UNLEASHING CREATIVITY WITH WIRELESS EMBEDDED PROGRAMMING FOR NEXT-GENERATION MAKERS

D. Jean, G. Stein, B. Broll, A. Ledeczi

Vanderbilt University (UNITED STATES)

Abstract

In the last two decades, we have witnessed the advent of the internet era, with nearly every application and tool we use on a daily basis providing some or all of its features by interacting with networks of communicating devices distributed across the world. These diverse services range from collaborative document editing, to online schooling, and even to industrial and municipal automation. However, despite the ubiquity of these technologies, distributed computing is typically overlooked in contemporary K-12 education, often due to complexity, cost, or a shortage of teachers who have experience in this field. One project which addresses this issue is NetsBlox, a block-based programming environment that prominently features high-level distributed computing abstractions. In this paper, we expand this work by introducing a new NetsBlox Virtual Machine that allows students to use NetsBlox to program stand-alone embedded devices and robots for a wide variety of engaging, internet-enabled makers projects.

Keywords: Education, distributed computing, visual programming, embedded programming, constructionism.

1 INTRODUCTION

In our rapidly evolving technological world, having a solid foundational knowledge of computing concepts is becoming increasingly vital for digital literacy. In particular, there has been an enormous increase in our reliance on internet-based services and applications. From the user-level applications of live digital classrooms, collaborative editing, and note taking tools, all the way down to the wireless embedded sensors that enable municipal utility automation and weather forecasting, distributed (networked) computing has revolutionized the efficiency and convenience of many facets of our daily lives. However, despite the ubiquity of these technologies, distributed computing is often overlooked in K-12 education, and the opportunity for students to gain hands-on experience building internet-powered projects is even more uncommon.

One project trying to counteract this is NetsBlox [1] [2], a block-based programming environment based on Snap*!* [3] (similar to Scratch [4]) which adds convenient distributed programming abstractions. NetsBlox simultaneously has a low enough floor to build a simple project visualizing historic CO2 concentrations from the online NOAA Antarctic ice core database with only a few blocks of code, as well as a high enough ceiling to build a fully-functional recreation of Google Maps and Street View with all features programmed by students using sufficiently abstracted blocks that access Google's APIs. In fact, NetsBlox has been so successful as an educational tool that it currently has a full, year-long high school curriculum approved for use in Tennessee and currently being piloted [5].

In this work, we take NetsBlox to the next level with a new NetsBlox Virtual Machine (VM). Traditionally, despite having full access to the internet, students' NetsBlox projects could only run in the browser on their laptop; however, with NetsBlox VM we can run students' same project code on practically any device, dramatically expanding the scope of creative projects students can build within the same NetsBlox programming language. Three new educational platforms are already powered by NetsBlox VM: 1) a smartphone app called PhoneIoT which allows students to program their own devices and access their phones' sensors remotely just like real-world industrial applications of the Internet of Things (IoT), 2) an embedded platform that allows students to create and program custom internet-connected embedded devices and robots similar to those used in popular Makers projects, and 3) a 3D virtual, collaborative robotics platform known as RoboScape Online that empowers students to create custom virtual worlds and populate them with programmable robots with a multitude of built-in virtual sensors. In this paper, we will overview the design of NetsBlox VM as it pertains to K-12 education, and explore the three specific applications we have listed, along with some results from a classroom study on RoboScape Online.

2 BACKGROUND

One of the biggest barriers to teaching advanced computer science concepts such as distributed computing and embedded programming is simply the complexity of coding said projects in traditional programming languages (e.g., C/C++). However, similar arguments have also been made about general introductory programming in education, for which we have seen dramatic improvements in approachability and accessibility through the application of visual programming techniques, the most common of which being block-based programming [6]. This was the approach taken by the Scratch project [4], which enables K-12 students to learn basic programming skills through the creation of graphical projects and games. However, although Scratch has a low floor, it also has a relatively low ceiling, at least when compared to general-purpose programming languages. One project that addressed this issue was Snap*!* [3], which took the model of Scratch and added additional powerful features such as lists as a first-class data type (i.e., students can now create lists on-the-fly and compose lists of lists, etc.), the ability to define custom blocks (functions) which support recursion, and the ability to use functions as values, which enables functional programming and higher-order functions. Thus, Snap*!* is able to provide the same low floor as Scratch, but also provide a very high ceiling by adding these "missing components" of advanced, but optional programming features.

