Contents lists available at ScienceDirect

Computers in Human Behavior

journal homepage: www.elsevier.com/locate/comphumbeh

Full length article

Phogo: A low cost, free and “maker” revisit to Logo

Pablo Molins-Ruano[*], Carlos Gonzalez-Sacristan, Carlos Garcia-Saura

Dpto. Ingeniería Informática, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 28049 Campus de Cantoblanco, Madrid, Spain

A R T I C L E I N F O

Article history:
Received 31 March 2017
Received in revised form
5 September 2017
Accepted 23 September 2017
Available online 27 September 2017

Keywords:
Computational thinking
Technology education
Educational robots
LOGO
Pre-university education

A B S T R A C T

Today it is almost impossible to spend a single day without depending on an information system, a computer or any other form of computation. Though the starting barrier is low, fundamental concepts are still required in order to manage the technicalities of the engineering environment and everyday computational systems. In 1967, Logo proposed to teach abstract programming concepts by providing a set of functions that had intuitive, visible effects over a robotic *Turtle*. LOGO was a success, but the robots quickly migrated into computer simulations. From LOGO, many followed. Scratch and Lego Mindstorm are some of the most notorious examples. Both introduced graphical block-based programming interfaces. We propose to bring back the powerful ideas behind LOGO by updating it with state of the art technologies. *Phogo* combines Python, Arduino and 3D printing into a low cost robot that is easy to build and control. The robot has a pen to draw shapes and can be commanded from a computer via a wireless link that is transparent to the students. The use of a physical robot can make programming more accessible for students with disabilities. The open and maker philosophies behind Phogo makes it more interesting as students will be able to access and study the electronic components. The textual programing language can be a long life companion for the students. In this work we discuss LOGO and other projects inspired by it, and we also share the methodology and design decisions behind Phogo, the results of its application in a workshop and the improvements we are currently developing.

1. Introduction

Today it is almost impossible to spend a single day without depending on an information system, a computer or any other form of computation. There are multiple examples of how the world is being changed by the Digital Revolution: the media industry (Manovich, 2013), medicine (Murdoch & Detsky, 2013) or transport (Cramer & Krueger, 2016), to name a few. We all need some computational understanding in order to comprehend our modern social context (Rushkoff, 2010), so any education that aims to empower the learner must acknowledge and consider the digital reality we live in. The benefits of programming-based education are widely studied and documented (García-Peñalvo, 2016; García-Peñalvo, Reimann, Tuul, Rees, & Jormanainen, 2016; Paliokas, Arapidis, & Mpimpitsos, 2013, pp. 115–131; Sáez-López, Román-González, & Vázquez-Cano, 2016) although there are still some open questions as whether the positive impact can be reproduced in all the subjects and areas, not just math or robotics (Gomes & Brito Correia, 2014; Moreno-León, Robles, & Román-González, 2016a). In this context, many studies have shown that presenting students with a robot as the study subject can help to grab their attention, and thus facilitate their learning experience (Benitti, 2012; Burbaitė, Damaševičius, & Štuikys, 2013). As such, teaching platforms designed to develop computational thinking are indeed a valuable pedagogical resource.

In the 1970s, Seymour Papert, Cynthia Solomon and many more at the M.I.T. Artificial Intelligence Laboratory developed a system composed of a mobile robot (the Turtle) controlled with a programming language derived from Lisp. The project was named LOGO, and its ambition was to completely revolutionize education. For many years the team at M.I.T. developed different robot “turtles”, and tested many activities and approaches. At the time, it was an expensive and complex enterprise.

Continuing the tradition, our EECS department at Universidad Autónoma de Madrid has been teaching robotics for almost 20 years, and many changes have occurred in this time. For instance many workshops from 1997 to 2003 used the TRITT micro-robot designed by a student branch of IEEE at Universidad Politécnica de Madrid (González, González, & Gómez-Arribas, 2003). These

* Corresponding author.
E-mail addresses: pablo.molins@uam.es (P. Molins-Ruano), carlos.gonzalez@estudiante.uam.es (C. Gonzalez-Sacristan), carlos.garciasaura@uam.es (C. Garcia-Saura).

https://doi.org/10.1016/j.chb.2017.09.029
0747-5632/© 2017 Elsevier Ltd. All rights reserved.

robots were made of Lego parts and incorporated basic sensing and actuating capabilities, features that made them ideal for education.

TRITT was also used in many workshops at Universidad Autónoma de Madrid, and in 2004 a group of students presented an evolution of the design: Skybot. This platform was more affordable, easy to build and program, and also incorporated more sensors (González-Gómez & Torres, 2005). More importantly, the Skybot soon became commercially available as a kit through a student-led spin-off company, which allowed its use in many engineering courses and robotic events.

The “Maker” movement was born in recent years together with many other platforms and technologies such as Arduino[1] & 3D printers.[2] This movement has changed the way people learn to design 3D parts and to use electronics. More importantly, it has changed how people learn to program. Arduino made it simple to learn and program electronics and to develop robotics projects. RepRap launched the 3D prototyping technology revolution that made 3D printers affordable. These two projects conform a very powerful ecosystem that has redefined the field of robotics and engineering education forever (Wong, 2011). In many cases, *Makers* complement their knowledge with engineering education, but this is not always the case. These are people of any age that, using the Internet as a nexus, come together as a community where every member can learn to build and repair their own gadgets and electronic devices. These *maker* communities and their demand for electronics have drastically lowered, in just a few years, the cost of basic components that used to be very expensive (i.e. microcontrollers, servomotors, sensors, etc).

At our robotics society, *Club de Robótica-Mecatrónica*, we were fascinated by these technologies and started incorporating them into every new design. In 2011 the Skybot design evolved into a 3D-printed version called Miniskybot (Gonzalez-Gomez, Valero-Gomez, Prieto-Moreno, & Abderrahim, 2012). Arduino was incorporated into our introductory workshops with the HKTR-9000 and ArduSkybot robots, with good reception from students (García-Saura & González-Gómez, 2012).

More recently, in 2016 the Office for Inclusion of Disabled People at Universiad Autónoma de Madrid asked us to give a robotics workshop to a group of high school students in risk of dropping out of higher education. The activity was framed within the national program “Campus Inclusivo, Campus sin Límites” a national initiative designed to contribute to reduce early drop out rate of students with disabilities, encouraging young people with disabilities from the second cycle of compulsory secondary education and high school and higher education to continue their education towards the University in order to facilitate their access to quality employment in the future.

We decided it was a great opportunity to improve previous designs by combining Python, LOGO and 3D printing into a low cost robotic platform, as the introduction of brief workshops have been successfully used to increase the students interest in STEM areas (Mohr-Schroeder et al., 2014) and to reduce the absenteeism during the next school year (Iver & Iver, 2015).

In this paper we present Phogo, a teaching platform based on a low cost 3D-printed robot capable of tracing its path with a marker, as well as a software infrastructure that allows for transparent wireless communication with each robot. All the software and hardware, as well as the teaching materials, have been published with open licenses and are freely available in a GitHub repository.

Phogo is a project in an early development stage, aimed to become a platform for learning and teaching programing concepts to teenagers, young adults and beyond. The final objective is to engage pre-university students towards STEM careers. We believe that Phogo has four strengths: a) simplicity and low cost, to reach schools, b) a free and open conception (based on a “maker” approach) that will allow students to assemble and modify their own robots, c) the use of Python as the main language, compared to other projects with arcane or less interesting languages for teaching, and d) a tangible entity that can facilitate learning for students with special needs, as it gives them more opportunities.

The following section reviews the main pieces of the LOGO project and some other projects that followed. In the third section there is a description of our Phogo robot's mechanic and electronic parts, the Python-based software and the possibilities of the combined system are presented. The introduction to Phogo is followed by a comparison of the previous projects with Phogo. Then comes a preliminary validation of a Phogo prototype in an informal experience with 19 students with special needs and no previous programming knowledge. Last, we show our current work improving the robot and the library based on that experience. Finally, our conclusions are presented, followed by the discussion and future work.

2. Previous and related works

In this section we provide an overview of LOGO and two of its daughter projects: Lego Mindstorms and Scratch. Later, in section 4, a final comparison between LOGO, Mindstorms, Scratch and our approach, Phogo, will be presented.

2.1. LOGO

One of the first successful examples of developing computational thinking with robots is the LOGO project by MIT (Papert, 1971a). LOGO started in the 1970s as a programming language (“baby Lisp”) combined with a robot, known as the Turtle (Papert, 1971b). The project was developed together with the Constructionism pedagogical theory, where students are believed to learn while being consciously engaged in constructing a public entity (Papert & Harel, 1991). As such, LOGO is a great tool that provides construction possibilities to the student while also preventing self-doubts about her or his capacities: when something does not come out as expected, the Turtle is the one making mistakes, not the student (Papert, 1971a).

As LOGO has been credited as the starting point and inspiration for almost every following attempt to develop computational thinking, we will review its three more important components: its vision (Constructionism), the language (LOGO) and the tool (the Turtle).

