing tools such as Google Docs or Overleaf work. However, there is a conceptual difference between static documents and dynamic, continuously-executing block-based code. The latter has a *state,* which includes the values of variables and the appearance and position of sprites and the stage. Since each user's computer *executes* the code independently, it would be hard, if not impossible, to synchronize program state. So NetsBlox only keeps the source code itself in sync across collaborators. That is, the scripts will be the same, but the appearance of the stage and the values of variables will typically be different across collaborators at any given moment.

NetsBlox's robust support for collaboration enables pair programming, team projects, remote tutoring, and remote collaboration. The latter two have been especially important for online learning during the COVID-19 pandemic. Recent research has shown collaboration in pair programming environments is conducive to the development of problem solving and programming skills in young women [41]. Using NetsBlox, we plan to investigate equitable methods of collaboration that promote student engagement and improve student learning objectives. Furthermore, NetsBlox's collaboration infrastructure also makes it possible to try out novel ways of teaching with collaborative activities. For example, designing for collaboration can involve assigning subtasks to collaborating students, an activity structure supported by NetsBlox. This can also help highlight problem decomposition—a key aspect of learning programming and computational thinking [19].

Activity Galleries

Another novel way to collaborate and share one's work with classmates and the teacher in NetsBlox is through Activity Galleries. Galleries enable members of a class or group to publish in-process or final-form NetsBlox projects to a shared space. A teacher can optionally initialize a Gallery with a starter project (in Figure 26 below, the starter project drew a square and invited students to generalize that code). This allows the class activity to focus on creating refinements or extensions to a basic program. When students are ready to publish their work, they simply click a green 'camera' icon, shown at bottom-left of Figure 26. After entering optional information about their submission, their work is stored on the server and a thumbnail immediately appears in the class Gallery (shown at left in figure 26). A published entry preserves the project's state at the time of publication, and anyone in the class can click on the corresponding thumbnail to view, comment on, load, and/or remix and republish the entry. Finally, Galleries are scoped to a particular group (e.g., a class section), so the group can build upon its own

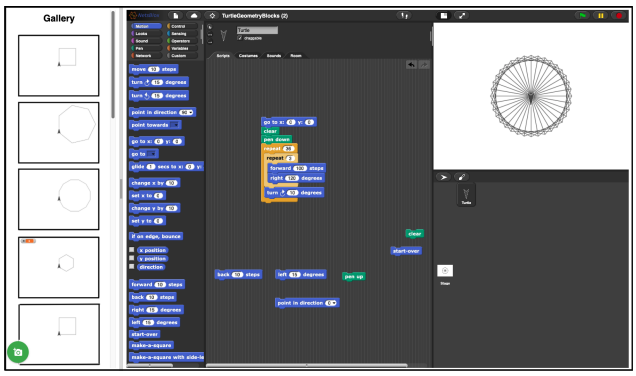


Figure 26: One class's Turtle Geometry activity gallery

members' insights, with students seeing, giving feedback on, and refining each other's ideas.

Activity Galleries enable group-based design [6, 7, 45], in which the diversity of thinking present in the group is seen as a critical resource for the functioning of the activity. Figure 26 shows a class using a Gallery to support the challenge, "Make a Polygon" starting from code that created a square. Different students took up the challenge in different ways, and the collective thinking grew increasingly more sophisticated. Here, for example, one line of inquiry pursued the question of 'star' polygons, like the one shown in the main workspace of Figure 26. In another line of inquiry, students began to introduce arguments or global variables to their initial solutions to make polygons with any number of sides. Later in the session they worked to make polygons with a fixed perimeter, and finally they considered ways to unite the two separate lines of inquiry, involving standard and star polygons.

A range of classroom activity types can be supported with Galleries; we describe three here. First, *Generative Activities* [46] encourage students to produce artifacts that reflect their own distinctive and creative ways of looking at a given problem or situation. After seeing the "space" of solutions the group generates, the class may reflect on how to generalize across solutions.

Second, in *Design Challenges* [27], students may share solutions to a more open problem. When the design problem is shared among students, the classroom group becomes an authentic audience for partial or complete solutions. Activity Galleries become more than a showcase of final projects; they offer a visual trace of individual and collective ways of thinking about the problem as they mature and interact.

Finally, combining aspects of the two activity types above, Galleries can be a good setting for testing and responsive refinement of code. For instance, when learning sorting algorithms, a Gallery might begin with the challenge to create and publish implementations of one or more algorithms. Next, students might be given the task to design input lists that would give the algorithm trouble. Working together as a class, they can develop characterizations of worst-case inputs for each algorithm, constructing rather than simply reading about these features of sorting approaches. Such

activities, in which students play roles of both algorithm-creating and algorithm-challenging, can also be used to introduce "adversarial" techniques that are themselves important topics.

9. Extensibility

The NetsBlox environment has two major components: the client and the server. The client has unlimited undo and redo support and the capability to replay the entire history of the project. This also serves as simple version control, since one can go back to any past point in the history and continue from there. The NetsBlox client also adds a new block category to Snap*!* called Network, containing the RPC block and blocks related to message passing.

Unlike most other environments, most NetsBlox functionalities are provided by the server, which runs the various Services, routes messages, and manages collaboration. The architecture of the server is modular (see Figure 27), facilitating extensibility. To add a new Service, only a single JavaScript file (based on a template) needs to be added. Some Services are as simple as a few lines of code, while others that provide more complex functionality can get large. However, all Services are well separated, with a simple API connecting them to the core.

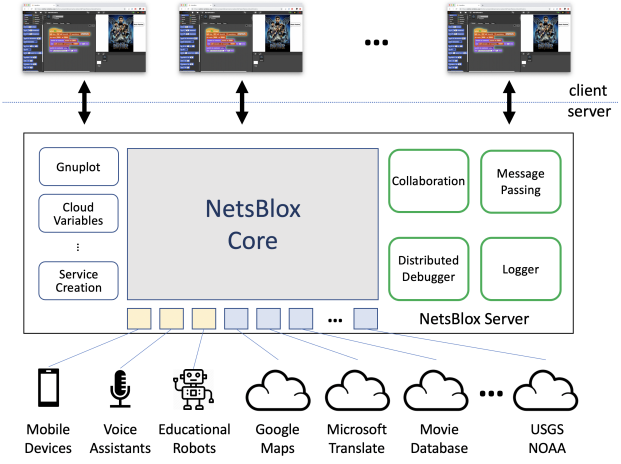


Figure 27: NetsBlox architecture

The power of this approach is illustrated by the fact that adding support for hardware devices in the form of the RoboScape Service that manages WiFi connected robots or the PhoneIoT app that connects to mobile devices required no change on the client side or the server core at all. Most importantly, students do not have to learn any new blocks when a new Service is added. All they see is a new, self-documenting option in the pull down menu of the "call" and "run" blocks.

