MUzECS: Embedded Blocks for Exploring Computer Science

Matthew Bajzek, Heather Bort, Omokolade Hunpatin, Luke Mivshek, Tyler Much, Casey O'Hare, Dennis Brylow
Marquette University
firstname.lastname@marquette.edu

***Abstract*—We build on the success of the Exploring Computer Science (ECS) curriculum, which has improved the accessibility of a Computer Science education to students of all backgrounds. While ECS has been well-received by its students and successful in reaching many students of diverse backgrounds, it currently suffers from a final module which is expensive and offers no easy transition to text-based programming.**

This module, which teaches robotics as an application of computing, rests on the LEGO Mindstorms[1] platform. The Mindstorms have a block-based programming language that abstracts too much code to be of any use for making a transition from block-based programming to text-based programming. They are also more expensive than most schools can afford. This paper reports on one aspect of our successful effort to create an alternative curriculum which meets the same curriculum objectives as the current sixth module, but for a reduced price.

Keywords*—*Computer Science, ECS, block-based programming, paper, Arduino.

I. Introduction

Despite its critical and growing importance, Computer Science is taught in only a small minority of United States high schools.[1] The MUzECS project has designed the hardware, software, and curriculum for a new, low-cost module to replace the expensive existing final module for Exploring Computer Science[2] , an introductory high school curriculum. ECS is aimed at increasing the accessibility of Computer Science education for women and minorities in order to improve the diversity and student opportunity within computing courses.

The curriculum has proven effective in its goal, and have improved upon it by both lowering cost and creating a dialect of Ardublock[3] upon, thus making the ECS curriculum a better and more widely avalible learning tool. This dialect is what our entire module relies upon to be the interface students have with learning basic programming through our module.

A. Project Scope

In order to offer a cheaper alternative for low-income schools to teach Computer Science courses, we have created a new module based on an entirely new platform. Our design utilizes the Arduino Leonardo, a streamlined set of simple peripherals, a rewritten class curriculum, and a custom block-based programming language.

Students write code in our dialect of Ardublock: an open-source hobbyist language already available for enthusiast use and modification which sends its blocks as strings of code to the Arduino IDE to deploy it directly to the Arduino board. The board receives input and provides output using the set of peripherals included in the Arduino shield, a board containing peripherals that can be plugged directly into the top of the Arduino, we provide. The small embedded system created by this combination of parts will closely follow the content of the previous modules during which students learned to program using Scratch[5], a block-based programming language akin to Ardublock.

The curriculum we deliver has well-defined lesson plans, assignments, and projects which meet specific student learning needs. This curriculum has been piloted at half-a-dozen schools totaling seventy individual hardware units being tested and used with our software. The student data from this pilot study last spring are currently being analyzed to see if our curriculum meets the same objectives as the current module. The data that has been processed points towards a successful integration into the curriculum.

The curriculum is backed by a custom circuit board

[1] http://www.exploringcs.org/resources/cs-teaching-resources

Fig. 1. The shield and its peripherals attach to the Arduino GPIO headers

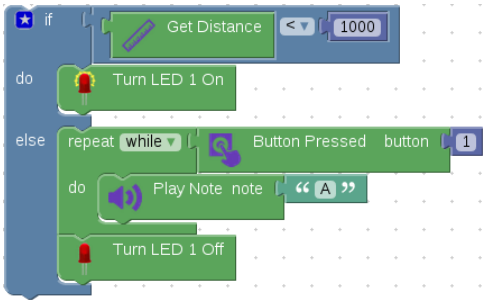


Fig. 2. Example use of control structures similar to high level programming

shield, as seen in figure 1, which mounts directly onto the General Purpose Input/Output (GPIO) headers of the Arduino Leonardo in order to power its peripherals, which the students will control via our final deliverable, the custom block-based language of MUzECS. This language, created by extension of Ardublock is to be used by students via a simple interface familiar to them from their Scratch training. Code will be deployed from this interface using a single button that prompts the Arduino software to compile to the board, ultimately executing the students' programs.

We support the natural progression to an adept understanding to text-based languages from block-based by supplying control structures as blocks and making all other blocks compatible with these. The control structures in use are if-block, if-else, if-else-if, repeat-until, repeat-while, and for-loop. A classroom could use these to create a variety of applications. For example, a block program where when an object is close it turns LED one on and when it is far away turns off and while button one is pressed a note plays, as seen in figure 2.

