UNC++Duino: A kit for learning to program robots in Python and C++ starting from blocks

Luciana Benotti, Marcos J. Gómez, and Cecilia Martínez

Universidad Nacional de Córdoba,
Haya de la Torre y Medina Allende, Córdoba, Argentina
firstname.lastname@unc.edu.ar
http://masmas.unc.edu.ar

Abstract. We present UNC++Duino, an open source educative software for learning to program a robotic kit in C++ and Python. Besides of these two industry programming languages, UNC++Duino can be programmed using 2 high level languages based on blocks are free of syntax errors. One of the block based languages included is completely iconic allowing for its use with preliterate children. The hardware we use with UNC++Duino, the open RobotGroup robotic kit, can be used to build different automated constructions based on an Arduino board, sensors and actuators. UNC++Duino was developed within Argentinean K-12 schools by the Universidad Nacional de Córdoba with the collaboration and support of the Argentinean National Ministry of Science and the RISE program in Google for Education. Its goal is to provide an engaging tool for learning to program in different programming languages with increasing difficulty and control of the hardware.

1 Introduction

Why is it necessary to find effective and innovative ways of engaging more students into Computer Science (CS)? Taking our country as an example, we know that Argentinean universities graduate approximately 4000 CS students a year (compared to 10000 in Law and 15000 in Economics) while the national industry needs to hire twice that amount per year. The lack of human resources in CS has an economical impact in Argentina. The Information and Communication Technology (ICT) industry in Argentina, despite having grown intensely in the last ten years (four times in the number of employees, and nine times in the amount of exports), is still struggling to find qualified workers for its workforce.

Part of the reason why students do not choose to pursue a career in CS is that, neither programming, nor other CS techniques and concepts, are taught at school. Previous studies show that this lack of early CS education can influence career choices; students may not be selecting CS simply because they do not know what CS is [4,10]. Since it is not taught at school, misconceptions about what the discipline actually is are commonplace. Indeed, recent work found that although more than 90% of Argentinean high school students surveyed use computers, most of them believe that programming means installing programs [2].

The typical K-12 student in Argentina never encounters CS topics during his/her school years. Programming is never taught at school, not even as an optional course. The ICT curriculum in Argentina focuses on user training classes rather than CS content. In these courses, computing entails little more than learning how to use a word processor, a spreadsheet or how to write an online blog. Students often get bored in their ICT classes and outperform their own teachers. This context is not unique to Argentina; many developed countries share the same problem. For example see the cases of the USA [20] and the UK [9]. There are some exceptions such as Israel, where CS has been taught at (some) high schools for many years now [22], and other countries are starting to follow. This is the case, for example, of New Zealand [1].

Children are using computers much earlier each decade. However, this intensive use is not contributing to their knowledge of CS as a discipline nor to developing the ability to understand a programming language. Children become software consumers very early but they do not learn the basics of how this technology works.

Given the current situation, there is increasing consensus that teaching programming in particular, and CS in general, in an engaging way in the school curriculum is necessary. It is imperative to both include students in the technological world as active and creative citizens, and to help them make educated choices about their professional future.

Bergen [3] points out that programmable toys such as robots, give children the possibility of creating, imagining, and exploring. Robot programming is at the same time engaging, stimulating and rich of many important CS concepts where the digital world connects to the real world.

With the goal of teaching programming and other CS important concepts in an engaging and interdisciplinary way the main contributions of this paper are:

- Introducing a multilanguage robot programming platform that allows children to change from one language to another making evident the unimportance of the particular syntax, helping students discover new concepts and learning new techniques when they are ready.
- Describing an open source educative robotic kit that can be used to build an unlimited number of automated prototypes based on Arduino boards using just a screwdriver—a 3-wheel robot, an elevator, a harvester, an automated house, etc.
- Proposing a set of original activities and suggestions for adapting the same activity for preliterate, primary school and highschool students. The activities make use of the programming kit to teach not only programming but also propose how to integrate content from other disciplines (e.g. astronomy and physics).