It is important to note that the clients access the server via a well-documented, open RESTful API. Therefore, all the Services and message passing support are available to potential alternative clients that do not need to be block-based. For example, we are already working on a Python front-end (See Section 10, below).

Client-side Extensions

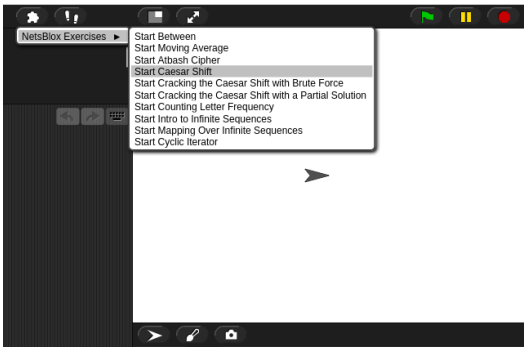
The client can be customized using NetsBlox extensions. A NetsBlox extension is similar to a browser extension in that it can contain arbitrary JavaScript that can access and modify the client itself. Extensions can be loaded automatically via URL parameters so they can be easily used to create more structured or customized experiences without any initial steps for users. To prevent malicious use, any extension hosted by an untrusted origin must be manually approved by the user (even when loaded via URL parameters).

NetsBlox extensions utilize an extension API which facilitates common patterns for customizing the client. These include creating custom primitive blocks and block categories. The API also supports the creation of a menu for each loaded extension. One example usage of NetsBlox extensions can be found in the creation and use of custom *autograders*. Autograders are activity-specific suites of automated tests. Using client-side extensions, the NetsBlox client was modified to allow users to select assignments for their course from a custom menu, grade them within the browser, and even submit them to platforms like Coursera using a Learning Management System (LMS) API.

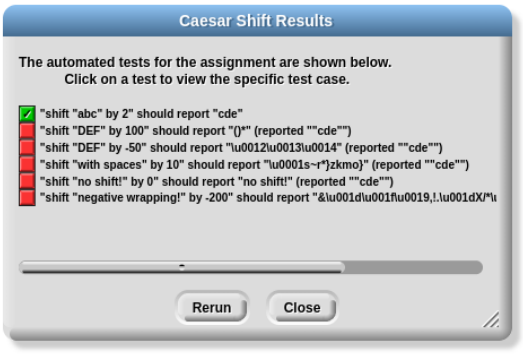
Figure 28a shows an assignment being selected from the autograder's extension menu. Upon selection, a starter template is loaded for the assignment along with the associated tests. Once the assignment is loaded, the extension menu is updated to include an option to grade the current assignment. This menu item will open the dialog shown in Figure 28b which displays the output of a set of automated tests for the assignment.

NetsBlox extensions make the creation of custom autograders possible but not necessarily accessible for instructors who may not be JavaScript developers (or have access to webserver to host the files). To lower the barriers to use, we have created a NetsBlox Service called "Autograders" and a custom block library for creating and sharing custom autograders within NetsBlox. The Autograders Service enables users to store configurations for their own autograders. The configuration is well documented, including an interactive video walkthrough on autograder creation. Autograders can be simply composed from the official, curated examples—complete with automated tests—hosted on GitHub. These are particularly easy to include as there is a pre-built library with each assignment configuration, available as a custom block.

Custom assignments can be defined using the custom block library. An assignment consists of a name, starter template URL, and list of tests. Tests can currently only evaluate custom blocks defined in the project, but the configuration is extensible so that more aspects of a project can be evaluated in the future. The custom block library enables users to define fixed input/output test cases for the given assignment or even provide custom code for more involved



(a)



(b)

Figure 28: Using a custom autograder to load an assignment (a) and grade a work-in-progress solution (b).

test cases. An example of each can be found in Figure 29. The autograder created here is called "Example Grader" and consists of a single assignment, which asks students to make a custom block that will perform the function, "Multiply by 2." The starter template for this assignment is provided by the "public project URL" block (a placeholder in the example). There are 2 tests for the assignment, which demonstrate two different ways to evaluate the same test case. The first block specifies a fixed input and output for the block; the second specifies a test which, given the student's implementation of the custom block, checks that when called with "2", it yields the expected result. After the autograder is created, the next block opens a new tab with the autograder loaded. This enables the user to test the autograder and, when satisfactory, share the URL that will automatically load the autograder with their students.

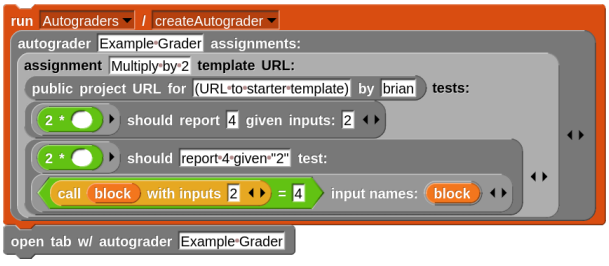


Figure 29: Creating a custom autograder in NetsBlox.

Make your own Service

There are multiple ways to add Services to NetsBlox. Since NetsBlox is an open-source project, contributors can implement their own Services in JavaScript and issue pull requests on GitHub. There is also a mechanism to create an extension server. The new server can be enabled for specific users or classes in NetsBlox, and the Services it provides will show up in the call block automatically. In fact, these additional servers do not even have to be implemented in JavaScript and can be in any language as long as they implement the RESTful API expected of NetsBlox Services.

However, most teachers are not software engineers, so there is a need to be able to add Services from the client side. To address this need, there is a NetsBlox Service called "Service Creation." This makes it possible to add one's own Service, which will then appear in the "call" block pull down menu under the Community and then *username* submenus, as shown in Figure 30.

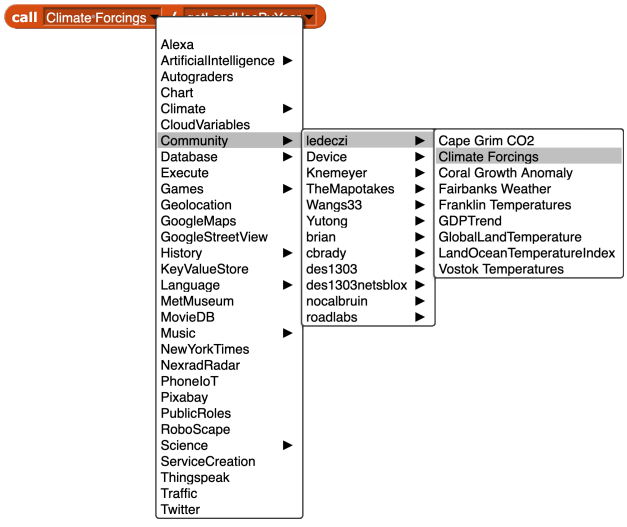


Figure 30: User-contributed services

To create a Service, one needs a CSV file with the data that the Service will supply. The user can drag and drop this file into the NetsBlox window to convert it into a variable (a two-dimensional array) automatically. The first row of this CSV file should contain column headers (e.g., "year," "Black," "White," etc.) as shown in the example in Figure 31.