The original concept for NetsBlox [1] [2] was to address one more missing component that Snap*!* lacked, namely the ability to access web-based resources and create distributed projects. To address this with minimal changes/complexity, only two new concepts were added: Remote Procedure Calls (RPCs), which allow students to access curated web services through simplified and abstracted interfaces, and message passing, which allows students to send and receive custom packets of data between projects over the internet. Figure 1 shows an example of using the "call" block to invoke RPCs and receive their result (in this case, the air quality index at a specific geographical location), as well as the blocks used for receiving and sending messages (e.g., to implement a distributed program or multi-player game). These advanced features provided by NetsBlox and Snap*!* make it possible to teach a wide range of topics spanning from introductory computing, to advanced computing (e.g., algorithms, data structures, and functional programming), and even to distributed computing and networking.

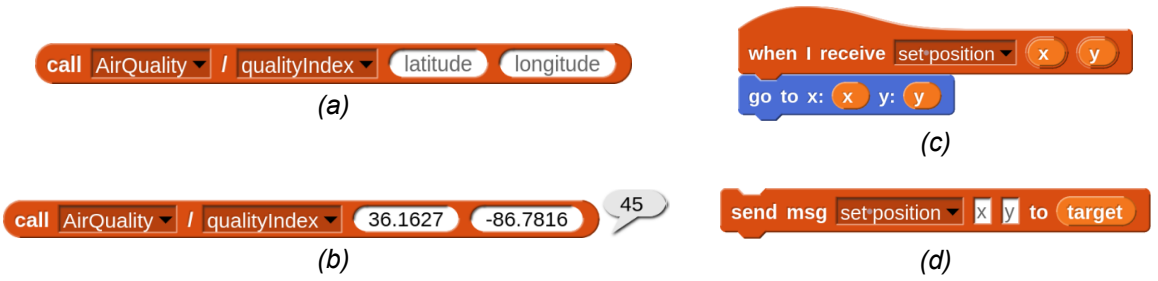


Figure 1. Examples of using NetsBlox RPCs (a) and (b), message receiving (c) and message sending (d).

However, despite all of this power, it is still the case that NetsBlox projects have been traditionally trapped running in the browser. Because of this, it is impossible to run NetsBlox (or Snap!) projects on other devices such as servers and especially embedded devices, which are by far the most limiting platform for any software. There are some projects which attempt to address this embedded programming limitation, such as the BBC micro:bit [7] and Snap4Arduino [8]; however, in providing access to the embedded programming world, these platforms lose the vast majority of the advanced programming features that tools such as Snap*!* have made possible. Indeed, these tools and others typically degrade to a point even below that of base Scratch in terms of available programming features. The goal, then, for NetsBlox VM is to address these same limitations through a different approach that allows us to retain all of the advanced features of NetsBlox (and Snap*!*) while still being able to target embedded devices and other platforms.

3 NETSBLOX VM

NetsBlox VM was designed with three primary requirements in mind: 1) it should support all of the same advanced programming concepts added by Snap*!* and all of the distributed computing features added by NetsBlox, 2) it should be able to run on any device, and 3) it should be extensible to allow for its application in a wide range of use cases with minimal effort. With the goal of running on embedded

devices, requirement 2 necessitated the use of a “systems” programming language (e.g., C/C++). Rust was selected for this project due to its emphasis on safety/reliability and ease of correctness testing while still being able to run on any platform; e.g., embedded microcontrollers, desktop applications, mobile applications, websites, servers, and so on. Extra considerations were made at this time to, for example, minimize the amount of memory needed for execution (vital for embedded platforms) and to ensure that any errors in student code would have access to a rich context of information to aid in student debugging. In fact, the error reporting system of NetsBlox VM is even more powerful than the original NetsBlox/Snap*!* error reporting systems, as we will see in the section on NetsBlox32.