2 Previous work

Starting very young, children can create, run and debug simple computer programs using programming languages that are both challenging and attainable for their age [5,14]. Even children who are preliterate, have diverse background and different developmental stages, can program tangible platforms using iconic commands [7,11]. The effects and implications of learning CS at such an early age have been analyzed in previous work. According to Clements [5], young children who program concrete objects have the opportunity to analyze a situation and reflect on the properties of the objects they have to manipulate.

While exploring how to teach CS to little children, researchers found that the difficulties in children programming laid in their immature motor skills and on syntax problems [16,8]. Thus, there has been an important development on specific programming platforms to address the developmental traits of preschool children (such as Toon Talk [17], Scratch Jr [15], Cherps [19], among others). The kit we present in this paper is one of such programming platforms, particularly designed for programming robots and other automated constructions. Differently from previous work, it offers an integrated way to grow from simple block programming languages into full fledged languages such as Python and C++.

Flannery and Berns [7] showed that as a result of robot programming in preschool, children imagine, plan its action, and construct a robot. In their study, the authors found that all 4-6 year old students could program short challenges and explore robot's capabilities.

Although most interventions to teach programming with robots achieve high student engagement and task completion, most available robotic programming kits are not accessible to schools for their high cost or their lack of flexibility. Some kits are tailored to be used with one (sometimes proprietary) block-based, or otherwise simplified, programming language. This rigidity hinders children's and teacher's creativity limiting the possibility of creating and solving new challenges appropriate for different age groups.

The following studies compare different age groups performance on similar robot programming activities. Magnenat and his colleagues [12] taught CS with robot programming to different age groups of children using an event handler language to program robot actions in response to different events. Comparing the groups performance in the same task, they found that most children understood and solved simple tasks such as moving a robot upon a touch of a button or identifying robot's instructions. However, older children performed better on complex programming that required several conditions or events. Dagiene et al [6] compared students from Finland, Sweden and Lithuania from 7 to 12 years-old performing similar algorithmic thinking tasks exercises. Using multiple choice questions, they evaluated concepts such as algorithm modularity, data structures, and control flow. They found no strong difference across age groups, but rather among countries. The authors suggest that educational context, academic quality and in particular, reading ability promoted by each school system may be strongly related to learning programming. Thus, we want to highlight the value of developing a completely open source robotic programming kit[1] such

[1] The robotic kit software as well as the hardware are open source and the sources are available at `http://masmas.unc.edu.ar` and `http://robotgroup.com.ar`

as `UNC++Duino`. Using it, teachers and students can choose not only what kind of robotic construction they want to build—a 3-wheel robot, an elevator, a harvester, etc—, but also the programming language they want to do it in—a high level, iconic language or a low level language such as C++, or Python. Python is one of the top programming languages used nowadays by universities in introductory programming classes. Having different programming languages accessible side by side in an interactive development environment, help the students explore and move into more complex languages when they are ready.

In this paper we present the robotic kit hardware and the multilanguage programming software `UNC++Duino`. `UNC++Duino` was developed by the Universidad Nacional de Córdoba in Argentina with the collaboration and support of the National Ministry of Science and Technology and the RISE program in Google for Education. The hardware, described in Section 3, can be programmed in parallel in different programming languages organized in an increasing order of difficulty and control, as described in Section 4. Section 5 propose a set of original activities and suggestions for adapting the same activity for preliterate, primary school and highschool students. The activities make use of the programming kit to teach not only programming but also propose how to integrate content from other disciplines.

3 Hardware: The robotic kit

The hardware that we use was designed by the Argentinean company RobotGroup. The RobotGroup kit, when used to build a 3-wheel robot as shown in Figure 1b, is small (13x13x12cm) but it includes an interesting set of sensors and actuators. It has 2 engines and gearboxes of 200 rpm located in the two front wheels. It includes two IR sensors located where the "eyes" of the robot are. These sensors can be used to test for proximity. It also has two sensors on the bottom that measure the ground reflectivity and its colour. As can be seen in Figure 1a, the Arduino board contains standard I/O connectors.