2.1.1. The vision: constructionism

Although the LOGO project started within the Artificial Intelligence Laboratory at M.I.T. it was always oriented to education. The project started as a research on Elementary Education with the idea of progressing to upper educational stages, with no upper limit (Papert, 1973). Papert, the father of constructionism, introduced it as follows:

> “*The word constructionism is a mnemonic for two aspects of the theory of science education underlying this project. From constructivist theories of psychology we take a view of learning as a reconstruction rather than as a transmission of knowledge. Then we extend the idea of manipulative materials to the idea that learning is most effective when part of an activity the learner experiences as constructing a meaningful product.*”

[1] http://www.arduino.cc/.
[2] http://www.reprap.org/.

Seymour Papert worked with Jean Piaget, the father of constructivism (Piaget, 1952), at the University of Geneva from 1959 to 1963. Both theories confronted the previous and still prevalent pedagogical method in which the student is considered as some kind of jar to be filled with knowledge provided by the teacher. A master class where a teacher lectures a group of students is the best example of what they fought fervently.

Both constructionism and constructivism consider that students learn by building their own mental models. Learning occurs when the learner experiences new information and that information is introduced into prior models or old models are changed to accommodate the new information. The student becomes an active component of the learning process.

Constructionism differs from constructivism in the importance that the eager gives to the construction of real social entities. Papert considered that it is required to build a physical object to achieve the smoothest learning possible. Learning is also improved when it occurs within a social environment where students help each other to understand concepts. Students are encouraged by the teacher to explore their own ideas and discover knowledge by trial and error or by constructing. The teacher is no longer expected to give a detailed explanation or a sequence of instructions, but to become a catalyst for knowledge.

This theory is not exempt of criticism. On one hand, Papert glimpsed a future where all subjects where approached following these ideas, but the suitability beyond STEM subjects is in question (Benitti, 2012). LOGO researchers also investigated music teaching (Bamberger, 1974, 1979) and writing (Lawler, 1980), but none of them were systematic and, therefore, not conclusive. On the other hand, one of the most important critics is the lack of evidence to support this family of pedagogical theories, and when there is, it may point to a worse performance in the students, specially when they acquire misconceptions (Kirschner, Sweller, & Clark, 2006).

2.1.2. The language: LOGO

LOGO was designed with the principle of "low beginning threshold and no ceiling," i.e., it's easy enough for novices but powerful for experienced users (Papert, 1971b). It was a *Lisp* language derivate, but does not use parenthesis to delimit statements. Instead, it uses new lines and does not use prefix notation for mathematical operations. This makes LOGO easier to read for a novice. Many different implementations existed and exist today, with a wide range of approaches as LOGO was not completely standardized. LLOGO, UCB Logo, MSW Logo, Terrapin Logo, etc. are some examples.

One of the most characteristic features of LOGO is its interactivity, allowing students to have immediate feedback, which makes the debugging and learning process easier, as error messages are displayed after executing each instruction. These error messages are quite descriptive. For example, if the user types `fowad` (instead of `forward`) the LOGO interpreter will print `I don't know how to fowad`. Then, the user will probably write `forward` and the interpreter will print `Not enough inputs to forward as forward requires a numerical distance to travel as input`. When the user finally writes `forward 20` the turtle then moves 20 steps. The LOGO language requires a Turtle, physical or on screen, to receive the instructions. We will go into more detail on the Turtle in next section.

The basic primitives of LOGO are those related to the movement of the Turtle:

- `forward n`: the Turtle moves n steps in the direction it is facing.
- `right d/left d`: the Turtle turns d degrees, clockwise or counterclockwise.
- `back n`: the Turtle moves n steps in the opposite direction to the one it is facing.
- `penup/pendown`: The Turtle has an instrument to mark its path. With pendown the next path of the Turtle will be marked, and with penup the drawing will stop.

LOGO allows the user to define new procedures. To do so, the student has to explain how to do the new procedure. For example, to move the Turtle in a complete circle you could write:

```
TO circle :degrees
  IF :degrees = 0 STOP
  forward 1
  right 1
  circle :degrees - 1
END

circle 360
```

Once a procedure is defined, it can be used as any other primitive of the language. Procedures also allow to specify inputs, preceded by a colon. The language has many other primitives to control all the sensors and actuators of the Turtle (i.e., a light sensor, a speaker, etc.) but as all of them differ between implementations -they depend on the characteristics of the physical or simulated Turtle- we will not discuss them here.

2.1.3. The tool: the Turtle

According to constructionism, students are required to create some sort of entity in order to understand and build their knowledge. From the very beginning, LOGO was designed with the Turtle in mind (see Fig. 1). The Turtle started as a physical robot that was able to move freely on a flat surface while tethered to a computer. The entity Papert had in mind was not the robot itself, but what the students built and explored with it. Students received the robot already assembled and ready to go. The original Turtles were

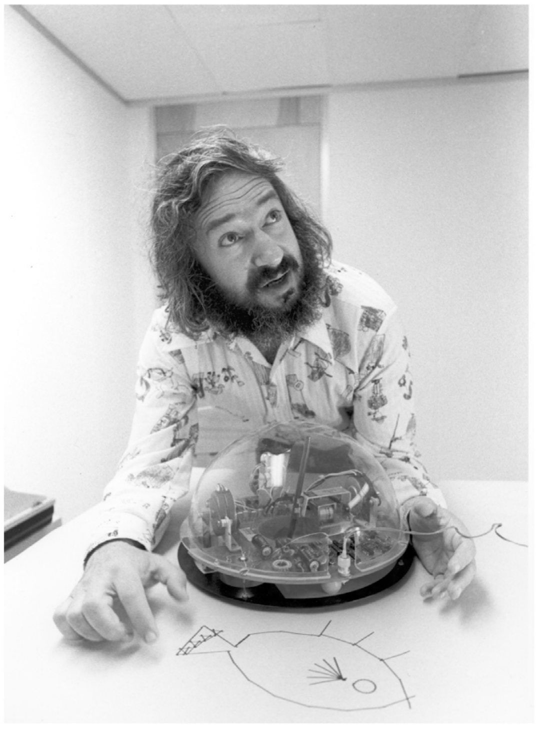


Fig. 1. Seymour Papert with one *Turtle*.

equipped with a pen to mark their path on the floor, and also with a light sensor, a speaker and two stepper motors. More actuators were also used in later Turtle builds (i.e. accelerometers, tilt sensors, touch sensors, etc.). The name "Turtle" was a reference to cybernetic turtles originally built by English neurophysiologist Grey Walter to mimic life-like behavior (Papert and Solomon, 1976).

LOGO's Turtle was eventually transformed from a real robot into a virtual instance, a graphical representation on the screen. At a time where motors and actuators were expensive, removing the robot was cheaper and easier to implement in schools or homes. This change also yielded new possibilities as it allowed to "teleport" the Turtle from one point of the screen to another or to fade away the marked path after some time defined by the user.

2.1.4. *LOGO history and results*

The first version of LOGO was created in 1967. For many years LOGO was used in different settings and experiments, so nowadays we have a good idea of what original LOGO promises became a reality.

Douglas H. Clements and Julie S. Meredith revised the literature related to LOGO and its effects over the students that used it (Clements & Meredith, 1993) (Clements & Sarama, 1997). They reviewed the effects of LOGO in many different areas. Some of them, such as Mathematics, were more easy to test. Students using LOGO to learn Geometry, Variables and Algebra clearly showed a good grip of the notions related to those areas. Though it was not clear whether the improvement was due to the use of LOGO. Some studies concluded positively (Dog, 1985) while others presented mixed results (Howe, O'Shea, & Plane, 1980) or even showed no differences at all (Battista & Clements, 1986).

The authors also looked for results in other areas: Problem-Solving abilities and Language. In these areas, many of the results were again inconclusive. Many experiments have been conducted over these 30 years, and though there was a sense of improvement when using LOGO for education, this is not conclusive at all. Clements and Meredith pointed to some reasons (Clements & Sarama, 1997):

1. LOGO was created as a piece for an educational reform, not as a standalone entity to be integrated in the existing system. As so, it is not easy to pull apart all the variables in a controlled environment, and previous ways of measuring improvement were not adequate. Because LOGO is an open tool, the teachers needed to chose what to do with it and how to present it, difficulting a fair comparison between different settings.
2. The LOGO benefits are maximized when experiences go beyond mere exposure, when there is a proper classroom culture and an active constructivist approach is chosen for the teaching/learning process. Not all this circumstances were coexisting in all the experiments.
3. As Clements and Meredith said: "While educational researchers debate the efficacy of various research methods, we conclude that there is no single best method for assessing the effects of Logo. Each has advantages—a certain lens that allows us to view people as they use Logo. Each has blind spots."
4. The benefits of LOGO are not impressive on each area, but are quite remarkable when seen as a whole. It was not an incredibly revolutionary mathematical learning method, but "mediated Logo environments are interesting in that they seem to enrich so many different aspects of students' lives. An alternate, narrow approach might yield similar gains on a single test, but few educational environments have shown consistent benefits of such a wide scope, from the mathematical and cognitive to the social and emotional." This wide scope was not easy to detect in single and isolated experiments.