However, this complete rebuilding of the entire NetsBlox/Snap*!* runtime was no trivial task, and creating such a system can be quite error-prone since behaviors must match exactly or else lead to student confusion. This was the intention behind design requirement 3, which would allow us to reuse this same reimplementation of the NetsBlox/Snap*!* runtime for future projects, thus guaranteeing identical program behavior while simultaneously greatly reducing the amount of work needed to create new educational tools with NetBlox code execution features. To achieve this, NetsBlox VM was equipped with a plugin-like system that allows tools to intercept existing or new/unknown blocks and give them special behaviors. For instance, the standard blocks for turning sprites left or right could be intercepted and instead used to control the movement of a physical robot. Another example might be intercepting the “play note” blocks and, rather than playing them through the speakers, append them to a live-transcribed piece of sheet music that is output as a rendered PDF (this is actually another real application of NetsBlox VM which is being developed by an undergraduate intern in our lab). With this level of extensibility in play, it is possible to create a wide range of applications which expand the reach of NetsBlox/Snap*!* into new domains of creative projects spanning the entirety of STEAM [9].

4 APPLICATIONS

In this section, we will explore three example applications, each running on a different category of device, which use NetsBlox VM to allow students to program said devices or simulations therein and ultimately create personally-meaningful educational projects. It should also be noted that each of these three applications uses the exact same NetsBlox VM code and only makes VM-related behavior changes via the previously-described plugin system.

4.1 PhoneIoT

The first platform we will discuss is PhoneIoT [10], which is an existing tool used in some NetsBlox curriculum on Internet of Things (IoT) topics [5]. PhoneIoT takes the form of a free and open-source mobile application for Android and iOS devices. Once launched, the app automatically connects to the NetsBlox server and makes itself accessible for remote interaction from student programs. To interact with devices remotely, students can open a NetsBlox project and use a collection of RPCs to send the phone requests such as getting the current values of sensors (e.g., gps location or air pressure) or to send the phone commands such as placing an interactive button at a certain location on the screen. In addition to these “polling” (on-demand) features, PhoneIoT also supports several “streaming” (asynchronous) access features. For instance, students can request to receive periodic updates from specific sensors on the device at specific intervals. These updates are sent from the Phone back to the student’s NetsBlox project via normal message passing. Thus, PhoneIoT allows students to use the same abstractions of RPCs and message passing to interact with real, remote IoT devices and use them in a number of educational projects (e.g., a physics experiment where students drop their phone onto a pillow while plotting the accelerometer value, or a project where students turn their phone into a remote controller for a game or other distributed application).

Originally, PhoneIoT supported only this remote access paradigm where students could use NetsBlox projects to interact with their device over the internet, but could not program their phone directly. Granted, this was the original novelty and purpose of the tool, as opposed to other projects such as MIT App Inventor [11] which has the opposite problem. However, with the creation of NetsBlox VM, it was possible to incorporate the ability to run NetsBlox projects directly on the device. To do this, the existing (unmodified) code of NetsBlox VM was used and a plugin was developed that intercepts all of the usual PhoneIoT-related RPCs and message passing and sends them directly to/from the phone, bypassing the internet entirely. That is to say, students need not even change their programs in order to access this local execution feature: the same student program that accesses the phone remotely can be loaded onto the phone (via QR code) in order to run locally. As an example, Figure 2 shows a (remote or local) PhoneIoT project that streams and plots live barometric pressure data on the phone.