II. Previous Research

There are many widely used visual block based programming languages and dialects in computer science education. These visual programming languages serve the needs of high school teachers with varying degrees of success. We built MUzECS to fill some of the needs of a high school computer science curriculum that some visual block based programming languages don't completely satisfy.

For instance, Scratch, is a visual programming language that makes it easy to create your own interactive stories, animations, games, music, and art. In addition, Scratch is very tinkerable, social, and diverse in the projects one can make with it. This is ideal for high school and middle school students who are just beginning to learn how to program. Scratch is easy to use and makes block programming interesting for students. Unfortunately, we have found that Scratch doesn't originally work with any open hardware and doesn't facilitate the progression to text based programming languages.

Furthermore, ECS found a block-based robot set, which supplied a visual block based programming environment and interactive hardware, that was currently being deployed in many schools' computer science curricula called Lego Mindstorms NXT[7]. There is a programmable robot set, which comes with several types of sensors, motor, and programmable brick, named Mindstorms NXT(NXT) made by the Lego Company.

MUzECS is designed to cost a fraction of Lego Mindstorms, while still supporting equivalent curriculum goals. Our MUzECS set is sixty dollars compared to six-hundred dollars for the LEGO Mindstorms NXT 2.0 by the Lego Company on Lego.com[2]. According to Barry Fagins, who is the developer of Ada for Mindstorms and has used Mindstorms in courses[6] there is no proof that Lego Mindstorms improves learning or enhances retention in computer science education.

In making our visual programming dialect MUzECS usable on all operating systems that can run the Chrome browser, we built it off of Google's open source visual block based programming environment called Blockly [4].

A. MUzECS Ensemble

Natural progression is further facilitated by requiring the user to click the 'Arduino' tab showing the Arduino code before they can upload to be compiled. This ensures the user sees the Arduino code after they use the blocks

[2]http://shop.lego.com/en-US/LEGO-MINDSTORMS-EV3-31313?p=31313&track=checkprice

and can make connections between the conversion. This connection will hopefully ease the transition to a text-based language.

Within MUzECS there is also an 'Advanced (Pins)' section where more complicated blocks are available. We have chosen to implement this to give the user more freedom and thus creativity. The labeling 'Advanced' signifies a user with knowledge of the pin numbers on the Arduino which correspond to different hardware peripherals on the shield can use these blocks. This is also where users who would like to use the Arduino pins labeled 'advanced pins' on the shield can come to use their own peripherals with the Arduino. If a user would like to use MUzECS without the shield designed for it, this section allows for direct pin reference on the Arduino and can be used to set up other peripherals on the Arduino.

III. Blocks

Throughout the creation of the software, hardware, and curriculum in the MUzECS project the hardest obstacle to overcome in the whole project was creating the design for what blocks are available for student use and how they are presented in the MUzECS dialect of Ardublock. Although it is built to be usable by both beginners and those familiar with programming, the project is geared towards helping teach a ninth grade audience who are learning to program. The design decisions for how MUzECS blocks were all made with this goal in mind.

The block interface allows MUzECS to show users multiple versions of simple ideas with differing complexity while allowing more difficult programming to be abstracted behind simpler blocks. The entirety of the design attempts to make programming not only easy for a ninth grade beginner, but to provide them stepping stones towards the comprehension of more complicated ideas in programming and ease the transition from our block-based programming dialect to more standard text-based languages.

A. Overlapping Block Objective

In many high level languages there are many ways to accomplish a single objective which is why we implemented a similar ability in MUzECS. Some sets of blocks in the MUzECS dialect accomplish this in order to provide the tools for students to learn how the blocks behave in written code.

Fig. 3. Example setup of how to play musical notes with the different methods provided by MUzECS

For example, the blocks 'play note time' and the combination of 'play note' or 'play note frequency' with 'no tone,' and 'delay,' as shown in Figure 3, all hold the same function: playing a musical note. 'Play note' time is the easiest to use for a beginner and holds the most abstraction of the three choices as it automatically takes a standard musical note and converts it to that note's frequency then sets a 'delay' until the note should be terminated at which point it sets a 'no tone'.

As students become more familiar with how the 'play note time' block works and the code behind it they are able to move forward and understand the building blocks of 'play note time' as they can access all of its components in block form. 'Play note' is an intermediary step that not only allows simple use of notes without having to look up frequencies for each note, but can be used to understand the underlying code to 'play note time' without comprehension of what a frequency is and how that can be used to make musical notes.

The above layering of block operation offers user friendliness and intricate control in how a note is played. It allows a user to create a song from strict notes or for sounds via frequency to depend on a changing variable or to play a desired pitch between standard notes.