Which different aspects were truly enriched? LOGO was quite promising in some areas such as the social and emotional development of the students. With LOGO students engaged in group problem solving and sharing, specially important results for the usually socially isolated students (Carmichael, 1985; Dog, 1985). Students get involved in more conflicts (as there is more interaction) but they resolve those conflicts (Clements & Nastasi, 1985) and become more involved in helping and teaching others (Dog, 1985; Hawkins, Sheingold, Gearhart, & Berger, 1982). According to the teachers, students increased their self-stem, but only when they are left greater autonomy and social interaction is fostered (Carmichael, 1985; Dog, 1985).

LOGO was also promising for students in risk of staying behind, either by social or by medical reasons. LOGO did improve the self perception of children with special needs, as they received more prestige and respect from their colleagues (Michayluk, Saklofske, & Yackulic, 1984). It also improved the situation of girls and their self-esteem, something very important in order to correct the troubling gender ratio within the STEM professions. In one study, the differences between two female groups (one using LOGO and the other as control) widened notably, and the group that was using LOGO overtook the control male group that had started ahead (Howe, Ross, Johnson, Plane, & Inglis, 1982). LOGO has also been reported to improve the internal feelings of personal responsibility and success in females (Olson, 1985).

One way or another, LOGO was displaced due to multiple and coexisting reasons. On one hand, the technology has improved and lowered its cost, making it possible to be widely adopted in many jobs and schools. But it has also reduced the application of the computer as a programming tool, as many of the new users limit their use to word processing, spreadsheets, games, web browsing, etc. On the other hand, LOGO presented some problems that were subsequently enumerated by the creators of Scratch (Resnick et al., 2009), already in the 21st century:

- The syntax of the languages used were difficult for children.
- Programming was usually introduced with activities that were not interesting for the students (i.e. generating lists of prime numbers). The outcome of the learning process was not significant for the students.
- The context of learning was solitaire, where nobody was able to offer support when something went wrong.

2.2. *LOGO … and beyond*

LOGO was a foundation stone, and from it many other projects followed (Chao, 2016; Gomes & Brito Correia, 2014; Paliokas et al., 2013, pp. 115–131). LOGO versions were developed for personal computers; the toy company Lego created some robots, and nowadays we have Scratch, a modern language oriented to young students and based on blocks (Resnick et al., 2009; Sáez-López et al., 2016). In this section Lego Mindstorms and Scratch are reviewed.

2.2.1. *Lego Mindstorms*

In 1988 a collaboration was created between MIT and the Lego Group to design an intelligent brick that could be programmed by a computer: The Lego Mindstorms. The bricks and their components were modular and could be used to build many different robotic devices. Each brick was also programmed using a modular, block-based language that was simple and intuitive to use. The Mindstorms platform has proven to be a valuable educational tool in STEM (Grandi, Falconi, & Melchiorri, 2014; Klassner & Schafer, 2014), specially as it combines real world robots with a software component hosted in a computer (Danahy et al., 2014).

Since the inception of Lego Mindstorms, different iterations were released in 1998, 2006, 2009 and 2013. The latest version is referred to as *Lego Mindstorms NXT EV3* and includes: a Programmable Brick, a color sensor, a touch sensor, a remote infrared beacon (also used as a remote control for the robot), an infrared sensor, two big-sized motors and one medium-sized motor in its most basic configuration. Other sensors, such as gyroscopes and ultrasound sensors are also available. All of these sensors and actuators share a common proprietary cable interface and are controlled by the Programmable Brick. This makes the installation of new components really easy. The Programmable Brick has four possible connections to other components, as well as USB and Bluetooth to connect to the computer.

There are programming interfaces to command the Mindstorms bricks with C or Java, but the official programming application is available for iPad and Android tablets and PC/Mac (as seen in Fig. 2a). The tablet version has a reduced set of blocks compared to the desktop versions. The blocks can be dragged and dropped on a canvas, and once they are connected they make a program. The blocks are divided in different categories:

- **Action blocks** control motor rotations and also images, sounds and lights on the Programmable Brick LCD.
- **Flow blocks** control the flow of the program. All programs must start with one of these blocks. There can be loops and conditional blocks.
- **Sensor blocks** read the input from each of the different sensors.
- **Data operation blocks** write and read variables, compare values, perform mathematical operations or generate random numbers. These blocks are only available in the desktop version.
- **Advanced blocks** control the Bluetooth connection, read and write files and perform other similar operations. These blocks are also only available in the desktop version.

The Lego Mindstorms series are easy and intuitive to use. All the components work out of the box without requiring any complicated configurations or assemblies. Robots can be built using regular Lego pieces, and the graphical programming interface is intuitive and easy to use. The two major downsides of Lego Mindstorms are the elevated cost and the use of proprietary components.

The cost for an introductory kit is around 350 USD and new sensors cost around 40 USD for the gyroscope or color sensor. This can be an excessive investment, specially for schools where more than one robot is required for them to be introduced in the classrooms.

Proprietary components make it really easy to plug new components, but limit the design possibilities to those options implemented by Lego and effectively hides all the components inside a black box. Curious students are forbidden to look inside the box and cannot understand, experiment or enhance the physical components that are included with the robot.

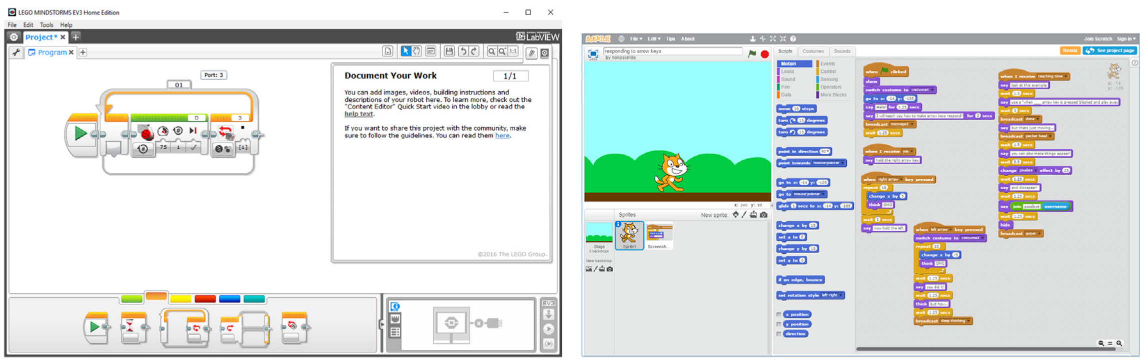
2.2.2. *Scratch*

Scratch is a free educational programming language developed by the Lifelong Kindergarten Group (part of the MIT Media Lab). Like Lego Mindstorms, Scratch is also based on a block programming interface, and instead of using a physical robot it allows to draw shapes and animations to the screen. The benefits of using Scratch have been thoroughly documented, particularly when applied to teaching programming concepts (Meerbaum-Salant, Armoni, & Ben-Ari, 2013) and even for accelerating the learning curves in several school subjects such as Mathematics and Social Sciences (Moreno-León et al., 2016a). The first version was released in 2005 and the second (and current) version was released in 2013.

The drag-and-drop interface is shared with what is known as the stage area. It can be seen on the top left of Fig. 2b. The stage area is where the results of the script are shown, similarly to the virtual Turtle of some LOGO implementations. The main difference with the virtual Turtle is that while the Turtle must move in order to draw, in Scratch it is possible (and even encouraged) to import images or photos to the stage. Once they are imported, they can be manipulated by the script using the blocks. For each imported image there is a different script area, that can be accessed by selecting in the sprite list each one of them. Therefore, the animation of each element is easily distinguished.

There are many blocks available:

- **Motion**: For moving or turning each sprite. They can be moved using relative coordinates (i.e., move 10 steps) or using absolute coordinates, where the origin is in the center of the stage area.
- **Looks**: Each sprite may have different costumes associated, so it is possible to make animations. In this group there are also blocks that can add speech bubbles to the sprites.
- **Sounds**: Students are able to add their own recordings and sounds, so it is also possible to play different instruments and notes.
- **Pen**: It works like the classic Turtle from LOGO, but it includes color options.
- **Data**: For declaring and using variables and lists.
- **Events**: Blocks that are activated by clicks on buttons, keys, microphone, camera or passing messages between the different sprites.