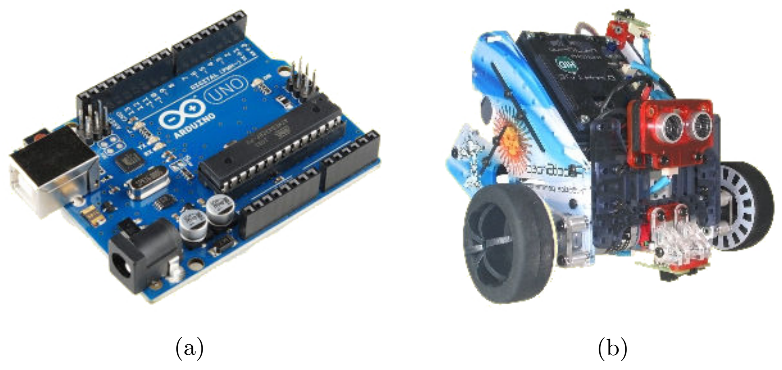


Fig. 1: The arduino board used by the kit and a 3-wheel robot built with it

The kit also has a USB 2.0 port and 6 connectors for analogical 10-bit sensors. A programmable array of user leds is also included, as well as leds indicating power, and engine direction rotation. Extensions can be added such as the standard shields Arduino-compatibles (WiFi, Ethernet, ZigBee, extra engines, etc.). Finally, the kit also includes a microphone, a sound synthesizer and an IR sensor for remote control.

We use this hardware because of the following advantages. To begin with, it is affordable and usable out of the box, allowing schools and parents to buy it with a reasonable budget. At the time of writing this paper, it was 4 times cheaper than the well known Lego® Mindstorms® (and even cheaper if less sensors and actuators of those described here are included). Moreover, the hardware design is open and based on Arduino boards supporting Grove System; which is widespread and continuously evolving. The board used is illustrated in Figure 1a. In addition, it includes several sensors, actuators and ports in order to add extra ones. Finally, the kit is very flexible allowing for the construction of an innumerable number of automated constructions. For example, it can be used to build a 3-wheel robot (1b), a mill or Ferris wheel (2a), a driverless cart (2b), a toy combine harvester (4a), a small crane (5b), etc.

The Ferris wheel (illustrated in Figure 2a) can be programmed to turn on an off different leds and to play music when reacting to some programmed condition coming from its sensors. Or it can be transformed, by just using a screwdriver and some programming, in a driverless toy cart (illustrated in Figure 2b) that can deliver toys to different rooms of a house. In the next sections we give more examples of automated constructions and how they can be programmed, but we believe that the possibilities are endless.

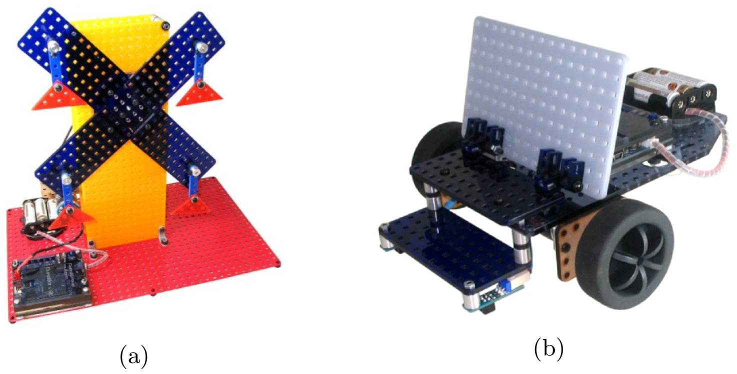
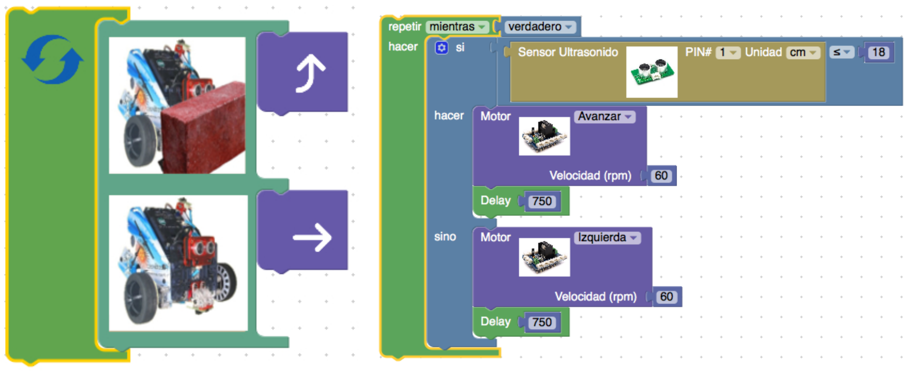