The user can then pass the variable with the data as an input argument to the "createServiceFromTable" RPC shown in Figure 32. If no options are supplied, the RPC will create a new Service with a number of default RPCs: 1) 'getTable' that returns the entire table, 2) 'getValue' that returns a single element by two indices: the value in the first column and the column name, 3) one for each column of the table (e.g., 'getHispanicColumn' and 4) one for each column as a function of the first column (e.g., 'getAsianByYear'). This latter option is particularly useful when the data is some kind of time series and the first column contains the date/time values.

	A	B	C	D	E
1	Year	Black	White	Asian	Hispanic
2	1980	0	143	9	0
3	1981	2	164	16	0
4	1982	1	136	13	1
5	1983	3	175	22	1
6	1984	3	164	21	3
7	1985	3	177	17	6
8	1986	1	194	37	7
9	1987	2	230	27	4
10	1988	2	265	44	3
11	1989	1	320	53	4

Figure 31: Example CSV file containing the number of new CS PhDs in the United States over 1980–2020 broken down by ethnicity.

Figure 32: User-specified Service creation

The 'options' argument to ServiceCreation's 'createServiceFromTable' RPC enables the user to change these default behaviors; for example, to specify additional RPCs. It is even possible to provide a NetsBlox script for any of the desired RPCs, in which case NetsBlox translates these into JavaScript and they become the code associated with the RPC. Figure 33 shows the default RPCs generated from the CSV file in Figure 31. Figure 34 then shows a simple use of this new Service that plots the number of new CS PhDs by ethnicity.

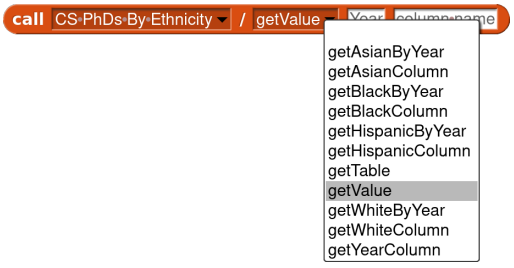


Figure 33: The newly-created Service

The ServiceCreation service is yet another feature that helps teachers and students to create pedagogically relevant and personally meaningful projects. Instead of emailing the students a data file, a teacher can create a new Service in a few minutes that becomes available to all students instantly. If some revision needs to be made, the Service can be updated just as easily, and all students, and existing projects, will have the latest data automatically.

10. Continued Learning with Python

As we have seen, block-based environments can be very powerful, with NetsBlox focused on offering an introduction to advanced CS topics such as distributed computing, cybersecurity, robotics, and the Internet of Things. However, one

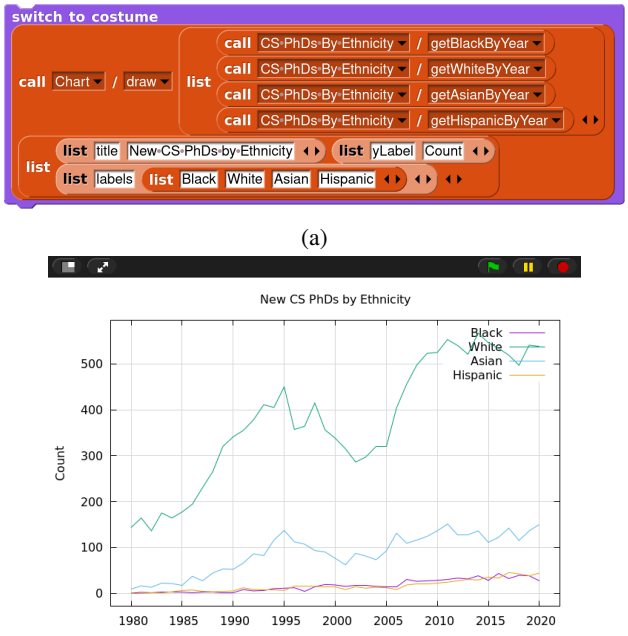


Figure 34: Code to plot the new CS PhDs data by ethnicity (a) and the resulting chart (b).

recurring criticism from advanced students is that, despite all of the powerful features, block-based environments can still feel more like a toy than a "real" programming language, simply due to its block-based interface [13]. Because of this, students might abandon tools like NetsBlox in favor of textual languages like Python. However, this is a massive change, and students might feel discouraged if they cannot easily reproduce the same advanced behaviors they used to be able to program in NetsBlox. To ease this transition while preserving most of students' existing NetsBlox project-based knowledge, we are developing a tool called PyBlox.

When moving from a block-based to a text-based environment, it is not just the programming language that changes, but also the underlying computational model and the IDE. The overarching design goal with PyBlox is to preserve as much of the NetsBlox experience as possible and replace only the programming language from blocks to Python. Hence, PyBlox is a Python-based turtle graphics environment with a stage, multiple sprites, multiple scripts per sprite, events, (almost) the same concurrency model as NetsBlox and RPCs and message passing all supported.

PyBlox has its own built-in IDE that supports code highlighting, context-aware completion suggestions, always-visible documentation for the selected item (including functions/RPCs), a palette of project-specific custom blocks that can be dragged and dropped to paste code snippets, built-in example projects, an experimental NetsBlox to PyBlox project converter, and more. A screen capture of the work-in-progress PyBlox IDE is given in Figure 35.

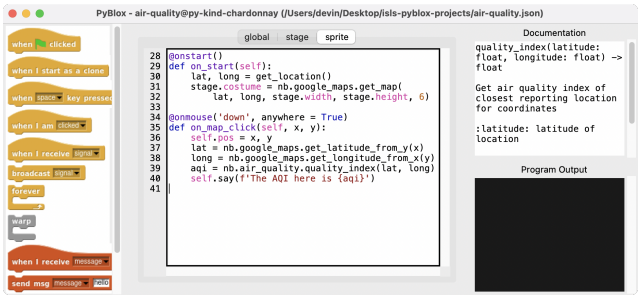


Figure 35: The PyBlox IDE showing an air quality index project functionally equivalent to the one shown in Figure 5 (all code in the sprite)

PyBlox mimics the concurrency model of NetsBlox scripts through various preprocessing steps applied to students' code. PyBlox also uses the same project breakdown of tabs for different sprites and the stage, using functions/methods as analogues for scripts and custom blocks, approximating NetsBlox-style event-based computing with function decorators, and providing access to all NetsBlox RPCs and message passing. Additionally, these Python constructs all have similar interfaces to their block-based counterparts; examples can be seen in Figure 36.

```
@nb.on_message('location')
def onloc(self, latitude, longitude, heading, altitude):
    self.pos = (
        nb.google_maps.get_x_from_longitude(longitude),
        nb.google_maps.get_y_from_latitude(latitude),
    )
    self.heading = heading
    if self.drawing:
        self.add_point([latitude, longitude])
        nb.phone_iot.set_text(device, text,
            f'Distance Covered: {distance}m')
    nb.phone_iot.set_image(device, display, stage.get_image())

@nb.on_message('start')
def on_start(self):
    self.drawing = True

@nb.on_message('stop')
def on_stop(self):
    self.drawing = False
```