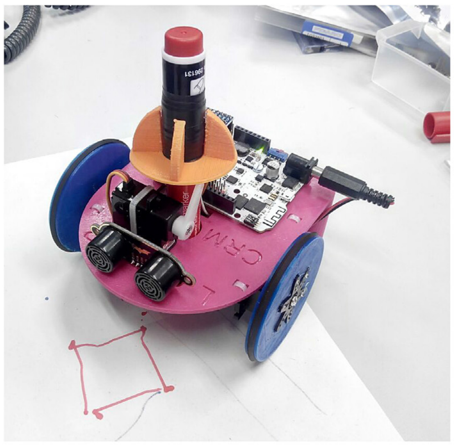


You can add images, videos, building instructions and descriptions of your robot here. To learn more, check out the "Content Editor" Quick Start video in the lobby or read the help text.

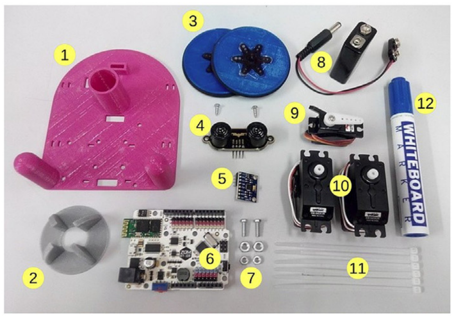
If you want to share this project with the community, make sure to follow the guidelines. You can read them here.

(a) Mindstorms programming interface. (b) Scratch interface.

Fig. 2. Different LOGO derivatives.



(a) A finished Tortoise.



(b) Tortoises are made of parts which are widely available: 1) 3D-printed chassis, 2) 3D-printed pen holder, 3) 3D-printed wheels, 4) distance sensor, 5) gyroscope, 6) controller board, 7) screws and spacers, 8) 9V battery, 9) servo for pen lifting, 10) main servo motors, 11) zip-ties and 12) dry-wipe marker pen.

Fig. 3. First version of the Tortoise robot used in Phogo.

- **Control**: Loops and conditional structures.
- **Sensing**: Different "sensors" like blocks that will activate when the mouse is in some position or when it touches a color.
- **Operators**: Mathematical operations, comparisons, boolean operators and random number generation.
- **More blocks**: It is also possible to define new blocks and to add extensions to connect to other hardware.

Scratch has as the motto "Imagine, Program, Share" and according to the Scratch official wiki page: *"This follows the basic principle of creating a Scratch Project. First of all, you think of an idea (imagine), next, you program your idea in Scratch (program), then finally share it with the world (share)."*[3] Scratch has a repository of community projects and the students are encouraged to try, comment and remix projects of other users, and these social interactions seem to improve the learning of the students (Moreno-León, Robles, & Román-González, 2016b).

Scratch has a few problems. On one hand, it is developed using Adobe Flash, a technology that is not as widely supported by web browsers as it was a few years ago. There was an attempt to port Scratch to HTML5 but is not longer developed by the Scratch team, as all efforts are focused on Scratch version 3 (which is expected to enter in alpha phase by the end of 2017 and will remove the Flash requirement). On the other hand, although it is a great tool for new young learners, the design and default sprites of Scratch might be considered infantile and could discourage high school or college students from using it.

3. Phogo: robots that can draw for students who can learn

As the costs of electronic components keep dropping, and thanks to Arduino, 3D printers, as well as some other platforms and technologies, it is now possible to easily prototype small mobile robots (García-Saura & González-Gómez, 2012). We have the opportunity to bring back the original idea of LOGO with the physical Turtle robot.

The success of LOGO's approach has its roots in three basic components: an interactive entity that can be physical or digital, simple methods for interaction and students with motivation. Our proposal follows the same principles, but uses a completely renovated implementation: the interactive entity is a robot named "Tortoise" (rather than LOGO's "Turtle"), and the methods for interaction are high-level Python commands (rather than a restricted Lisp derivative). The Tortoise and the library to control it have been designed with low cost and ease of use in mind. A finished tortoise is shown in Fig. 3a.

The following sections will discuss the two main components of Phogo: An Arduino-based Tortoise robot, and the Python control library.

3.1. The Tortoise

Thanks to Arduino, nowadays it is relatively simple and cheap to build a robot. Components, motors, sensors and actuators are easy to find, and again, they have a relatively low cost. With domestic 3D printing becoming widespread, and with the on-line *Maker* community as a support, robots can be easily put together at home.

The Tortoise was designed to give multiple options to the students. It is a small wheeled robot with two motors to move around and a distance sensor in the front. Most importantly, the Tortoise also has a programmable pen that can draw at will on top of the surface where it is moving on.

We faced the problem of achieving a precise motion of the robots while maintaining simple control commands and low cost parts (i.e., at first the robots did not follow a straight path and did not turn accurately, yielding distorted drawings). At the end we decided to use a small gyroscope connected to the Arduino board, removing the need of manually calibrating each robot (García-Saura, 2015).

The controller board that was selected is a bq Zum, an Arduino derivative. We decided to use it mainly for its built-in Bluetooth connectivity and ease of motor connection. The communication between the computer and the robot is transparent, so when the robot is equipped with a standard 9 V battery it has no restricting wires and can move freely.

The design is simple enough to be put together by anyone with basic electronic skills (see Fig. 3). It is also very affordable, with a build cost of around 81 USD per robot (plus the material needed for the 3D-printed parts). We have shared an assembly guide in the GitHub repository of the project[4]

[3] https://wiki.scratch.mit.edu/wiki/Scratch_Program/#Motto. Accessed March 2017.

[4] https://github.com/CRM-UAM/Phogo/wiki/Assembly-guide.

3.2. The library

As discussed earlier, Python was chosen as the base programming language for Phogo. All the code has been encapsulated into a Python module. Doing so has allowed us to have a transparent setup for the connection between the computer and the Tortoise robot. The "magic" happens in the __init__ file of the phogo Python module, which determines which user controls each Tortoise. The user is automatically connected to its assigned Tortoise by simply issuing the following command: `from phogo import *`. Once the connection is set up, the rest of the code is executed and motions are performed by the connected Tortoise. The functions available to the students are similar to the ones in LOGO:

1. `forward(units = 10)`: Moves the Tortoise forward.
2. `back(units = 10)`: Moves the Tortoise backward.
3. `right(degrees = 90)`: Turns the Tortoise clockwise. The marker pen is located in the center of the rotation, allowing to draw sharp corners.
4. `left(degrees = 90)`: Turns the Tortoise counterclockwise.
5. `pen_up()`: The pen is lifted up, so no trace is left on the surface under the robot.
6. `pen_down()`: The pen is released, so when the robot moves it draws a trace in the floor.
7. `obstacle()`: The Tortoise polls its front distance sensor, and returns the distance to the nearest obstacle, in centimeters.

As some code examples we show a function to draw a square and a function to solve a maze:

```
from phogo import *

def square():
  pen_down()
  i = 1
  while (i < 5):
    right()
    forward()
    i = i + 1
```

```
from phogo import *

def right-hand_rule():
  if (obstacle() < 10):
    left()
  forward()
  right-hand_rule()
```

The communication protocol between the computer and the Tortoise, though transparent to the students, has also been kept simple: Each serial command to the Arduino control board is formed by two letters plus an optional argument. e.g., `pen_up()` ⇒ `PU`, `forward(60)` ⇒ `FD 60`. The Tortoise always answers back once the command finishes, whether it is with a simple OK, or with a value (e.g., `obstacle()` ⇒ `OE` returns the distance in centimeters to the obstacle in front of the Tortoise: `OE` → `130`). These communications are logged in the console, so advanced students can see how their code is converted into sequential control commands that the Tortoise follows.

4. The novelty of Phogo. A comparison

In the previous section we have presented Phogo and its two main components: The Tortoise (hardware) and the Library (software). We have also presented the relevant previous work that inspired this project. Now we want to introduce a comparison between Phogo and each of the three "parent" projects: LOGO, Lego Mindstorms and Scratch. The three "parents" are compared in that order to Phogo with a final table summarizing (see Table 1).

4.1. LOGO and Phogo

Phogo takes from LOGO the principle of "low threshold and no ceiling", i.e., being easy enough for novices but powerful for experienced users (Papert, 1971b). We create this low floor by giving the students very simple and direct functions to control the Tortoise, allowing them to achieve something attractive, fast and easy, thus increasing their motivation.

In order to eliminate the learning ceiling we decided to choose Python as the programming language and move away from LOGO, as it is no longer a modern language for todays standards. Python is simple and powerful and can be a life-long companion for new students, whereas LOGO probably can't as:

1. **LOGO lacks a thriving community.** Python's vast community of users has advantages: documentation, tutorials and libraries are easily found and will allow the student to continue beyond the initial material provided. For example, in the popular *Stack Overflow* forum, as of April 2017 there are more than 725.000 questions related to Python while there are only 40 regarding LOGO.
2. **Python has real world applications** and we believe this can have an impact on the students' engagement as it will probably motivate them to continue exploring: College courses are using Python, many jobs positions value Python experience, etc.

Table 1
A summary of the different systems compared.