(a) (b)

Fig. 2: An interactive Ferris wheel and a driverless cart that deliver toys

4 Software: Programming with Blocks, Python and C++

The UNC++Duino programming environment includes different programming languages. Its two block languages are based on BlocklyDuino[2], a platform that builds on Blockly. BlocklyDuino was developed by Google for programming Arduino boards supporting Grove System[3] with blocks. A fragment of code in the block language created by us for preliterate children is shown in Figure 3a, a fragment of the original BlocklyDuino code is shown in Figure 3b. The code has a repeat while true that includes an if-then-else instruction. This instruction means that if there is an obstacle in front of the ultrasound sensor then the robot will move forward, otherwise it will turn left.



(a) (b)

Fig. 3: The same code written in UNC++Duino iconic language and Blockly

Many educative platforms like Code.org (in their Hour of Code initiative [21]) or MIT APP Inventor [18] use Blockly. Blockly is open-source under the Apache 2.0 License. UNC++Duino extends BlocklyDuino to adapt it to the robotic kit described in Section 3 by creating new blocks for some sensors and actuators. For example, we added a block to play different songs using the robot sound synthesizer.

More importantly, we also extended BlocklyDuino to add the iconic language illustrated in 3a. UNC++Duino can also be programmed in other full programming languages such as Python and C++, each language was selected because it offers a different level of difficulty and expressiveness. The simplest one is the iconic language, but it is also the one that offers less control to the student. In this language the student cannot choose, for example, the speed of the robot, its speed is defined as a constant. The iconic language represents the move forward action

[2] https://github.com/BlocklyDuino/
[3] http://www.arduino.cc/

by a straight arrow that moves the robot forward a fixed length (set arbitrarily to 20cm) in the block code. If the student wants the robot to move exactly 1, meter she will have to include 5 forward actions in her code (or a block that repeats 5 times). Likewise, the turn arrow represents turning a predefined angle (set to 90 degrees). The language also includes simplified blocks for conditionals as can be observed in Figure 3a. This design decision means that a different conditional block needs to be created for each condition that wants to be tested. In the figure the condition is: is there something blocking the view of the robot? This leads to a proliferation of blocks but we thought it was a reasonable compromise for conditions that are frequent when teaching preliterate children.

UNC++Duino interface offers all programming languages in different tabs. The automated translation into Python or C++ occurs when the student changes to the Python or C++ tab and the interface detects that the block code has been modified. If the Python code is modified then it is also automatically translated into C++. The translation is not done backwards, it is not bidirectional. That is, if a student changes directly the code in Python or C++, the block code is not updated and a warning sign is showed in the interface. It is not possible to make the translation bidirectional because some of the programming languages used are less expressive than others. For example, in the iconic language it is not possible to express that the robot has to advance exactly 10cms. We informed the students in advance that the translation is not bidirectional, and they had no problem with it. We assume that children will eventually explore text languages, seeking to have more control of the robot functions and grow out of the iconic interface.

UNC++Duino is a web platform programed in JavaScript and runs in most web browsers. The interface is minimalistic, it shows only one programming language at a time, and the student can easily change from one language to another by tabs. It includes a button that sends the code to the robot and another one to share the code (e.g. with the teacher or a fellow student). The block languages include a menu of the available blocks that the student can drag and drop building the code as a puzzle.

Since UNC++Duino is open source, it allows users to create a new block, and to define its translation to Python and C++. We have defined already a set of blocks that are useful for the specific hardware we have been working. But, the software was designed to be easily extendable by adding new blocks as well as new programming languages other than Python or C++.

We can create a new block by performing the following two steps. First, we define the block; specifying its name, color, structure (that is, whether it requires some nested code such as the loop or the conditional) and its parameters. The new block definition can be directly written in JavaScript, referring by code to the jpg or png image(s) that represent the icon. When we created the if-then-else block shown in Figure 3a we used two photos of the robot, one with and one without an obstacle. Instead of programming the new block, it can be designed graphically using the Blockly tool designed for this purpose[4].