Figure 36: Example PyBlox code. For reference, this is equivalent to the block-based code from Figure 22

Supporting the transition to text programming

However, regardless of the similarity between NetsBlox and PyBlox, the change of language imposes some overhead on students. The goal of PyBlox is to minimize this learning curve by providing not only similar features, but also tools that help students find the features that they need. For instance, the default palette of blocks in PyBlox contains examples for all of the existing function decorators, which are shown in the block-based form that students are already familiar with. Additionally, the contents of the blocks palette is customizable and saved in the project file. This allows instructors to hand-pick the set of blocks they think students would need for a given project and distribute them to students as a starter project. Further, PyBlox supports a mechanism to have blocks be automatically pulled from an online repository, meaning instructors could edit the custom blocks palette for all student projects without actually modifying the project files; this could be used to correct errors in custom blocks, or to add new "hint" blocks to help students who get stuck on some sub-task. Another helpful tool is a curated list of ubiquitous Python packages that can be toggled on/off to automatically import them into the project without the students' needing to write any code. These curated packages include customized descriptions and simple usage examples that are shown in the documentation panel when hovered over.

PyBlox's context-aware completion suggestions are invaluable in helping students find the right methods to call on objects, including both PyBlox-specific sprite/stage methods, as well as Python builtin functions such as `list.append`. The always-visible documentation panel ensures that students have effortless access to information about any function they are trying to use, including PyBlox-specific utilities like decorators and RPCs, or even their own custom methods/functions, as the documentation is extracted in real time from normal Python docstrings. The documentation panel is similar to the "help" menu for blocks in NetsBlox, but it does not need to be explicitly opened and updates automatically with the text cursor. This makes its existence and usage more obvious, meaning students are more likely to use it. Ultimately, we hope that listing the methods/fields on objects and presenting documentation for each entry (common features in modern IDEs) will encourage students to be more comfortable reading API documentation early on, which is an important practice in real-world programming.

Addressing differences in error-handling

However there are still some notable differences between NetsBlox and PyBlox. One big difference is with error handling; for instance, in NetsBlox, any error from an RPC causes the result to simply be the error message as a string. This string is also stored in a special "error" variable (successful RPC invocations clear the error back to the empty string). This has the unfortunate effect of allowing errors to propagate through a student's program. For instance, if the return value is not used or if the successful return type is a string (like the error return type), then there is no chance of discovering the error without explicitly checking the error block after every RPC invocation; this is tedious, and students rarely include error-checking logic in their programs. However, this lack of hard errors is consistent with NetsBlox's parent language, Snap!, which largely attempts to avoid hard errors when possible, even in some cases where it arguably should not. (For instance, indexing out of bounds or using invalid values for indices, such as `2.3` or `hello` simply returns the empty string, ""). In contrast, Python and virtually all textual languages are comparatively much more strict, and give hard errors (i.e., exceptions) for many of these cases. Because of these issues with propagating errors in NetsBlox and the motivation of giving students a realistic introduction to textual languages, it was decided that PyBlox should instead throw exceptions for RPC errors;

thus, students can rest assured that RPC return values are not error message strings, and are indeed proper RPC return values. Python-style exception handling code can be used to catch these errors, but there is an opt-in feature to have individual RPC invocations act just like the NetsBlox version and instead return an error message string and set a special error state accessible via `get_error()`. This feature is used by the experimental NetsBlox to PyBlox project converter to ensure identical error handling. A consequence of this decision to throw exceptions by default is that some code that students would write in blocks (which sometimes fails) will halt that script (but notably not the entire project) if error-handling code is not written; however, the exception (which is shown in the "Program Output" panel of the IDE) includes the exact line number of the error, which, coupled with the built-in documentation panel in the PyBlox IDE should make it easier to identify the problem with the provided RPC inputs and correct the issue (or add error handling code to allow failures).

Addressing differences in the execution model

Another difference between NetsBlox and PyBlox is in the execution model. NetsBlox projects, like Snap*!* projects, are *lively,* meaning the project is always running and modifications to the code take effect immediately, even in a running script. This is a very unorthodox feature to have in text-based programming environments, and the results of modifying a script during its execution are dubious at best for any non-trivial logic. For example, variables from the old code could be used with logic from the new code or vice versa, resulting in neither functioning correctly until a fresh execution of only the new code is performed. In light of these issues, PyBlox projects are run like ordinary Python code, with modifications of the program during execution not affecting the running program. Another runtime difference is in the concurrency model. NetsBlox and Snap*!* are interpreted languages, with very rigid, well-defined points at which context switching can occur. This provides very strong guarantees about the sequencing of instructions among various scripts. However, this is yet another feature that is unorthodox in textual languages, in which concurrent instructions are arbitrarily sequenced, even in languages like Python where concurrent scripts cannot truly execute simultaneously due to the global interpreter lock. PyBlox takes a best-effort approach, adding thread yield points at locations where NetsBlox/Snap*!* *would* have performed a context switch. The result roughly imitates the NetsBlox concurrency model without imposing too much execution overhead. It *would* be possible to perfectly imitate the block-based concurrency model if pausing/resuming threads were supported, but this is not possible in native Python. A related project called Pytch uses the Skulpt library to emulate Python from javascript running in the browser. This emulation allows for higher-level features like pausing/resuming threads, meaning that Pytch *could* emulate the NetsBlox/Snap*!* concurrency model more accurately [34]. (On the other hand, the authors also mention that violations of some of their timing assumptions can lead to time desynchronization between real time and simulator time, as seen by, e.g, the "wait secs" block). Our response to this particular concurrency model issue reflects one of PyBlox's larger hypotheses: that students want to switch from blocks to Python not because of Python itself, but because they see it as "real" programming. Because of this, in general, the design of PyBlox will make a best-effort approach to provide the same environment and utilities as NetsBlox, but it will not sacrifice vital, real-world Python features such as access to native libraries. Further it will use only abstractions that can be built up from native features such as OS threads, which inherently lack some sequencing controls. The result is that the vast majority of student programs will run virtually identically to the block-based equivalent, all while having realistic, native-like behaviors that will remain applicable in real-world Python development.

With all of these affordances, PyBlox serves as a tool to allow students to "graduate" from NetsBlox's block-based environment to Python, while still retaining much of their existing perspectives on project structure, concurrency models, and advanced CS topics that were introduced in NetsBlox. Further, because PyBlox features are largely isomorphic to NetsBlox, instructors can even (to some extent) recycle existing NetsBlox activities and reintroduce them in PyBlox with incrementally-added complexity (e.g., reimplementing and then extending old projects with new features).

PyBlox is still in active development, but it is available in its current, experimental state as a Python package called `netsblox`. We are in the process of designing studies to investigate PyBlox's effectiveness in supporting knowledge transfer from NetsBlox to PyBlox and Python.