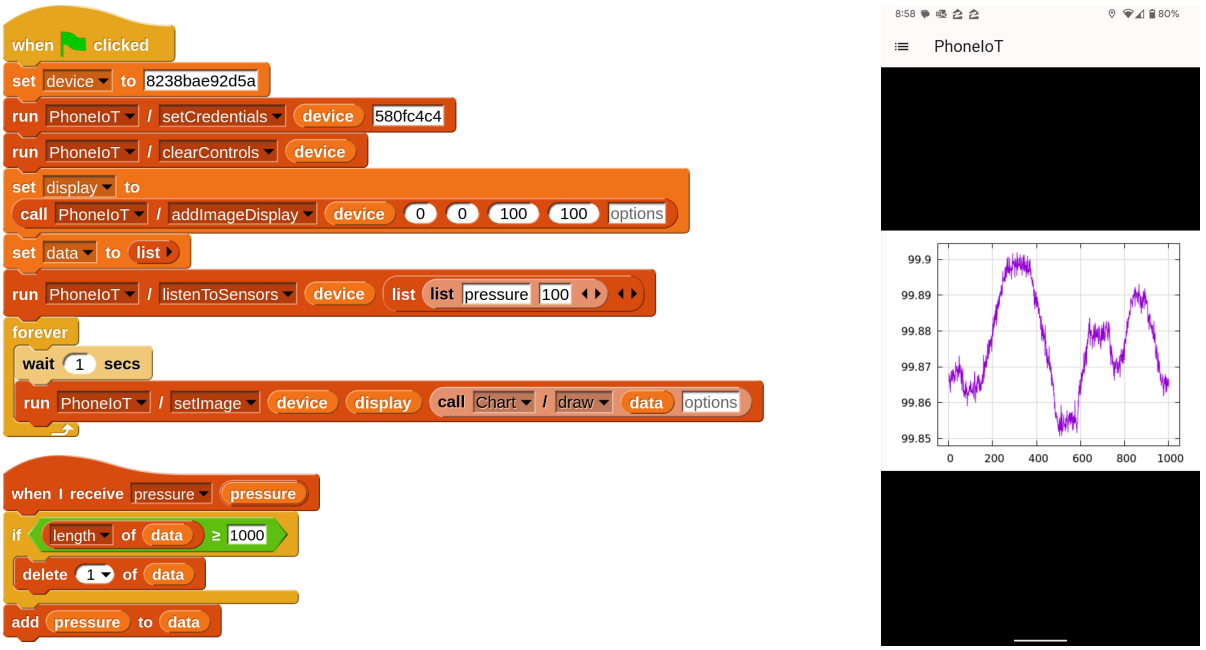


Figure 2. A PhoneIoT project that plots barometric pressure data (kPa) on the phone screen in real time. This same project can run remotely or directly on the device with no change of behavior.

Local execution also allows for significant speedups; whereas remote sensor access is artificially limited to 10Hz due to network throughput considerations in large classroom settings, local execution speed is unrestricted and can reach into the range of 300-400Hz. This enables the use of some advanced real-time sensing projects, such as an existing PhoneIoT project which translates “tap code” (similar to Morse code) into text by detecting vibrations from any resting surface via high-frequency accelerometer access. Overall, this goes to show how easy it is to integrate NetsBlox VM into an existing tool and expand the types of projects students are able to create within the same NetsBlox programming language.

4.2 NetsBlox32

In the previous section, we explored how the latest version of PhoneIoT was equipped with NetsBlox VM and gained the ability to provide real-time access to device sensors through local execution. This makes it possible to use PhoneIoT to teach some topics within the purview of embedded programming, but we can do better. Because NetsBlox VM is written in “no-std” Rust, it is fully capable of running directly on embedded microcontrollers. NetsBlox32 is specifically a tool which uses NetsBlox VM to run directly on ESP32 microcontrollers, and in particular is tested on an ESP32-S3 with 8 MB of RAM (though smaller RAM sizes would also be sufficient). Notably, even the complete development board costs only 17 USD, making its cost comparable to micro:bit.

One of the key design principles of NetsBlox32 was to provide a completely over-air experience and not require any physical connection to the device for any purpose. To begin programming a NetsBlox32 device, students need only open the NetsBlox editor with an extension hosted by the board; this extension provides a way to interact with the device through a pop-up “terminal” window shown in the NetsBlox editor. This includes the ability to upload the current program to the device, download its stored program, as well as control program execution with the familiar NetsBlox/Snap*!* controls to start/pause/stop execution. The terminal also contains a scrolling text display that shows the output of any “say” blocks (the NetsBlox32 equivalent of a print statement), as well as any messages from the system (e.g., error messages from student code).

When errors occur, a general description is output to the terminal, which is similar to the existing NetsBlox/Snap*!* error reporting system in terms of detail. However, NetsBlox32 goes much further and also produces a visual stack trace of the error including a snapshot of all relevant variables in scope at each call site. This visual stack trace is displayed as a collection of red error comments shown directly in a student’s project; Figure 3 shows an example of such an error being displayed. This additional information when errors are encountered is expected to greatly aid students in debugging their programs. When the error is resolved, students can dismiss any of the red error comments and the

entire collection will be deleted together. When not connected to the device, both output and errors are stored in cyclic buffers, which allows students to debug errors that occurred even when they were not connected to the device.