[4] https://blockly-demo.appspot.com/static/demos/blockfactory/index.html

Second, we define the translation of the new block to the specific programming language we want. `UNC++Duino` already includes two files, the python.js and arduino.js, that are responsible for merging the code generated by the translation of a set of blocks into Python and C++. If we want to translate the new block (or an existing block) in another language, we also have to develop the merging code file. The following Python code is generated by python.js when automatically translating the code in Figure 3a.

```
def avoidObstacles(robot):
while True:
     if robot.ping() <= 20:
         robot.turnLeft(60,0.5)
     else:
         robot.forward(60,0.5)
```

The robot's methods `ping`, `turnLeft` and `forward` chose the appropriate sensors or actuators to use. The method `ping` uses the ultrasound sensor to check that the road is clear for at least 20cm; `turnLeft` rotates one wheel in order to turn left at 60rpm during half a second, and `forward` rotates both engines at 60rpm during half a second.

Once we have defined the block and generated the code, we have to the compile the new code. The interface will then be updated to include the new block or programming language.

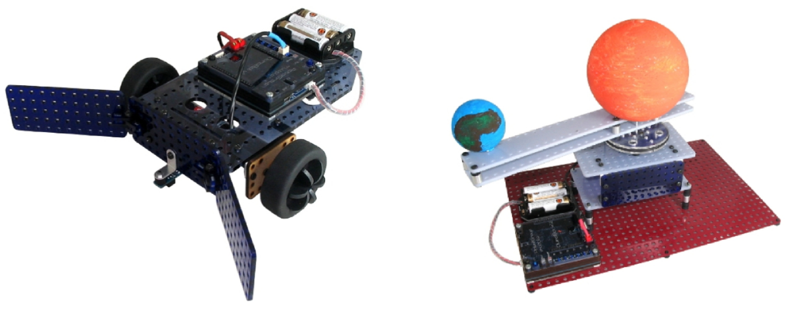
5 Activities for different age-groups in the classroom

In this section we describe different activities developed in classroom with the `UNC++Duino` platform. Our iconic programming language, allows people from a wide range of ages to program: from preliterate children to students who are taking their first steps in programming. This high level language let students focus on the program structure rather than on the syntax and other technical details until they are ready to do so. Indeed, although originally designed for little children and evaluated with them [14], `UNC++Duino` proved to be engaging but also challenging for in-service K-12 teachers when taking their their first steps in programming [13]. One of our goals of `UNC++Duino`, is that students can grow with the platform, so we describe here a set of activities that we designed in order to illustrate how to take advantage of the flexibility of the tool.

Consider again the challenge of programming a robot that could avoid objects. In the previous section we showed how this challenge can be solved using the iconic language and also the Blockly language (Figure 3). We also showed how the same problem can be solved using Python. In the Python method `avoidObstacles` the students can directly modify the parameters to change, for instance, the speed of the robot, or the distance from the obstacles that the robot tolerates. They can also make the robot move backwards before turning by sending a negative parameter to the method `forward` (something that will prove necessary if they define the ping distance too low).

Once students have programmed the robot that avoids objects, we could add an extra challenge: create a firefighter robot. Teachers and students should design a maze with different objects that the robot should avoid. In a part of the maze, the teacher turns on a candle. The robot has to find the candle and turn it off. The room has to be dark, because the robot can detect where the candle is using photosensors.

Instead of programming a robot that avoids objects, students could program one that collects them. With the robotic kit and a screwdriver students can construct a toy "harvester" similar to that illustrated in Figure 4a that can "harvest" a predefined field inside the classroom collecting crumbled yellow paper balls representing the grain. Fields can be delimited with color tape so that the robot bottom sensors are used to stay inside. A competition between different students groups can be organized; the student who collects more grain wins.



(a) (b)

Fig. 4: A combine harvester that collects grain and a planetary system

Teachers from other disciplines could integrate Robotics and CS into their curriculum. A planetary project could be part of the activities. Using the educational kit, as illustrated in Figure 4b, we can show our students a solar system structure, and tell the children that their challenge is to make it move like the sun and the earth move. We can introduce programming concepts such as *parameter* and *cycle*. Younger children would use iconic blocks, that would help them move the engines faster or slower, in order to experiment with different speeds.