11. Use Case: Natural Language Processing and Literary Analysis

We have shown how the NetsBlox platform can be used to eliminate superficial or "accidental" barriers to advanced CS topics, enabling novice programmers to engage conceptually with powerful ideas in computing. But not all programming novices are youths. Another important ongoing example of using NetsBlox in interdisciplinary work involves a project in the Digital Humanities [1]. Here, we describe how NetsBlox has leveraged to introduce powerful ideas and to mediate between exploration in a visual environment and scaled-up implementation in a Python notebook.

Working with faculty collaborators in the Vanderbilt Libraries and the English Department, we are using NetsBlox to support a two-way exchange of disciplinary practices, integrating (a) Computer Science understandings of the models used in AI and Natural Language Processing (NLP) and the computational techniques used in integrating them into analytic pipelines, with (b) Literary and Cultural Studies understandings of phenomena around shifts in language and social constructs as reflected in late 18th and early 19th century British periodical literature. In this article, we focus on the first aspect—the way that NetsBlox has enabled

challenging CS concepts in AI and NLP to be grasped and used creatively by undergraduates and professors in the Humanities.

For the past three years, we have been working with undergraduates from both computer science and the humanities to explore topics relevant to our English department colleague's research on analysing 19th-century British periodicals.. To work effectively with a corpus of over 4 million articles, we have been running exploratory analyses in a collaborative cluster computing environment, using PySpark notebooks on Amazon EMR and, more recently, Databricks's Lakehouse platform. These are powerful computing environments, but in order for all of our participants to be able to bring their insights bear and contribute actively to the collaborative work, it is important that they all have a fundamental understanding of the kinds of computational operations they can execute.

NetsBlox has enabled us to apply many of our NLP tools and concepts in a visual and interactive setting before bringing them to the PySpark notebook environment. While we have in fact connected NetsBlox to the full British Periodicals repository, we have found that after using the visual environment to engage playfully and iteratively with texts, tools, and techniques on a smaller corpus, students have been able to make the leap to notebooks, which have their own advantages, especially when operating at the scale of the entire repository.

Here, we provide an example of how NetsBlox Services have enabled us to provide a rich introduction to the strengths and limitations of AI tools: in this case, Sentiment Analysis models. Under its Language category, NetsBlox offers several Services, one of which provides an interface to ParallelDots, a commercial tool for NLP including sentiment analysis. Figure 37 shows the RPCs available in this Service, each of which takes a text argument (with "getSimilarity" taking two). While such tools will eventually be incorporated into pipelines that may involve a longer sequence of operations that select articles from the corpus and segment them into chunks (e.g., sentences or groups of sentences) to analyze, our initial explorations aimed to promote understanding of these tools' independent operation, so that they do not appear to students as 'black boxes.'

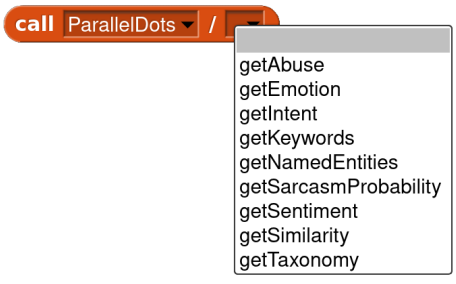


Figure 37: The RPCs available in the ParallelDots Service

Beyond the technical aspect of understanding return values and how to process them in a chain of actions, our work with NLP tools in NetsBlox involves more playful and collaborative forms of exploration to understand the *judgments* made by the model in question. For instance, with Sentiment Analysis, we have asked students to extend the Chat application described in Figure 9 to run sentiment analysis on the chat entries and color the text of the on-screen printout accordingly. This integrates the technical skill of operating on the model's judgments (and the format of the return) with an environment that allows the classroom group to experiment collaboratively and iteratively to make sense of how the model "thinks," how it can be fooled, and more generally, what its limitations are.

In our latest implementation of this activity, students made discoveries about the length of text that seemed to work best with the model; about how small changes (even in punctuation) could have significant impact on the model's judgment; and about how the model worked on informal chat-like entries as compared with passages they copy-pasted from articles in the British Periodicals corpus. These findings led students to study the documentation of the services, to learn about the training-sets that were used for them, and to reason about how any effort to do sentiment analysis on the British Periodicals corpus might need to involve domain-adaptation and/or other fine-tuning steps.

Turning from the tools of AI and NLP to the computational techniques involved in using them in practice, it was also important for students to understand and be able to mentally trace operations involving higher-order functions such as *map*. Here, it has been very powerful to have a visual environment where the relevant blocks can be assembled, hooked up to smaller corpus of sample articles, clicked to run, tested, and re-assembled. Once students created, tested, discussed, and refined block stacks that ran simple pipelines in NetsBlox, we found that they were in a much stronger position to interpret and construct text-based operations within PySpark notebooks.

12. Evaluation

NetsBlox was created to explore the core premise that a block-based construction environment could be developed that would make advanced distributed computing concepts accessible to younger learners. As such, the design and development work described in this article provides the *foundation* for future research on effectiveness. Nevertheless, we have already begun to conduct evaluation studies to assess whether NetsBlox is successful in realizing its design goals, and we have begun to use NetsBlox in empirical studies that explore different conditions and aspects of student learning.

In particular, we have conducted several small-scale evaluation studies of NetsBlox through summer camps and in after-school settings. Since we cannot assume prior programming knowledge, the first two days of a week-long camp usually focus on introductory programming before tackling the more advanced topics NetsBlox was designed to teach. These studies have shown both statistically significant learning gains and increased student interest and

engagement. Broll et al. report results from two summer camps that demonstrated between 15 and 20 percentage point gains in both CT and networking knowledge, using a pre- and post-test [10]. Four summer camps with 62 students total were conducted in 2018 and 2019 focusing on robotics and cybersecurity using physical robots [26, 54]. Significant learning gains were achieved in both CT and cybersecurity. A quote from a participating high school teacher illustrates the level of student engagement: *"I did not see them on cell phones, they were engaged with programming their robot."*

Feedback after a professional development workshop on 'Distributed Computing using NetsBlox' revealed teachers' ease with using RPCs and message passing blocks; their excitement about how these features could expand students' projects to include various data sources from the internet; and their interest in using NetsBlox in various ways in their schools—as part of teaching CS topics such as networks in AP CS Principles or in after-school camps [17].

In a majority-female, virtual high school camp on 'Climate Change & Computing,' students examined issues of climate change through working with real climate datasets, using multi-dimensional data structures, coding data visualizations, and engaging in data analyses. End-of-camp surveys probing students' perceptions of computing suggested the positive shift fostered by this experience. Some students expressed a-ha's that computer science "is more than making a character move and follow directions" and that it instead can be a "useful tool" to pull in a variety of real data to analyze and visualize and "learn and discover" real phenomena [18].