B. Block Abstraction

Using blocks to abstract more complex code can be used in ways other than layering their use with other blocks. This method of abstraction is helpful to hide code that is much too complex for a beginner to comprehend.

The 'get distance' block is an example of this abstraction. It may seem to be a simple data return as all it gives is the distance of an object, but it has complicated programming behind it. '[G]et distance' returns refined data that uses the time it takes an ultrasonic ping to bounce off of an object and return to the sensor. That data is then filtered and smoothed prior to being sent to the user's computer.

The block-based nature of the MUzECS dialect of Ardublock is perfect to put this code behind a simple

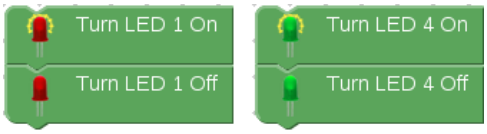


Fig. 5. Visualization queue to quickly assist beginners in understanding what their code will do

block in the interface. We have also put it into its own function within the text-based code. This means that a beginning programmer can just as easily look at the text-based code created by MUzECS and see their program with a simple function call to getDistance(). A user may then look at both the text-based and block-based code and see a direct one-to-one correspondence between their blocks and the Arduino code as shown in 4.

C. Visual Aids

We have found it is important to visualize the one-to-one relationship between code and output. It saves time and aids in a friendly user interface. To add this visualization, many blocks in MUzECS have associated images with them. An example of such is 'turn LED # on' with a bright colored LED image, 'turn LED # off,' with a dark LED image, as seen in Figure 5. Similarly, 'play note' has a speaker and sound waves, and 'set up keyboard' with a keyboard image.

These encourage the user to visualize what their code will do prior to compilation. When a user is able to imagine what their code will do without compiling it and seeing the output is first, they are able to make less mistakes and save time. This is also necessary for our user's natural progression to adept programmers.

D. Block Expansion

We bolster the transition to text-based programming languages from block-based ones through use of setting up resources for input and output devices. The keyboard runs via the three blocks: 'setup keyboard,' 'update keyboard,' and 'key pressed' all of which in conjunction allow a student to read input from a keyboard just as they would a button on the shield. However, unlike a button the method to poll for a keyboard key is much more complicated than polling for a button press.

This means that while 'button pressed' is a single block the keyboard uses three. This complication is due to the button being directly on the shield, making it fairly easy to interact with while the keyboard needs

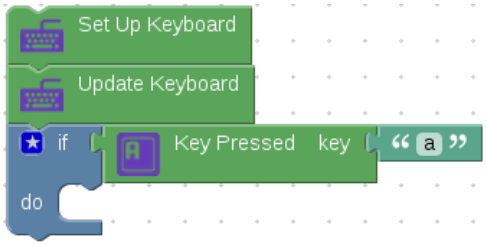


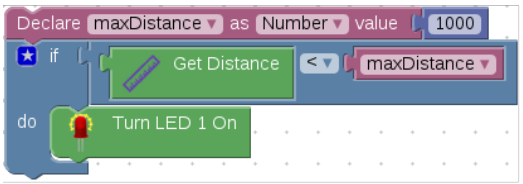
Fig. 6. Setup of I/O to assist in transition to text-based languages

to be found on the user's local system and interacted with via a queue. In order to use the keyboard the student must use the blocks in the following order: 'setup keyboard,' 'update keyboard', then 'key pressed,' as shown in figure 6. While all three of the blocks could be combined into one, keeping them separate allows the students using MUzECS to learn the basic differences between communicating with a button on the shield. The 'setup keyboard' block defines the variables being used by the other two blocks while 'update keyboard' sets up communication with the local computer to talk to the keyboard and checks the buffer for recently pressed keys and flags them. Finally, 'key pressed' checks for the flag of a specific key and returns true or false.

This separation allows a student to comprehend what steps are going into the process to communicate with the keyboard and its complexity compared to communicating with a button on the shield even if they cannot grasp the specific code elements of each block. This introduces the student to the idea of communicating with different I/O devices on the computer and helps take a step towards a better understanding how the Arduino works and takes the student closer to being able to use and interact with the 'Advanced (Pins)' or 'Advanced (Code)' sections of MUzECS.