High school students could program the same experience, but they could build the planetary system as well. In this way, they will interact directly with arduino board, sensors and engines. The programming goal would be to decompose the problem, and work with programming concepts such as *conditionals*, *cycles*, *methods*. They could also program the constants and calculations necessary to mimic the movements of the earth and the sun on a faster scale. Another challenge could be to turn on a led when the country they live in is not facing the sun.

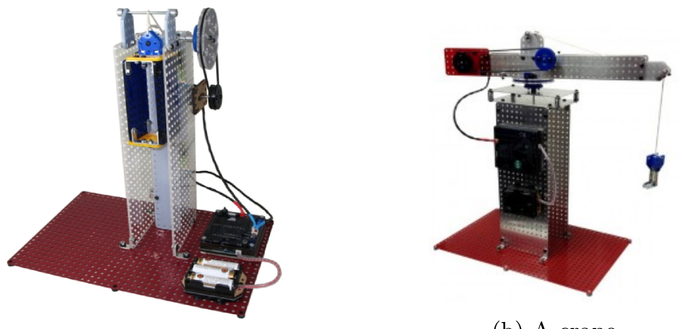

(a) An elevator (b) A crane

Fig. 5: An elevator programmed with a queue and a crane

A good idea could be to create something related with students life. Elevators make children feel curious about how they work. If we are in a mall or building, children want to use the elevator. Here we define a challenge to develop an elevator at school.

For kindergarteners, we can talk about elevators and how, in general, they work. With the educational kit, teachers could build the elevator, and little children could program it with `UNC++Duino` iconic blocks. The elevator will be equipped with 3 buttons that students will activate when testing the elevator (ground floor, 1st floor and 2nd floor).

High schools students, should construct the elevator using the educational kit. Then, they will incorporate programming, using `UNC++Duino` language blocks to begin with. Students could use events for programming the elevator. To resolve the elevator problem, students will have to manage when different users of elevator touch different floors buttons. How the elevator should behave to be fair to the people waiting for it? More advanced students could move to Python and use the queue data structure.

6 Discussion and conclusions

`UNC++Duino` has been evaluated by teaching programming to kindergarten and primary school children [14] as well as for in-service teacher training in a Professional Development program [13]. For precise evaluation data we refer the reader to these articles. Below we summarize their findings.

These studies found that most evaluated subjects, starting from 6 years-old, could learn fundamental programming concepts such as sequence, loops, parameters, conditionals through `UNC++Duino` and were capable of applying these notions into robot programming. However, older children (from 10 years-old) could combine these concepts better to create a new program [14]. The study described in [13] found that, after a 50-hour teacher training program, teachers could use the programming concepts but only those with previous background in CS were able to fully explain them to their students. The authors in [14] report that students showed high engagement and that UNC++Duino triggered exploration. Most students explored and commented on the different tabs with other programming languages and were able to notice the parallelism between the equivalent program structures. Indeed, some students were able to modify parameters in the text languages that were not modifiable in the block languages although the activity was not designed with this goal in mind.

In this paper we presented UNC++Duino, an open source educational software for learning to program a robotic kit in C++ and Python starting from drag-and-drop programming languages. One of the block based languages included is completely iconic allowing for its use with preliterate children as well as with beginners. Besides simplifying the initial steps, the code resulting from translating block code into text languages is designed to be highly modular, as illustrated in Section 4.

In general, children grow out of computer platforms designed for a particular age group very quickly, this is good for children but not so for teachers who sometimes cannot keep up learning to use different complex (and many times proprietary) interfaces. Our platform encourages students to grow with it and because of it. We acknowledge that further research is necessary with multi-language platforms for CS Education, but this, combined with robotics could be one direction to encourage children (and teachers) growth in CS.

Acknowledgments

This work was partially funded by the grants PICT-2014-1833, PICT-2012-712, PDTS-CIN-CONICET-2015-172, and PID-2012-2013-R18.