	LOGO	Mindstorm	Scratch	Phogo
Cost	Low	High	Low	Low
Type of programming	Text-based	Block-based	Block-based	Text-based
Maker approach	No	No	No	Yes
Thriving community	No	No	Yes	Yes
Real world application	No	No	No	Yes
Ages	6 and beyond	10 and beyond	8–16	14 and beyond

3. **With Python, nothing is lost.** It is still possible to have a modular, extensible, interactive and flexible language, the main principles in LOGO design.

Linda Mannila and Michael de Raat made a comparison of the different programing languages available for teaching and learning (Mannila & de Raadt, 2006). They choose a list of 17 different criteria based on works by Seymour Papert, Niklaus Wirth (creator of Pascal), Guido van Rossum (creator of Python) and Bertrand Meyer (creator of Eiffel). From these criteria they assigned a mark to each language. They compared C, C++, Eiffel, Haskell, Java, Javascript, LOGO, Pascal, Python, Scheme and Visual Basic. Python was the highest ranked language with a mark of 15/17 while LOGO obtained a 9/17.

All the LOGO points are also in present in Phyton except for one that we consider is covered with Phogo thanks to the Tortoise and the library: "To meet this criterion the 'language' should not only be restricted to implementation, but cover many aspects of the software development process. The 'language' should be designed as an entire methodology for constructing software based on 1) a language and 2) a set of principles, tools and libraries." So according to these criteria Phogo would receive a 16/17.

With Phogo we hope to introduce Python programming to the students and progressively show them all of its possibilities. First, the more basic functions of the Library are presented, but once the students feel confident they may start exploring with other standard Python libraries as those related to input-output, numeric and mathematical operations or string manipulation, as it was done with LOGO (Lawler, 1980). From these standard libraries, they can be encouraged to explore other community Python libraries and frameworks, according to their interests and eventually dropping the need of the physical robot and making it just accessory. We believe that a smooth transition from the Phogo Library to, for example, a *Django*[5] web page is possible with Phogo, but impossible to imagine with LOGO.

Last but not least, the development of code is not limited to the language. All the development tools around the language are as important. Python has plenty of IDE freely available, as debuggers or even frameworks as Django or Jupyter[6] and its notebooks, that also offer promising features for teaching (Dennis et al., 2017; Perez & Granger, 2015).

4.2. Lego Mindstorm and Phogo

The Lego Mindstorm product line is in many ways an improvement to the original LOGO. It introduces a modular robot and a graphical programming interface. The setup is simple and the bricks can be combined with the rest of the Lego bricks. But it has two major disadvantages:

1 **Cost:** The most basic Lego Mindstorm kit costs around 350 USD. It includes various sensors (i.e. color, touch, infrared), a remote controller and three different motors. More sensors at available starting at 40 USD each. This price tag makes it really expensive to introduce them in a classroom. And also, expensive to maintain, as each broken component costs at least another 40 USD. The initial version of Phogo cost around 80 USD per robot, and the new version being developed (presented in section 6) costs less than half. The final cost of a Phogo tortoise can potentially be less than an eighth of the Lego Mindstorm price.

2 **License and black box approach**: While Phogo is built as free software and with open licenses for all the documentation, designs and code, this is not the case for Lego Mindstorms. All the components are privative, as well as the software. This may not be a problem for the student, but it restricts the learning opportunities. In the Phogo environment the student is capable of fully assembling and disassembling the Tortoise, while in Lego Mindstorms all the electronic components are confined inside plastic blocks.

Phogo and Lego Mindstorms share the motto *robot as a physical entity*, but the Tortoise provides a greater learning opportunity in the sense the student is able to see all the electronics behind the hardware. We believe that exposing electronic components is fundamental for the students, as understanding the hardware should also be part of any computational thinking educational program. The hardware is, after all, just as necessary as the software. Phogo can be assembled and repaired by the students themselves, offering a true opportunity to introduce them to the hardware part with a real entity. The use of 3D printed parts in Phogo also allows for some personalization of the robots, as is the case with Lego's bricks, but with the added bonus of introducing 3D printing and other real-world design tools to the students.

4.3. Scratch and Phogo

Scratch and Phogo have common ancestors, but they are distant relatives. Scratch is a graphical programing language that is web based, where the student can program different graphical assets. Phogo is a Python-based platform in which a robot is programmed. Scratch was designed specifically for students between 8 and 16 years old, while Phogo is clearly not adequate for 8 year-olds.

The use of graphical, block-based programming languages presents some advantages compared to traditional text-based languages, as pointed by Weintrop and Wilensky (2015). In a study with high school students that were introduced to Javascript, Java and Snap! (a Scratch derivate with more functional programming), they found that students had less difficulties using the block interface because of the simple shapes and colors, the drag-and-drop composition and the ease to browse the blocks library. But "students also identified drawbacks to the blocks-based programming approach, including issues of authenticity, expressive power, and challenges in authoring larger, more sophisticated programs".

These findings are in accordance to our defense of Python in the previous section comparing LOGO and Python. The lower expressive power and challenges when creating larger programs of the block-based languages effectively mean there is a ceiling to them (a high ceiling, but a ceiling after all). With the use of a textual language, this is not the case. As a penalty, Phogo is more difficult for younger learners, and that is why we think the ideal starting point for Phogo would be 14 and beyond (though the lower age limit is yet to be studied).

In the paper that introduced Scratch, three problems were pointed out related to the previous attempts to create teaching platforms for the development of computational thinking (Resnick et al., 2009):

1 The syntax of the languages used was complicated.
2 Programming was often introduced with activities that were not interesting to the students.
3 The context of the learning was solitaire (nobody was able to offer support when something went wrong).

Scratch managed to reduce these problems, and we believe that Phogo has done it as well. The Python language, as discussed

[5] https://www.djangoproject.com/.
[6] http://jupyter.org/.

before, is a proper one for learning. The physical interactions and activities that Phogo presents with the Tortoise are more interesting for the students than the typical command line interaction that a textual programing environment offers for the novice. A solution for the solitaire context is not necessarily part of Phogo as it is for Scratch, where a community of learners was built around the platform so people could see or mix projects from others. Instead, we believe that with the use of existing Python communities in platforms such as GitHub or Gitlab and StackOverflow the students are not as isolated during their learning process.

We believe that Phogo also has an advantage compared to Scratch when applied to students with disabilities. People with some form of disability may require different adaptations, but we believe that the use of a physical robot and a textual language can make it easier to provide a useful setup.

For example, there is a necessity for new tools for blind students (Thieme, Morrison, Villar, Grayson, & Lindley, 2017). The Scratch programing interface is not an option as it built to depend on visual clues, and the use of Adobe Flash difficults some common adaptations such as making the text bigger or interacting with adaptation software. We have found some attempts to make Scratch more accessible for blind people by using voice commands (Wagner & Gray, 2016). Even if the programing interface was improved, the main output of Scratch is the movement of graphical sprites on the screen, something uninteresting for a blind student. With Phogo all the previous adaptation tools such as screen readers or adapted keyboards do not require any modification to work. The physical robot can be more interesting for the students, as it can be heard or touched, or as the pen could be substituted with an awl to mark the path the robot follows while moving over a suitable material.

To summarize, we believe that Phogo can be better than Scratch for older students and students with disabilities (especially for those with visual impairments). Still, Scratch is a great tool and in the future we want to test if it could be used as an introduction to Phogo. It has been shown that the differences between learning a textual language and learning Scratch are not profound, so it may be helpful for some students to start with a graphical approach (Armoni, Meerbaum-Salant, & Ben-Ari, 2015).

5. Phogo workshop experience

On March 2016 the Office for Inclusion of Disabled People at Universidad Autónoma de Madrid asked us to give a robotics workshop to a group of high school students. Three months later, 19 students from 16 to 18 years old were invited to participate in the first Phogo workshop. We took the opportunity to see how students responded to Phogo in a real environment.

5.1. Set up and objectives

They were a mixed group of men and women. None of them reported to have any previous knowledge about programming or robots at all. It was a very diverse group, as 12 of them were people with functional or intellectual disabilities. There were students with blindness, intellectual disability, deaf mute, motor disabilities and Autism spectrum disorders.

They were accompanied by four programming teachers, a sign language interpreter and some assistants. All of the participants were able to program the robot (on their own or with some kind of assistance) and that was all they were asked to do.

The group of students was participating in “Campus Inclusivos, Campus Sin Límites”, a program developed to encourage them to continue their studies after high school. The Phogo workshop was one of the many activities they attended to during that week. In parallel to this activity, the students were able to attend to a talk and a demonstration about 3D printing. Both activities lasted about 90 min and the students were free to move from one activity to the other at any moment. Offering two different activities and the possibility to move was important to make it accessible as students were able to choose what they prefer according to their interests and abilities.

In this experience we did not collect any personal information or did any formal evaluation. This was the case as we did not want to stress or scare the students with any form of test or evaluation, as for some of them this may be problematic. Furthermore, as they were able to freely move from this workshop to the one related to 3D printing at any moment, any collected data would be difficult to compare, as not all students were exposed to Phogo for the same amount of time. The objective was to motivate the group of students with a fun and rewarding time at the university, and also to see how accessible Phogo was and if they were attracted to use it.