For several years now, NetsBlox has also been used to introduce first-year college students to programming during the first two weeks of an introductory programming course at Vanderbilt. The course teaches programming with MATLAB to non-CS engineering students. Anonymous surveys indicate that students with previous programming experience would rather not spend time with block-based programming, but most students appreciate the gentle introduction before switching to the main text-based language of the course. Finally, as described in section 11, NetsBlox has enabled a heterogeneous group of undergraduates of all levels and majors to engage in cutting-edge analyses of large historical literary corpora.

These studies suggest that NetsBlox is highly usable and that novices can engage with key concepts in distributed computing through its block-based implementation of RPCs and message passing. The design-based research of these studies has also produced some principles that guide ongoing implementations with NetsBlox. The curricula used in these studies are all project-based. Many times we present a starter project, e.g., a current weather app or a chat program, and then let students work on enhancing it any way they like. For example, one student team in one of the camps added their own encryption algorithm to the chat project, so that they could keep their conversation private. Most camps conclude with an individual or team project of the students' own choosing. Innovative examples include various multiplayer games such as a "Tron" clone, an interactive map interface for learning about country demographics, and a running route planner on top of Google Maps.

NetsBlox's support for collaboration has also enabled multiple empirical studies comparing approaches to collaborative programming. Zacharia et al. compared students working in driver-navigator or driver-driver pairs, showing that the driver-driver configuration did not have the perceived imbalance in student agency that driver-navigator did [55]. These results align with a 2020 study by Tsan, et al. on pair programming in 4th and 5th grade, comparing programming on one computer versus students both acting as drivers on their own devices [47]. Student interviews suggested that the one-computer condition helped them communicate more with their partner, and the two-computer condition was preferred, but that students struggled to coordinate the programming with their partner.

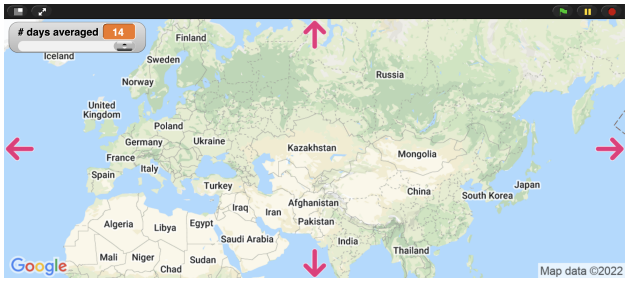
Lytle et al. compared three types of driver-driver NetsBlox collaboration in a middle school summer camp [29]. In this study, pairs of students worked on a series of four game-themed programming projects using different collaboration styles in NetsBlox: separate, together, and puzzle. "Separate" involved students programming two separate NetsBlox Roles to complete a pong game, with one student programming the Left Paddle role and the other taking the Right—with no collaborative editing across Roles. "Together" involved two students in the same Role, with synchronous editing to create a single paddle for Brick Breaker. "Puzzle" involved collaborative programming of a Basket sprite to collect falling fruit—but with the blocks partitioned so that each partner could only access half of them, requiring partners to talk to coordinate efforts. In a fourth game project, 16 of 24 pairs chose Puzzle while 8 chose Together-style collaboration, with many citing that working with complementary blocks was more fun and interesting. An overwhelming majority expressed interest in collaborative programming in the future, with 27 preferring Puzzle-style, 17 preferring Together-style, and only 4 preferring to work Separately on future projects. These statistics demonstrate that students in middle grades 6-8 (aged 11-14) appreciate collaboration while learning to program!

Empirical studies with NetsBlox in the future can explore a ever-widening range of topics that are of interest to Computing and Computer Science Education. An important part of the strength of the platform is its ability to present a range of distributed computing ideas and applications in a coherent and unified way. We look forward to longer-term studies that will enable us to document how students draw conceptual connections between the principles of designing distributed systems that enable users to interact with other humans, robots, IoT sensors, web-based data sources and services, and more. Through this line of work, we will be able to test and refine our conjectures about how distributed computing can serve as an entry point to big ideas in computer science, attracting a broader population of learners to engage with these concepts in personally meaningful ways.

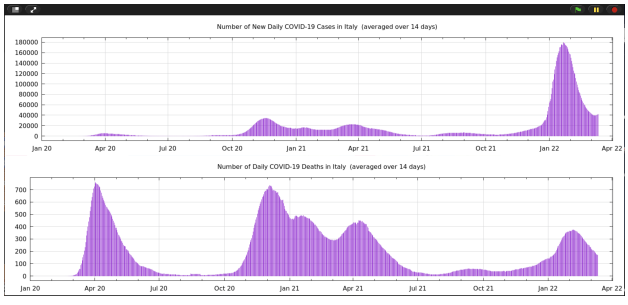
13. Conclusions

There is a widespread perception among high school-aged students that block-based environments are toys and not real programming languages [13, 51]. Our counter-argument in this article has been that the most limiting factor in introductory programming environments (whether text- or block-based) is being closed—"walled off" and disconnected from data, web services, and other applications. There is only so much one can do in a closed environment, and only so much relevance that a closed environment can achieve, when contrasted with the typical teenager's phone or laptop where the power of the entire internet is one click away.

NetsBlox demonstrates that introductory programming experiences do not have to be limited in this way. Leveraging modern web technologies and the affordances of block-based programming environments can enable novice programmers to create personally-meaningful projects that solve real problems or otherwise *matter* to them—making programming more relevant, more motivating, and more interesting. Projects such as distributed multi-player games, a shared whiteboard, and interactive global maps with superimposed climate data place powerful, creative possibilities in young learners' hands. Being able to utilize gadgets—phones, voice assistants, or robots—makes the experience all the more compelling. Both teachers and students from middle grades up can successfully create such programs using NetsBlox. Furthermore, such projects can democratize access to learning important modern computing concepts, such as distributed computing, cybersecurity, and computer networking. Until now, these topics have only been taught to computing undergraduates, despite their importance to computer literacy for everyone.



(a)



(b)

Figure 38: Clicking on an interactive map of the world (a) shows up-to-date COVID data in the selected country (b).

Let us summarize what is possible, once we remove the walls around a block-based programming environment:

- Student programs can access the wealth of information and services available on the internet. This makes it possible to create all kinds of STEAM-related projects, sparking the interest of students who may not be attracted to traditional approaches to computing. For example, Figure 38 shows a project visualizing up-to-date pandemic information anywhere in the world.
- Being able to create programs that can communicate with each other opens up a world of online multi-player games and social apps for students to create.
- A novel approach to robot and device programming becomes possible. This enables collaborative robotics, remote control of games and robots with mobile devices, voice assistant integration, and engaging, hands-on approaches to teaching cybersecurity.
- Novel forms of collaboration, including truly remote teamwork and sharing of in-process work can be supported seamlessly. In the age of COVID-19, this has become a crucial requirement, but the advantages of flexible collaboration will endure beyond the pandemic.
- Connections to other powerful environments for expressing computational logic, such as Python and PySpark notebooks, can ease the transfer of the core ideas learned in NetsBlox to settings they encounter in the future.