Figure 3. An example of an error produced by NetsBlox32 and its visualization in the NetsBlox editor.

Basic code execution aside, NetsBlox32 also supports a collection of external hardware peripherals that can be accessed by a special “syscall” (system call) block, which is conceptually the embedded analogy for the familiar RPC call block. Currently, these supported peripherals include: digital inputs/outputs, DC motors, ultrasonic distance sensors, temperature sensors, pressure sensors, light sensors, 3-axis accelerometers, and RGB LED matrix displays. This collection was carefully selected to enable a broad range of practical science projects (e.g., a wireless remote weather station measuring temperature and pressure over several weeks). Additionally, when possible, peripherals were selected which use the I2C bus protocol, which are quick and easy for students to connect and swap. Although built-in support for new types of peripherals must be added in Rust, the specific collection of peripherals on a NetsBlox32 device and their settings can be configured wirelessly through the board’s configuration page and requires no flashing process, unlike similar platforms. Figure 4 shows an example of a NetsBlox32 device equipped with several peripherals and a short autonomous driving program.

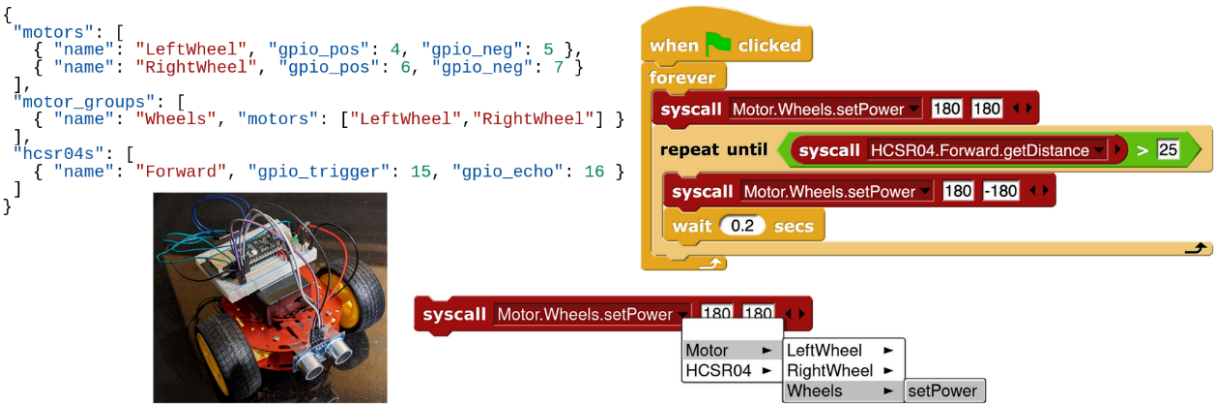


Figure 4. A NetsBlox32 robot (bottom left) configured with two motors and an ultrasonic distance sensor (top left). A short program (top right) controls the robot to drive forward but avoid obstacles by turning right.

4.3 RoboScape Online

While NetsBlox32 provides a demonstration of how NetsBlox VM may be used to control a mobile robot, it represents a significant difference from previous NetsBlox robotics activities. Existing robotics support for NetsBlox is provided through the RoboScape service [12], where robots are exposed as distributed components belonging to a web service where commands are sent through RPCs. This system allows students to control robots through the same abstractions used with other web services, making robotics a more natural extension of the initial NetsBlox lessons for students still learning the basics of computer science.

However, the physical robots traditionally used with RoboScape have many barriers to entry restricting their use in classrooms. The costs associated with purchasing and maintaining physical robots, among other such barriers, have motivated the creation of a networked simulation environment for educational robotics in NetsBlox, known as RoboScape Online. While earlier versions of RoboScape Online had scenarios created in JavaScript, C#, or another editor, the newest iteration of the platform has integrated NetsBlox VM to allow all scenarios used in its curriculum to be implemented entirely in NetsBlox itself. A suite of new services built through the IoTScape service [14] has been created to allow elements of the simulation to be created, modified, manipulated, and removed all through NetsBlox RPCs. Now, each RoboScape Online scenario is simply the execution of a shared NetsBlox project running in a NetsBlox VM instance on the simulation server. The introductory project for creating RoboScape online scenarios and the resulting simulation state is shown in Figure 4. While creating a new scenario for the simulation, the user is able to work and test their code in the browser NetsBlox environment, but in actual use the project will be executed in NetsBlox VM. This not only lowers the floor of creating a scenario to skills learned through normal NetsBlox use, but also simplifies the process for students to distribute their creations.