Our main goal was to introduce the group to fundamental programming concepts:

1 Understanding that a computer executes the orders previously given by a programmer.
2 Basic use of variables and pre-defined functions.
3 Understand flow control structures:
  (a) Conditional jumps.
  (b) Deterministic loops.
4 Grouping and reusing code by defining functions.
5 Realize that these notions are the backbone of every complex software out there.

A total of 7 Tortoise robots were built before the workshop by the research team. Each Tortoise and computer had been set up prior to the workshop in order to facilitate a quick initiation. At the beginning of the workshop each team received one Tortoise with a tutorial booklet introducing each of the primitives with examples and with possible problems for the students to explore. This booklet was designed to encourage them to explore all the possibilities of the Tortoises and to make them as independent as possible, as it was also a reference manual.

5.2. The workshop

Once all the students arrived they were explained the two different activities (the Phogo workshop and the 3D printing talk/demonstration) and that they would be able to move freely from one to the other at any time. Once the students decided which activity they wanted to attend, groups of 2–3 students were formed to work with each robot. Each group sat in front of a pre-configured computer paired with their Tortoise, and the workshop started.

They were only verbally instructed on how to move the Tortoise forward. The original plan was to make a longer, more didactic introduction but it was impossible given the excitement of the students once they saw the robot moving at their command. Every group started to work on their own while frantically scanning the guide looking for new commands to try.

All the groups worked on their own, being able to ask any doubts to the programming teachers that were present in the room offering assistance and encouraging with new challenges when the students got stuck with the same set of functions or concepts. The programming teachers were instructed to encourage the students to explore different possible solutions rather than providing directly the right answer.

We stored the code written by the students, but since they were reusing the same source file throughout the workshop, only the latest version was kept.

5.3. *Remarks on the experience*

Students seemed to enjoy the workshop and the 90 min went by without any problem or complain. Of all the students that started in the workshop, just two of them dropped and went into the 3D printing workshop. The majority decided to stay with the robots instead. After the workshop many students expressed they gratitude to the programming teachers and assistants. Three of the students asked how they would be able to do the same in their own computers. We believe that the students enjoyed the workshop and that they got, at least, some basic insight.

The following are real snippets of code implemented by the students:

A vast majority of the students achieved more complex code, using loops, variables and self-defined functions, as in the Code 3 example. They dedicated the majority of their time to draw geometrical figures (see 4b), probably because the first challenge in the guide was to draw a square and probably because it is one of the most immediate things to try. Besides, it is easy and appealing to draw starts and spirals once the polygons are understood, as in Fig. 4a.

Drawing a circle was no simple task, as the robot could not move and turn in a single instruction. Remarkably this was seen as a challenge, and some students explored this possibility (see Figs. 5a and 6). The attempts to draw polygons yielded non-regular and open shapes, in part due to the low cost servo motors used in the construction of the Tortoises. As the lines and vertex show, turns were imperfect and distances were not consistent enough between calls.

Code 1

```
from phogo import *

pen_down()
forward()
right()
forward()
right()
forward()
right()
forward()
right()
```

Code 2

```
from phogo import *

if obstacle() < 10:
  right(180)
  forward()
```

Code 3

```
from phogo import *

def square():
  pen_down()
  times = 1
  while times <= 4:
    forward(6)
    right()
    times = times + 1

def circle():
  pen_down()
  times = 1
  while times <= 18:
    forward(1)
    right(20)
    times = times + 1
```

The first proposed challenge was to draw a square. The naive approach of repeating the same functions calls four times was quite common, as in Code 1, but the majority of students tried more complex control structures, as in Code 3. Some students finished the session by trying the distance sensor (as in Code 2), but the majority of them completely ignored it. We believe it is nonetheless an interesting sensor to present and motivate the use of conditional jumps. We will try to make it more attractive in future workshops, giving it more importance in the tutorial booklet (as the ultrasound sensor was only explained in the last page). It could also help to provide the students with a physical maze so they can test the sensor in the context of labyrinth solving.

There is a very interesting drawing in Fig. 5b. Prof. Papert narrated in the first LOGO memo (Papert, 1971a) that a common situation is when a student tries to draw a triangle but produces a hexagon. He explains the problem with a diagram of how, depending on the direction of rotation, the same angle has to be expressed using different angular values (as the robot may need to turn the internal or external angle). That same diagram has been used in our workshop by a programming teacher, without knowing it beforehand! It can be seen on last figure on the bottom. It turns

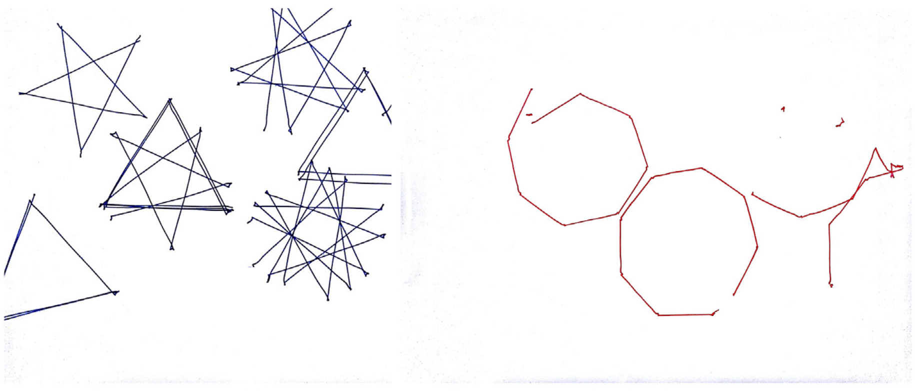

(a) Different stars programmed by a student. (b) Many students tried regular polygons.

Fig. 4. Some examples of the drawings made by the students commands.

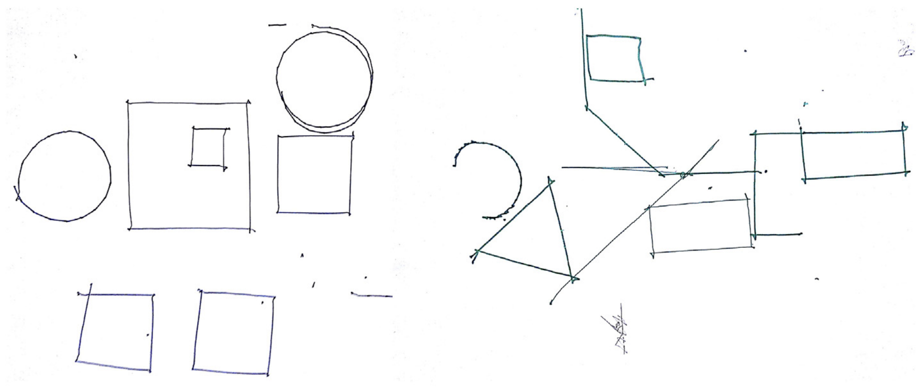

(a) Drawing a circle was no simple task.
(b) Some relevant clarification notes by the teachers.

Fig. 5. More example programs from the workshop.

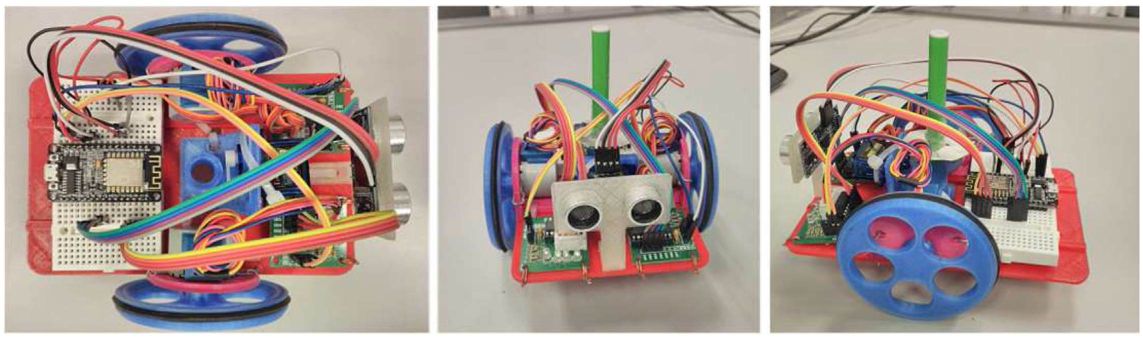


Fig. 6. Phogo's Tortoise version 2 prototype. The pictures show a general view of the new hardware, commanded by an ESP8266 wireless microcontroller.

out that today's students face the same fundamental problems as in the 1970s.