As we have shown, a lot of added functionality becomes available once programs have access to the internet, in an environment where connectivity and distributed computing are treated as 'first-class' design elements. The most important consideration is to keep the abstractions that provide this access simple and intuitive. When a new extension is provided to the typical block-based environment, it comes with many new blocks. This makes it difficult both to *learn* the related concepts and even to *find* the blocks, especially if multiple extensions are used. Instead, a more general mechanism should be provided. NetsBlox added just two new abstractions and introduced just three new blocks (*call*, *send*, *when I receive*) to those typical of block-based environments, in order to provide the wide array of new capabilities described in this article. Furthermore, these two new abstractions are similar to ones that many students are already familiar with: RPCs are like custom blocks, and messages are similar to events. This makes them intuitive and easy to learn and use. Furthermore, the analogies between these concepts are powerful tools for reasoning about key ideas in distributed computing. This makes NetsBlox a conceptual tool that extends the promise of block-based environments to offer a *restructuration* [52] of introductory computer science.

Another important consideration for any learning environment is to keep the environment extensible, even by the

users themselves. Adding Services, including support for new devices, should not require new custom blocks or any changes to the client code or the interface. Moreover, in NetsBlox, users themselves can add their own online data Services for everyone to use without leaving the environment.

Once we show students the wide variety of advanced projects and technologies such an environment enables with just a few blocks of code, they quickly reconsider the misconception that block-based programming is just for little kids. As two students said last summer: "It's really cool to see real world experience and real world data and real world things," and these projects help "a lot more people think [block-based programming] was really cool." A teacher added: "With the virtual delivery of my course, allowing students to collaborate in real time on a project, and understand HOW the collaboration works is a great learning experience...."

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 1835874, the National Security Agency (H98230-18-D-0010) and the Computational Thinking and Learning Initiative at Vanderbilt University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

The authors are grateful to Jens Mönig, Brian Harvey and Dan Garcia for the wonderful Snap*!* environment without which NetsBlox would not exist. A number of people have made important contributions to NetsBlox. We are grateful to Hamid Zare, Miklós Maróti, Péter Völgyesi, János Sallai and Cliff Anderson. Many undergraduate and high school students contributed by either adding new services to or finding bugs in NetsBlox.

References

[1] Anderson, C.B., Brady, C.E., Broll, B., Ramey, L.T., 2020. Human-centered computing for humanists: Case studies from the computational thinking and learning initiative at vanderbilt university., in: DH 2020.

[2] Assunção, L., Osório, A.L., 2006. Teaching web services using. NET platform, in: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education, pp. 339–339.

[3] Bennett, B., 2018. Accurate Distance Calculation using GPS while Performing Low Speed Activity. Master's thesis. University of Oregon.

[4] BirdBrain Technologies, . Remote Robots. https://www.birdbraintechnologies.com/remote-robots/. Cited March 1, 2022.

[5] Birman, K.P., 2005. Consistency in distributed systems. Reliable Distributed Systems: Technologies, Web Services, and Applications , 375–390.

[6] Brady, C., Holbert, N., Soylu, F., Novak, M., Wilensky, U., 2015. Sandboxes for model-based inquiry. Journal of Science Education and Technology 24, 265–286.

[7] Brady, C., Stroup, W., Petrosino, A., Wilensky, U., 2018. Group-based simulation and modelling: Technology supports for social constructionism, in: Proceedings of Constructionism 2018, Vilnius University.

[8] Broll, B., Lédeczi, A., Stein, G., Jean, D., Brady, C., Grover, S., Catete, V., Barnes, T., 2021. Removing the walls around visual educational programming environments, in: 2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE. pp. 1–9.

[9] Broll, B., Lédeczi, A., Volgyesi, P., Sallai, J., Maroti, M., Carrillo, A., Weeden-Wright, S.L., Vanags, C., Swartz, J.D., Lu, M., 2017. A visual programming environment for learning distributed programming, in: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, ACM. pp. 81–86.

[10] Broll, B., Lédeczi, Á., Zare, H., Do, D.N., Sallai, J., Völgyesi, P., Maróti, M., Brown, L., Vanags, C., 2018. A visual programming environment for introducing distributed computing to secondary education. Journal of Parallel and Distributed Computing 118, 189–200.

[11] Collective, B.M., Shaw, D., 2012. Makey makey: improvising tangible and nature-based user interfaces, in: Proceedings of the sixth international conference on tangible, embedded and embodied interaction, pp. 367–370.

[12] Cooper, S., Dann, W., Pausch, R., 2000. Alice: a 3-d tool for introductory programming concepts. Journal of computing sciences in colleges 15, 107–116.

[13] DiSalvo, B., 2014. Graphical qualities of educational technology: Using drag-and-drop and text-based programs for introductory computer science. IEEE computer graphics and applications 34, 12–15.

[14] Franklin, D., Weintrop, D., Palmer, J., Coenraad, M., Cobian, M., Beck, K., Rasmussen, A., Krause, S., White, M., Anaya, M., Crenshaw, Z., 2020. Scratch encore: The design and pilot of a culturally-relevant intermediate scratch curriculum, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, Association for Computing Machinery, New York, NY, USA. p. 794–800. URL: https://doi.org/10.1145/3328778.3366912, doi:10.1145/3328778.3366912.

[15] Garber, S.J., 1999. Searching for good science: the cancellation of NASA's SETI Program. J Br Interplanet Soc 52, 3–12.

[16] Garcia, D., Harvey, B., Barnes, T., 2015. The beauty and joy of computing. ACM Inroads 6, 71–79.

[17] Grover, S., Cateté, V., Barnes, T., Hill, M., Ledeczi, A., Broll, B., 2020. First principles to design for online, synchronous high school cs teacher training and curriculum co-design, in: Koli Calling'20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research, pp. 1–5.

[18] Grover, S., Oster, J., Ledeczi, A., Broll, B., Deweese, M., 2022. Climate science, data science and distributed computing to build teen students' positive perceptions of cs, in: Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2, pp. 1101–1101.

[19] Grover, S., Pea, R., 2018. Computational thinking: A competency whose time has come. Computer science education: Perspectives on teaching and learning in school 19.

[20] Grover, S., Pea, R., Cooper, S., 2015. Designing for deeper learning in a blended computer science course for middle school students. Computer science education 25, 199–237.

[21] Grover, S., Pea, R., Cooper, S., 2016. Factors influencing computer science learning in middle school, in: Proceedings of the 47th ACM technical symposium on computing science education, pp. 552–557.

[22] Harvey, B., Garcia, D.D., Barnes, T., Titterton, N., Miller, O., Armendariz, D., McKinsey, J., Machardy, Z., Lemon, E., Morris, S., Paley, J., 2014. Snap! (build your own blocks), in: Proceedings of the 45th ACM Technical Symposium on Computer Science Education, ACM, New York, NY, USA. pp. 749–749. URL: http://doi.acm.org/10.1145/2538862.2539022, doi:10.1145/2538862.2539022.