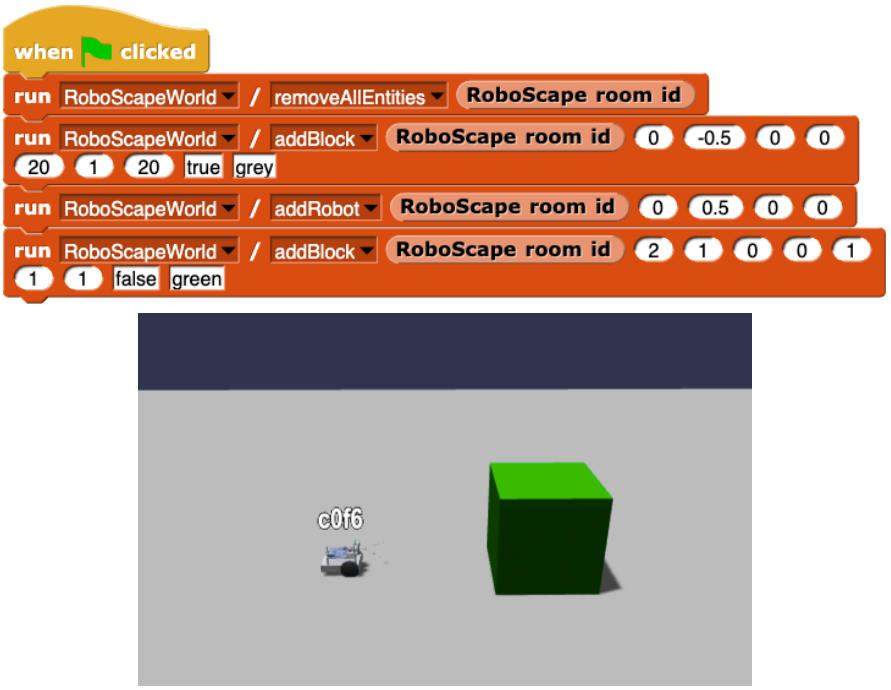
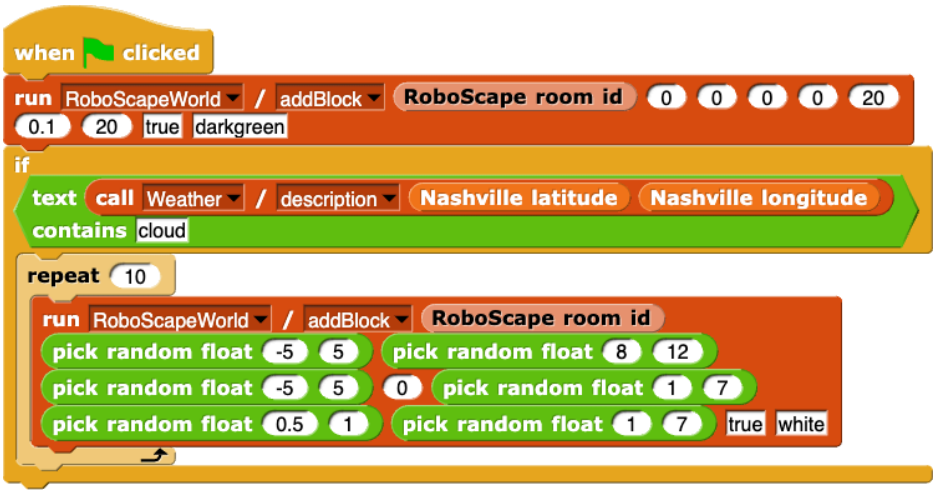