References

1. T. Bell. Establishing a nationwide CS curriculum in New Zealand high schools. *Communications of the Association for Computing Machinery*, 57(2):28–30, 2014.
2. L. Benotti, M. C. Martínez, and F. Schapachnik. Engaging high school students using chatbots. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ITiCSE '14, pages 63–68, NY, USA, 2014. ACM.
3. D. Bergen. Technology in the classroom: Learning in the robotic world: Active or reactive? *Childhood Education*, 77(4):249–250, 2001.
4. L. Carter. Why students with an apparent aptitude for computer science don't choose to major in computer science. *SIGCSE Bulletin*, 38(1):27–31, 2006.
5. D. H. Clements and J. Sarama. Teaching with computers in early childhood education: Strategies and professional development. *Journal of Early Childhood Teacher Education*, 23(3):215–226, 2002.
6. V. Dagiene, L. Mannila, T. Poranen, L. Rolandsson, and P. Söderhjelm. Students' performance on programming-related tasks in an informatics contest in Finland, Sweden and Lithuania. In *Proceedings of the 2014 Conference on Innovation; Technology in Computer Science Education*, ITiCSE '14, pages 153–158, NY, USA, 2014. ACM.

7. L. P. Flannery and M. U. Bers. Let's dance the "robot hokey-pokey!" Children's programming approaches and achievement throughout early cognitive development. *Journal of Research on Technology in Education*, 46(1):81–101, 2013.
8. L. P. Flannery, B. Silverman, E. R. Kazakoff, M. U. Bers, P. Bontá, and M. Resnick. Designing ScratchJr: support for early childhood learning through computer programming. In *Proceedings of the 12th International Conference on Interaction Design and Children*, pages 1–10. ACM, 2013.
9. S. Furber. Shut down or restart? The way forward for computing in UK schools. Technical report, The Royal Society, London, 2012.
10. S. Grover, R. Pea, and S. Cooper. Remedying misperceptions of computer science among middle school students. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, pages 343–348, New York, NY, USA, 2014. ACM.
11. E. Kazakoff, A. Sullivan, and M. Bers. The effect of a classroom-based intensive robotics and programming workshop on sequencing ability in early childhood. *Early Childhood Education Journal*, 41(4):245–255, 2013.
12. S. Magnenat, J. Shin, F. Riedo, R. Siegwart, and M. Ben-Ari. Teaching a core CS concept through robotics. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*, ITiCSE '14, pages 315–320, New York, NY, USA, 2014. ACM.
13. C. Martinez, M. Gomez, and L. Benotti. Lessons learned on computer science teachers professional development. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, NY, USA, In press. ACM.
14. C. Martinez, M. J. Gomez, and L. Benotti. A comparison of preschool and elementary school children learning computer science concepts through a multilanguage robot programming platform. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, pages 159–164, New York, NY, USA, 2015. ACM.
15. O. Meerbaum-Salant, M. Armoni, and M. M. Ben-Ari. Learning computer science concepts with Scratch. In *Proceedings of the Sixth International Workshop on Computing Education Research*, ICER '10, pages 69–76, NY, USA, 2010. ACM.
16. L. Morgado, M. Cruz, and K. Kahn. Preschool cookbook of computer programming topics. *Australasian Journal of Educational Technology*, 26(3):309–326, 2010.
17. L. Morgado and K. Kahn. Towards a specification of the ToonTalk language. *Journal of Visual Languages and Computing*, 19(5):574–597, Oct. 2008.
18. K. Perdikuri. Students' experiences from the use of MIT app inventor in classroom. In *Proceedings of the 18th Panhellenic Conference on Informatics*, PCI '14, pages 41:1–41:6, New York, NY, USA, 2014. ACM.
19. D. Portelance, A. Strawhacker, and M. U. Bers. Constructing the scratchjr programming language in the early childhood classroom. *International Journal of Technology and Design Education*, pages 1–16, 2015.
20. C. Wilson. *Running the Empty: Failure to Teach K-12 Computer Science in the Digital Age.* Association for Computing Machinery, 2010.
21. C. Wilson. Hour of code: We can solve the diversity problem in computer science. *ACM Inroads*, 5(4):22–22, Dec. 2014.
22. I. Zur Bargury. A new curriculum for junior-high in computer science. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, pages 204–208, 2012.