6. Improvements to the Tortoise and library

After the workshop we focused on some possible improvements that we consider can be helpful to future workshops, based on our experience:

1. The Tortoises were not accurate enough for good drawings, mainly because of the servo motors. This was a distraction and a frustration for most of the students as they spent some time trying to adjust their commands to make the exact figures they were expecting.
2. All the software dependencies needed to run our library made it somewhat tedious to set up Phogo in new computers. It could be too complicated for the students to use such a system on their homes.
3. As we did not store every modification made by the students to their source files, now it is not possible to make a non intrusive analysis of their progress.
4. The price of the hardware could be lower. The lower the cost, the more universally accessible it will become, which is one of our main objectives.
5. More sensors and actuators should be built in: speakers, light sensor, etc.

With these objectives in mind we started to work on a second iteration of Phogo (see 6), making improvements to the robot Tortoise and with a complete new approach for the software (while keeping the well-tested public user interface).

The Tortoise now uses a NodeMCU (development board that includes an ESP8266 chip) as the control unit, instead of the original bq Zum. The ESP8266 has a price of less than 3 USD and includes Wi-Fi connectivity. We will probably use the newer ESP32 (the next version of the ESP8266) in the final Tortoises, as it has more memory, a better CPU, more GPIOs (General Purpose Input Output pins), and a Bluetooth connection plus the Wi-Fi interface while keeping a similar price. The servo motors + gyroscope used for the movement of the Tortoise have been replaced with stepper motors that give a better precision to the movement of the robot.

The change from Bluetooth to Wi-Fi makes the Tortoise more accessible in two ways. On one hand, it is uncommon for a computer to have Bluetooth but not Wi-Fi (but not the other way around). On the other hand, using Wi-Fi as the main connection interface allows us to take a complete different approach to the software library. The ESP microcontroller has enough computational power to run a web server and control the robot, so the new Tortoise can either create its own Wi-Fi access point or connect to an existing one. Then students can easily connect and access the Tortoise's web interface through any web browser. From the web interface students have access to a code editor and a terminal where information and error messages are displayed.

With this new approach the setup required by the student is reduced to a minimum. It is only required to connect to a Wi-Fi network created by the Tortoise, and access to a web page hosted by the robot itself. This way we can avoid complicated network configurations and the installation of other software dependencies, making usability as immediate as possible. Students are able to save and restore the code written in the web interface to their own computers, and we also plan to introduce a non-invasive monitoring interface that would allow to download all the code to the instructor's computer without interference with the students, for further analysis.

7. Conclusion and future work

Recognizing how important it is to create tools that help students develop computational thinking, we have tackled the design of a new educational platform based on the LOGO project from MIT. We have named this effort as Phogo. Phogo consists on a robot (referred to as the "Tortoise") and a high level Python library that simplifies its use. It is possible to assemble the Tortoise at a low cost (around 80 USD per robot) bringing back the original concept of LOGO as a programming language accompanied by a physical robot that can draw.

The advances in computer languages have made LOGO an archaic language, but its principles are still relevant and worth considering. We see Python as a good candidate for the base language used by the students, and the library and the robot software have been designed adopting the principle of "low beginning threshold and no ceiling". The Phogo library is transparent enough so a Tortoise robot can be controlled by any student after a very brief introduction. At the same time it provides more advanced users with the powerful capabilities of the Python language, a modern programming language that can provide many opportunities to the students thanks to its flourishing community.

Just as the LOGO Turtle, Phogo's Tortoise can move freely on a flat surface and mark its trace with a pen. The Tortoise also has a distance sensor. The use of batteries and Bluetooth wireless communication removes from the experience any cable or tethering restrictions. These features make the Tortoise a valuable tool for teaching computational basics to students, and from there on to explore more advanced matters, as proposed by Prof. Papert and his team when developing LOGO. We have published all of the Phogo documentation (design files, assembly instructions, and teaching resources) in a GitHub repository under free and open licenses.

Compared to previous attempts, Phogo offers many improvements: it is a revision to the original LOGO project, using modern components. Particularly it uses Python, a modern programing language that is more adequate for learning, as it offers a vast online community that can provide support and many libraries and expansion opportunities. The introduction of a maker vision to the LOGO original project allows to create more learning opportunities, as the students have access to the electronic components and can take part on the assembly process. This is also an important idea when comparing Phogo to Lego Mindstorms as there is a significant difference in price. Scratch and Phogo are two complementary tools. While Scratch is ideal for young students, Phogo can be more interesting for older students, and it can be a better companion for their further experience with computers. It introduces them to a standard textual programing language, that together with the physical robot makes Phogo more easy to adapt to students with disabilities, giving opportunities to a broader range of potential users.

We have tested Phogo with a group of diverse teenagers without any prior programming knowledge. They were able to learn how to use the Tortoise and how to create some complex programs in just a brief session. The students showed great interest and excitement with the Tortoise and their ability to control it through code. The majority of students were people with physical, cognitive or intellectual disabilities and all of them were able to follow, enjoy and learn as any other student making this an accessible activity to everyone.

The project is going to continue in two directions. First we want to improve the Tortoise by reducing its costs, as presented. The lower the price, the more universally accessible Phogo will become. We want to change the servo motors to stepper motors in order to improve accuracy and remove the need for a gyroscope. We also want to replace the expensive bq ZUM control board with an ESP8266 integrated Wi-Fi module that costs just a fraction (less than 3 USD). This way it will be possible to upload all the software to the Tortoise itself and make it a standalone Wi-Fi device that the students can connect to with any web browser.

We are also very interested in figuring out where is the learning ceiling, the floor and how wide are the walls of Phogo as a teaching resource. That is why we will keep organizing Phogo workshops that can motivate more students. We plan to test for the age ranges where Phogo is valid, as we hope that adults could also benefit from it.

Another possibility is to test combinations of Scratch and Phogo, and see to which extent students benefit from both approaches in conjunction, or if just one of them is enough. We also plan to test how much Phogo increases the students interest in STEM areas and if learning with Phogo has any effect their future career choices.

In our tests we have seen that Phogo can be entertaining for the students for some hours. We also plan to make longer workshops, possibly spread over different weeks, to see if the initial interest continues on and to test if there is a real learning experience that can be quantified. We want to test how different is the learning experience to the ones achieved when using Scrath or Lego Mindstorms. In future workshops, students should take pre and post tests to measure the learning outcome. Another improvement would be to automatically record all the modifications to the files on which the students work, to study the progression and how they explore the different concepts presented.

We think that Phogo offers to students the possibility to learn a popular, real world language (Python) in a progressive and intuitive manner. Python is a useful life companion and students can further explore its possibilities without any limit. In the future we want to see if students are willing to continue their learning without the robot but with other Python libraries, such as those for web programming.

Acknowledgments

The authors thank Prof. Pilar Rodríguez and Prof. Cecilia Simón for their helpful advice and relevant discussion, and also Cristina Kasner, Eduardo Hilario, and Rafael Leira at Club de Robótica-Mecatrónica UAM for their involvement throughout the organization of the workshops. P.M.R. gratefully acknowledges the generous contribution of Fil and Nala. Showcased activities took place in the context of *Programa Campus Inclusivos, Campus sin Límites 2016*, sponsored by the Spanish Ministry of Education, Culture and Sports, the ONCE Foundation, and the Repsol Foundation. This research has also been funded by the eMadrid (S2013/ICE-2715) and the E-Integra (TIN2013-44586-R) projects.

References

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (Feb. 2015). From scratch to "real" programming. *Transactions on Computing Education, 14*(4), 25:1–25:15. http://doi.acm.org/10.1145/2677087.

Bamberger, J. (December 1974). *Uses of technology to enhance education*. Tech. Rep. LOGO Memo No. 12. Massachusetts Institute of Technology - A. I. Laboratory http://hdl.handle.net/1721.1/6225.

Bamberger, J. (May 1979). *Logo music projects: Experiments in musical perception and design*. Tech. Rep. LOGO Memo No. 52. Massachusetts Institute of Technology - A. I. Laboratory http://hdl.handle.net/1721.1/5726.

Battista, M. T., & Clements, D. H. (1986). The effects of logo and cai problem-solving environments on problem-solving abilities and mathematics achievement. *Computers in Human Behavior, 2*(3), 183–193.

Benitti, F. B. V. (2012). Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education, 58*(3), 978–988. URL http://www.sciencedirect.com/science/article/pii/S0360131511002508.

Burbaitė, R., Damaševičius, R., & Štuikys, V. (2013). Using robots as learning objects for teaching computer science. In *X world conference on computers in education* (pp. 101–110).

Carmichael, H. W. (1985). *Computers, children and classrooms: A multisite evaluation of the creative use of microcomputers by elementary school children*. Final Report. ERIC.

Chao, P.-Y. (2016). Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers & Education, 95*, 202–215.

Clements, D. H., & Meredith, J. S. (1993). Research on logo: Effects and efficacy. *Journal of Computing in Childhood Education, 4*(4), 263–290.

Clements, D., & Nastasi, B. (1985). Effects of computer environments on social-emotional development: Logo and computer-assisted instruction. *Computers in the Schools, 2*(2–3), 11–31.

Clements, D. H., & Sarama, J. (1997). Research on logo: A decade of progress. *Computers in the Schools, 14*(1–2), 9–46.