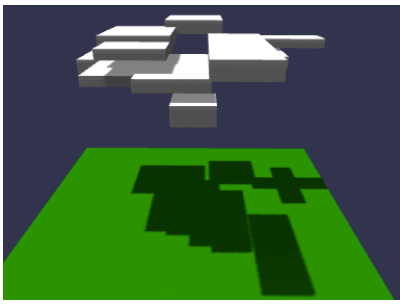
Figure 4. An example of a simple RoboScape Online program

Furthermore, the integration with NetsBlox VM has allowed for many components of the simulation which would otherwise require custom implementations for code reuse between different scenarios, such as a timer or a state machine, to no longer need implementation in the server code. Instead, the built in capabilities of NetsBlox no longer require additional work to expose these components to the user. Existing NetsBlox libraries can be used in these projects, and the same web services are available to projects powering simulation scenarios. Figure 5 shows an example scenario creating a different environment based on real-world weather conditions in Nashville obtained through the “Weather” web service. Message passing may also be used, communicating with other NetsBlox projects to widen the walls of potential projects even further.



(a) Scenario code

(b) With clear weather



(c) With cloudy weather

Figure 5. An example of a RoboScape Online scenario program using a web service.

5 METHODOLOGY

Use of NetsBlox VM in the RoboScape Online application has been evaluated through a classroom case study. During the Fall 2023 semester, a class using RoboScape Online was implemented at the School for Science and Math at Vanderbilt (SSMV). The SSMV program allows high school age students in the Metropolitan Nashville Public Schools district to attend special STEM courses, with a focus on participating in research, on the Vanderbilt campus one day per week instead of their usual classes. Over eight two-hour sessions, the students were introduced to NetsBlox, given a series of activities on autonomous robotics, and finally introduced to creating their own environments for the simulator and running them on the VM through the cloud-based RoboScape Online server instances. While only the final three lessons directly demonstrated the use of student-written code running in NetsBlox VM, all environments used in all simulated robotics activities were NetsBlox projects run in the VM. The class consisted of 26 high school sophomores. Although the ages of the students were similar, their other demographics make up a diverse group: 10 students identified as male, 15 identified as female, and 1 did not wish to identify their gender. 10 students identified as non-Hispanic white, 8 as black, 2 as Asian/Pacific Islander, 2 as South Asian/Indian, 1 as Hispanic/Latino, and 3 as bi-/multi-racial.

Evaluation of the program focused on student opinion surveys and interviews. A range of questions adapted from the Electronic Textiles Student Survey [13] modified for a focus on robotics and computational creativity were used to assess student attitudes before and after the course. These surveys were given online through the REDCap platform, and all opinion questions were scored on a six-point Likert scale. In interviews, students were asked open-ended questions about their favorite activities, likes and dislikes regarding the environment, feelings about creating their own environments, and other similar topics.

The use of NetsBlox VM in PhoneIoT and NetsBlox32 are currently awaiting evaluation in classroom settings.

6 RESULTS

The entire class of 26 students using RoboScape Online provided pre-survey responses, while only 21 completed the post-survey. An imputation method was used to allow for analysis of the responses received, using the students' pre-survey data to fill in their missing post-survey data, which would only reduce the probability of rejecting the null hypothesis and the effect size. Although the surveys were unpaired, demographic data of the missing students was sufficient to give only 24 possible combinations of missing data. The values presented in this work are based on the worst-case of these combinations. The statistically significant survey responses are listed in Table 1.

Table 1. Significant Survey Results.

Question	*Pre-survey average*	*Post-survey average*	*Effect size (Cohen's d)*	*p-value*
I think I am very good at coming up with new ideas when working on projects.	4.640	5.077	> 0.46	< 0.03
Thinking like a computer scientist will help me do well in my classes.	4.000	4.538	> 0.61	< 0.03
I can be creative in computer science.	4.308	5.040	> 0.78	< 0.01
I can express myself in computer science.	3.769	4.600	> 0.76	< 0.02
I can explain how robots make decisions.	3.500	4.385	> 0.74	< 0.01

7 CONCLUSIONS

In this paper, we have provided an overview of the NetsBlox VM system, which is able to execute student-written block-based programs on any platform while providing extensibility for its use in a number of diverse educational tools/platforms. Specifically, we have explored the merits of NetsBlox VM being used in three educational tools: PhoneIoT, NetsBlox32, and RoboScape Online. In each case, we saw that students are able to create relatively simple programs in the same, familiar NetsBlox block-based language in order to target different aspects of STEAM learning; e.g., IoT and app development through PhoneIoT, real-time sensing and robotics through NetsBlox32, and 3D scenario creation and simulation management through RoboScape Online.