E. Error Handling

Similar to standard IDEs for text based coding which attempt to point out as many possible errors prior to compilation, MUzECS uses a system of alerts. For example, the 'button pressed' block takes any integer as input for a button, but only an integer containing only a valid button number will get passed on to be compiled. When an invalid button number is passed and the code is attempted to be sent to the Arduino IDE for compilation, MUzECS alerts the student prior to the code being sent and the blocks are converted to code. The alert message in this case consists of "Invalid button used in block. Hint: Valid button numbers are 1-4."




```
void setup()
{
  maxDistance = 1000;

  Serial.begin(9600);

  pinMode(3, OUTPUT);

  pinMode(2, OUTPUT);

  pinMode(5, OUTPUT);

}


void loop()
{
  if (smoothDistance() < maxDistance) {
    digitalWrite(5, HIGH);

  }

}
```

Fig. 4. Example of the code mapping in both text-based and block-based of a sample program that uses a variable and lights an LED if an object is close to the shield

Additionally, 'play note' works in a similar fashion. This allows students to learn the basics of handling poor argument passing and other coding practices to be avoided prior to learning about the compilation process and how to interact with a more complex IDE in order to find the bugs in their code. Similar alerts are thrown whenever a user attempts to compile code that has errors in it such as attempting to use an undefined variable (Arduino uses on its own flavor of C) or improperly setting up the overarching loop structure embedded systems like Arduinos run on.

IV. Future Work

We will be collecting block usage statistics from high school students and teachers, who are already using MUzECS, through the use of surveys. We are eager to see if block usage statistics from MUzECS indicate a natural progression towards text based programming languages and stimulates a penchant for programming among our users. The kind of conclusions we hope to draw from this data collection is whether our users liked using MUzECS; whether they found using MUzECS difficult in any way; what bugs they encountered with MUzECS; and what features they want added to MUzECS. Our analysis will include a coding of student final projects with both a MUzECS experimental group, and a Mindstorms control group to assess whether students are achieving similar or improved outcomes with MUzECS.

Our next step is to develop more additions to the shield in order to foster more creativity among our users and expand curriculum. We have plans to add more sensors in addition to the distance sensor on the shield, such as a temperature sensor, a light sensor, a pressure sensor, and an accelerometer.

After speaking with a significant part of our user base, we have noticed that some schools are shifting to using Chromebooks in their computer science curriculum. This is understandable as Chromebooks are cheap compared to standard laptop prices, and fit most of high school computer science's needs. Unfortunately, Chromebooks can't install Arduino software. Due to this, we have developed a Chrome browser based web-page and web app to eliminate limited platforms. Anything that can run a Chrome browser can use our software and hardware in tandem. This is planned to be rolled-out to schools in Wisconsin this coming school year for feedback.

V. Conclusion

As a final point, our main goal was to design a low-cost alternative to the sixth module of the Exploring Computer Science curriculum that we piloted in schools during the Spring 2015 semester. While future work does exist and analysis of survey data will no doubt provide useful insight into the strengths and shortcomings of our designs, our team is confident that we have successfully produced hardware, software, curriculum, and resources which meet our customers needs and wants and fulfills our main objective.

As has been mentioned several times throughout this document, our projects ability to meet the subjective customer needs put forward during the initial design phase of this project are to be assessed in the future after the collection of teacher feedback and survey data post-piloting of our module. Once these data are collected,

there will be extensive work to be done in improving our designs to meet the feedback. In addition, further testing work will need to be performed as new features are implemented in order to guarantee the continued reliability of the MUzECS platform. Our team has already made plans with Dr. Brylow to continue these efforts into the fall and beyond to ensure the longevity of the project.

VI. Acknowledgment

Some of the students on the MUzECS team were supported by the National Science Foundation, grants CNS-1339392, and ACI 1461264. We would like to thank our pilot teachers and students. We owe special thanks to curriculum consultants Robert Juranitch and Gail Chapman. The pilot curriculum testers both teachers and students alike, software and hardware included the work of previous student researchers Alex Calfo, Farzeen Harunani, Jonathan Puccetti, and John Casey. The MUzECS project was built on top of the existing Ardublock system which in turn was built on the Blockly architecture.

References

[1] Code, https://code.org/promote

[2] ECS, www.exploringcs.org

[3] http://blog.ardublock.com

[4] Google Blockly, developers.google.com/blockly/

[5] Resnick, Mitchel, et al. "Scratch: programming for all." Communications of the ACM 52.11 (2009): 60-67.

[6] McNally, Myles, et al. "Do lego mindstorms robots have a future in CS education?." SIGCSE. Vol. 6. 2006.

[7] Kim, Seung Han, and Jae Wook Jeon. "Programming LEGO Mindstorms NXT with visual programming." Control, Automation and Systems, 2007. ICCAS'07. International Conference on. IEEE, 2007.