Cramer, J., & Krueger, A. B. (2016). Disruptive change in the taxi business: The case of uber. *The American Economic Review, 106*(5), 177–182.

Danahy, E., Wang, E., Brockman, J., Carberry, A., Shapiro, B., & Rogers, C. B. (2014). Lego-based robotics in higher education: 15 years of student creativity. *International Journal of Advanced Robotic Systems, 11*(2), 27. https://doi.org/10.5772/58249.

Dennis, H. E., Ward, A. S., Balson, T., Li, Y., Henschel, R., Slavin, S., et al. (2017). High performance computing enabled simulation of the food-water-energy system: Simulation of intensively managed landscapes. In *Proceedings of the practice and experience in advanced research computing 2017 on sustainability, success and impact. PEARC17* (pp. 43:1–43:10). New York, NY, USA: ACM. http://doi.acm.org/10.1145/3093338.3093381.

Dog, P. F. (1985). Exciting effects of logo in an urban public school system. *Educational Leadership, 43*, 45–47.

García-Peñalvo, F. J. (2016). A brief introduction to taccle 3-coding european project. In *Computers in education (SIIE), 2016 international symposium on. IEEE* (pp. 1–4).

García-Peñalvo, F. J., Reimann, D., Tuul, M., Rees, A., & Jormanainen, I. (2016). *An overview of the most relevant literature on coding and computational thinking with emphasis on the relevant issues for teachers*.

García-Saura, C. (2015). *Self-calibration of a differential wheeled robot using only a gyroscope and a distance sensor* (Master's thesis). London: Imperial College http://arxiv.org/abs/1509.02154.

García-Saura, C., & González-Gómez, J. (2012). Low cost educational platform for robotics, using open-source 3d printers and open-source hardware. In *ICERI2012 proceedings* (pp. 2699–2706). IATED.

Gomes, A., & Brito Correia, F. (17-19 November, 2014). Programming education strategies. In *ICERI2014 proceedings. 7th international conference of education, research and innovation* (pp. 2551–2560). IATED.

González-Gómez, J., & Torres, A. P.-M. (2005). Hardware libre: la tarjeta skypic, una entrenadora para microcontroladores pic. In *Actas del I Congreso de Tecnologías del Software Libre* (pp. 57–66).

Gonzalez-Gomez, J., Valero-Gomez, A., Prieto-Moreno, A., & Abderrahim, M. (2012). A new open source 3d-printable mobile robotic platform for education. In *Advances in autonomous mini robots* (pp. 49–62). Springer.

González, I., González, J., & Gómez-Arribas, F. (2003). Hardware libre: clasificación y desarrollo de hardware reconfigurable en entornos gnu/linux. In *VI Congreso de Hispalinux*. Universidad Rey Juan Carlos I.

Grandi, R., Falconi, R., & Melchiorri, C. (2014). Robotic competitions: Teaching robotics and real-time programming with lego mindstorms. *IFAC Proceedings Volumes, 47*(3), 10598–10603, 19th IFAC World Congress http://www.sciencedirect.com/science/article/pii/S1474667016432970.

Hawkins, J., Sheingold, K., Gearhart, M., & Berger, C. (1982). Microcomputers in schools: Impact on the social life of elementary classrooms. *Journal of Applied Developmental Psychology, 3*(4), 361–373.

Howe, J. A., O'Shea, T., & Plane, F. (1980). *Teaching mathematics through Logo programming: An evaluation study*. Department of Artificial Intelligence, University of Edinburgh.

Howe, J., Ross, P., Johnson, K., Plane, F., & Inglis, R. (1982). *Learning mathematics through Logo programming: The transition from laboratory to classroom* (Vol. 118). University of Edinburgh Department of Artificial Intelligence.

Iver, M. A. M., & Iver, D. J. M. (2015). *Stemming the swell of absenteeism in the middle years*. Urban Education. https://doi.org/10.1177/0042085915618712.

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist, 41*(2), 75–86.

Klassner, F., & Schafer, B. (2014). Using the new lego mindstorms ev3 robotics platform in cs courses (abstract only). In *Proceedings of the 45th ACM technical symposium on computer science education. SIGCSE '14* (pp. 745–746). New York, NY, USA: ACM. http://doi.acm.org/10.1145/2538862.2539024.

Lawler, R. (March 1980). *One child's learning: Introducing writing with a computer*. Tech. Rep. LOGO Memo No. 56. Massachusetts Institute of Technology - A. I. Laboratory http://hdl.handle.net/1721.1/6340.

Mannila, L., & de Raadt, M. (2006). An objective comparison of languages for teaching introductory programming. In *Proceedings of the 6th Baltic Sea conference on computing education Research: Koli calling 2006. Baltic Sea '06* (pp. 32–37). New York, NY, USA: ACM. http://doi.acm.org/10.1145/1315803.1315811.

Manovich, L. (2013). *Software takes command* (Vol. 5). A&C Black.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. M. (2013). Learning computer science concepts with scratch. *Computer Science Education, 23*(3), 239–264. https://doi.org/10.1080/08993408.2013.832022.

Michayluk, J., Saklofske, D., & Yackulic, R. (1984). Logo. In *Meeting of the CAP convention, Ottawa, Ontario*.

Mohr-Schroeder, M. J., Jackson, C., Miller, M., Walcott, B., Little, D. L., Speler, L., et al. (2014). Developing middle school students' interests in stem via summer learning experiences: See blue stem camp. *School Science and Mathematics, 114*(6), 291–301. https://doi.org/10.1111/ssm.12079.

Moreno-León, J., Robles, G., & Román-González, M. (2016a). Code to learn: Where does it belong in the k-12 curriculum? *Journal of Information Technology Education: Research, 15*, 283–303.

Moreno-León, J., Robles, G., & Román-González, M. (Dec 2016b). Examining the relationship between socialization and improved software development skills in the scratch code learning environment. *Journal of Universal Computer Science, 22*(12), 1533–1557. http://www.jucs.org/jucs_22_12/examining_the_relationship_between.

Murdoch, T. B., & Detsky, A. S. (2013). The inevitable application of big data to health care. *Jama, 309*(13), 1351–1352.

Olson, J. K. (1985). *Using logo to supplement the teaching of geometric concepts in the elementary school classroom* (Ph.D. thesis). Oklahoma State University.

Paliokas, I., Arapidis, C., & Mpimpitsos, M. (2013). *Game based early programming Education: The more you play, the more you learn*. Berlin Heidelberg: Springer. https://doi.org/10.1007/978-3-642-37042-7_7.

Papert, S. A. (June 1973). *Uses of technology to enhance education*. Tech. Rep. LOGO Memo No. 8. Massachusetts Institute of Technology - A. I. Laboratory http://hdl.handle.net/1721.1/6213.

Papert, S. A. (October 1971a). *A computer laboratory for elementary schools*. Tech. Rep. LOGO Memo No. 1. Massachusetts Institute of Technology - A. I. Laboratory http://hdl.handle.net/1721.1/5834.

Papert, S. A. (October 1971b). *Teaching children thinking*. Artificial Intelligence Memos LOGO Memo No. 2. Massachusetts Institute of Technology - A. I. Laboratory http://hdl.handle.net/1721.1/5835.

Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism, 36*(2), 1–11.

Papert, S. A., & Solomon, C. (June 1976). *Twenty things to do with a computer*. Artificial Intelligence Memos LOGO Memo No. 3. Massachusetts Institute of Technology - A. I. Laboratory http://hdl.handle.net/1721.1/5836.

Perez, F., & Granger, B. E. (2015). *Project jupyter: Computational narratives as the engine of collaborative data science*. Tech. rep., Technical Report. Technical report, Project Jupyter.

Piaget, J. (1952). *The origins of intelligence in children*. International University Press.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60–67.

Rushkoff, D. (2010). *Program or be programmed: Ten commands for a digital age*. Or Books.

Sáez-López, J.-M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using scratch in five schools. *Computers & Education, 97*, 129–141. http://www.sciencedirect.com/science/article/pii/S0360131516300549.

Thieme, A., Morrison, C., Villar, N., Grayson, M., & Lindley, S. (2017). Enabling collaboration in learning computer programing inclusive of children with vision impairments. In *Proceedings of the 2017 conference on designing interactive systems. DIS '17* (pp. 739–752). New York, NY, USA: ACM. http://doi.acm.org/10.1145/3064663.3064689.

Wagner, A., & Gray, J. (2016). An empirical evaluation of a vocal user interface for programming by voice. *Artificial Intelligence: Concepts, Methodologies, Tools, and Applications: Concepts, Methodologies, Tools, and Applications, 307*.

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. IDC '15* (pp. 199–208). New York, NY, USA: ACM. http://doi.acm.org/10.1145/2771839.2771860.

Wong, G. C. Y. (2011). *Open source hardware: The history, issues, and impact on digital humanities* (Ph.D. thesis). University of Alberta.