Although evaluation of PhoneIoT and NetsBlox32 has not yet been performed, we have explored some results from using RoboScape Online in a classroom study with 26 students. All simulation scenarios used in the study were implemented as NetsBlox projects running in NetsBlox VM to construct and administrate the scenario. Additionally, the final three sessions in the study gave students the ability to create their own custom scenarios (as NetsBlox projects) which could then be published (shared) and run in NetsBlox VM just like the other "official" scenarios used earlier in the study. Additionally, from the pre-/post-survey results, it was found that the use of RoboScape Online led to statistically significant increases in several positive attitudinal axes, including but not limited to, creativity when working on projects, belief that computational thinking will help students in their classes, and understanding of robot decision methods.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation (#1949472). REDCap was made available through grant UL1 TR000445 from NCATS/NIH.

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.

REFERENCES

[1] B. Broll, A. Ledeczi, P. Volgyesi, J. Sallai, M. Maroti, A. Carrillo, S. Weeden-Wright, C. Vanags, J. Swartz, and M. Lu, “A Visual Programming Environment for Learning Distributed Programming,” *Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education*, pp. 81–86, 2017.

[2] B. Broll, A. Ledeczi, P. Volgyesi, J. Sallai, M. Maroti, and C. Vanags, “Introducing Parallel and Distributed Computing to K12,” *IEEE International Parallel and Distributed Processing Symposium Workshops*, pp. 323–330, 2017.

[3] B. Romagosa, “The Snap*!* Programming System,” *Encyclopedia of Education and Information Technologies*, pp. 1–10, 2019.

[4] J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman, and M. Resnick, “Scratch: A Sneak Preview,” *Proceedings of the Second International COnference on Creating, Connecting, and Collaborating through Computing*, pp. 104–109, 2004.

[5] G. Stein, I. Gransbury, D, Jean, M. Hill, V. Catetem S, Grover, T, Barnes, B. Broll, and A. Ledeczi, “Engaging Female High School Students in the Frontiers of Computing,” *ASEE Annual Conference and Exposition*, pp. 1–14, 2022.

[6] D. Weintrop, “Block-Based Programming in Computer Science Education,” *Communications of the ACM*, vol. 62, no. 8, pp. 22–25, 2019.

[7] T. Ball, J. Protzenko, J. Bishop, M. Moskal, P. Halleux, M. Braun, S. Hodges, and C. Riley, “Microsoft Touch Develop and the BBC micro:bit,” *ICSE 2016 Companion*, pp. 1–4, 2016.

[8] R. Bernat and G. Jaon, “Snap4Arduino: Arduino goes lambda!” *web*, 2023. https://snap4arduino.rocks/

[9] M. Land, “Full STEAM Ahead: The Benefits of Integrating the Arts into STEM,” *Complex Adaptive Systems*, vol. 20, pp. 547–552, 2013.

[10] D. Jean, B. Broll, G. Stein, and A. Ledeczi, “Your Phone as a Sensor: Making IoT Accessible for Novice Programmers,” *Proceedings of the IEEE Frontiers in Education Conference*, pp. 1–5, 2021.

[11] S. Pokress and J. Veiga, “MIT App Inventor: Enabling Personal Mobile Computing,” *arXiv*, pp. 1–3, 2013.

[12] Á Lédeczi et al., “Teaching cybersecurity with networked robots,” in Proceedings of the 50th ACM Technical Symposium on Computer Science Education, pp. 885–891, 2019.

[13] Y. B. Kafai et al., “Stitching the Loop with Electronic Textiles: Promoting Equity in High School Students’ Competencies and Perceptions of Computer Science,” *in Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 1176–1182, 2022. doi: 10.1145/3287324.3287426.

[14] Y. Tan, M. Rizk, G. Stein, and Á. Lédeczi, “User-Extensible Block-Based Interfaces for Internet of Things Devices as New Educational Tools,” *in SoutheastCon 2022*, pp. 711–717, 2022. doi: 10.1109/SoutheastCon48659.2022.9763937.