[23] Kafai, Y.B., 2016. From computational thinking to computational participation in k–12 education. Communications of the ACM 59, 26–27.

[24] Kelly, A., Finch, L., Bolles, M., Shapiro, R.B., 2018. BlockyTalky: New programmable tools to enable students' learning networks.

International Journal of Child-Computer Interaction 18, 8 – 18. URL: http://www.sciencedirect.com/science/article/pii/S2212868918300394, doi:https://doi.org/10.1016/j.ijcci.2018.03.004.

[25] Korpela, E., Werthimer, D., Anderson, D., Cobb, J., Leboisky, M., 2001. SETI@home-massively distributed computing for SETI. Computing in Science Engineering 3, 78 –83.

[26] Lédeczi, Á., Metelko, M., Koutsoukos, X., Biswas, G., Maróti, M., Zare, H., Yett, B., Hutchins, N., Broll, B., Völgyesi, P., Smith, M.B., Darrah, T., 2019. Teaching Cybersecurity with Networked Robots, in: Proceedings of the 50th ACM Technical Symposium on Computer Science Education, ACM. pp. 885–891. doi:10.1145/3287324.3287450.

[27] Lesh, R., Hoover, M., Hole, B., Kelly, A., Post, T.R., 2000. Principles for developing thought-revealing activities for students and teachers, in: Research design in mathematics and science education. Lawrence Erlbaum Associates, Inc., pp. 591–646.

[28] Lim, B.B., Jong, C., Mahatanankoon, P., 2005. On integrating web services from the ground up into CS1/CS2, in: Proceedings of the 36th SIGCSE technical symposium on Computer science education, pp. 241–245.

[29] Lytle, N., Milliken, A., Cateté, V., Barnes, T., 2020. Investigating different assignment designs to promote collaboration in block-based environments, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, pp. 832–838.

[30] Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., Resnick, M., 2004. Scratch: A sneak preview, in: Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing, IEEE Computer Society, Washington, DC, USA. pp. 104–109. URL: http://dx.doi.org/10.1109/C5.2004.33, doi:10.1109/C5.2004.33.

[31] Maloney, J.H., Peppler, K., Kafai, Y., Resnick, M., Rusk, N., 2008. Programming by choice: urban youth learning programming with scratch, in: ACM SIGCSE Bulletin, ACM. pp. 367–371.

[32] Moroney, L., 2017. The firebase realtime database, in: The Definitive Guide to Firebase. Springer, pp. 51–71.

[33] NetsBlox, . NetsBlox website. https://netsblox.org. Cited March 1, 2022.

[34] North, B., Strong, G., . Pytch. https://www.pytch.org/doc/vm/developer/threading-model.html.

[35] Papert, S., 1980. Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc.

[36] Pokress, S.C., Veiga, J.J.D., 2013. MIT App Inventor: Enabling Personal Mobile Computing. arXiv:1310.2830.

[37] Price, T.W., Barnes, T., 2015. Comparing textual and block interfaces in a novice programming environment, in: Proceedings of the eleventh annual international conference on international computing education research, pp. 91–99.

[38] Repenning, A., 2000. Agentsheets®: An interactive simulation environment with end-user programmable agents. Interaction .

[39] Rodríguez, F.J., Price, K.M., Boyer, K.E., 2017. Exploring the pair programming process: Characteristics of effective collaboration, in: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, Association for Computing Machinery, New York, NY, USA. p. 507–512. URL: https://doi.org/10.1145/3017680.3017748, doi:10.1145/3017680.3017748.

[40] Schmidt, A., 2016. Increasing computer literacy with the bbc micro: bit. IEEE Pervasive Computing 15, 5–7.

[41] Shahin, M., Gonsalvez, C., Whittle, J., Chen, C., Li, L., Xia, X., 2022. How secondary school girls perceive computational thinking practices through collaborative programming with the micro: bit. Journal of Systems and Software 183, 111107.

[42] Smith, N., Sutcliffe, C., Sandvik, L., 2014. Code club: bringing programming to uk primary schools through scratch, in: Proceedings of the 45th ACM technical symposium on Computer science education, pp. 517–522.

[43] Solomon, C., Harvey, B., Kahn, K., Lieberman, H., Miller, M.L., Minsky, M., Papert, A., Silverman, B., 2020. History of logo. Proceedings of the ACM on Programming Languages 4, 1–66.

[44] Stein, G., Lédeczi, A., 2021. Enabling collaborative distance robotics education for novice programmers, in: 2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE. pp. 1–5.

[45] Stroup, W., Ares, N., Lesh, R., Hurford, A., 2007. Diversity by design: The what, why and how of generativity in next-generation classroom networks. .

[46] Stroup, W.M., Ares, N.M., Hurford, A.C., 2005. A dialectic analysis of generativity: Issues of network-supported design in mathematics and science. Mathematical Thinking and Learning 7, 181–206.

[47] Tsan, J., Vandenberg, J., Zakaria, Z., Wiggins, J.B., Webber, A.R., Bradbury, A., Lynch, C., Wiebe, E., Boyer, K.E., 2020. A comparison of two pair programming configurations for upper elementary students, Association for Computing Machinery, New York, NY, USA. p. 346–352. URL: https://doi.org/10.1145/3328778.3366941, doi:10.1145/3328778.3366941.

[48] Upton, E., Halfacree, G., 2012. Meet the raspberry Pi. John Wiley & Sons.

[49] Weintrop, D., Wilensky, U., 2013. Robobuilder: a computational thinking game, in: SIGCSE, p. 736.

[50] Weintrop, D., Wilensky, U., 2015. 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, pp. 199–208.

[51] Weintrop, D., Wilensky, U., 2017. Comparing block-based and text-based programming in high school computer science classrooms. ACM Transactions on Computing Education (TOCE) 18, 1–25.

[52] Wilensky, U., Papert, S., 2010. Restructurations: Reformulations of knowledge disciplines through new representational forms. Constructionism 17, 1–15.

[53] Witherspoon, E.B., Higashi, R.M., Schunn, C.D., Baehr, E.C., Shoop, R., 2017. Developing computational thinking through a virtual robotics programming curriculum. ACM Trans. Comput. Educ. 18. URL: https://doi.org/10.1145/3104982, doi:10.1145/3104982.

[54] Yett, B., Hutchins, N., Stein, G., Zare, H., Snyder, C., Biswas, G., Metelko, M., Lédeczi, Á., 2020. A hands-on cybersecurity curriculum using a robotics platform, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, pp. 1040–1046.

[55] Zacharia, Z., Boulden, D., Vandenberg, J., Tsan, J., Lynch, C., Wiebe, E., Boyer, K., 2019. Collaborative talk across two pair-programming configurations, in: A Wide Lens: Combining Embodied, Enactive, Extended, and Embedded Learning in Collaborative Settings, 13th International Conference on Computer Supported Collaborative Learning (CSCL) 2019.