Scuola democratica First International Conference

Proceedings of the 1st International Conference of the Journal Scuola Democratica

EDUCATION AND POST-DEMOCRACY

5-8 June 2019 Cagliari Italy

VOLUME II

Teaching, Learning, Evaluation and Technology

ASSOCIAZIONE "PER SCUOLA DEMOCRATICA"

Page intentionally left blank

Proceedings of the 1st International Conference of the Journal Scuola Democratica

EDUCATION AND POST-DEMOCRACY

VOLUME II
Teaching, Learning, Evaluation and Technology

Scuola democratica First International Conference

ASSOCIAZIONE "PER SCUOLA DEMOCRATICA"

Via Francesco Satolli, 30 – 00165 - Rome, Italy

Edited by

The Organizing Committee the 1st International Conference of the Journal Scuola Democratica

https://www.rivisteweb.it/issn/1129-731X

Published by

ASSOCIAZIONE "PER SCUOLA DEMOCRATICA"

Via Francesco Satolli, 30

00165 – Rome

Italy

FILE IN OPEN ACCESS

How to cite a proceeding from this Volume. APA citation system:

Author, N., & Author, S., (2019). Title, in *Proceedings of the 1st International Conference of the Journal Scuola Democratica "Education and post-democracy"*, VOL. 2, *Teaching, Learning, Evaluation and Technology,* pp-pp

This book is digitally available at:

http://www.scuoladem ocratica-conference.net/

ISBN 978-88-944888-1-4

Title **Proceedings of the First International Conference of the Journal "Scuola Democratica" - Education and Post-Democracy**
VOLUME II Teaching, Learning, Evaluation and Technology

This volume contains papers presented in the First International Conference of the Journal "Scuola Democratica" which took place at the University of Cagliari on 5-8 June 2019. The aim of the Conference was to bring together researchers, decision makers and educators from all around the world to investigate the concepts of "education" in a "post-democracy" era, the latter being a set of conditions under which scholars are called to face and counteract new forms of authoritarian democracy.

Populisms, racisms, discriminations and nationalisms have burst and spread on the international scene, translated and mobilized by sovereigntist political movements. Nourished by neo-liberalism and inflated by technocratic systems of governance these regressive forms of post-democracy are shaping historical challenges to the realms of education and culture: it is on this ground, and not only on the political and economic spheres, that decisive issues are at stake. These challenges are both tangible and intangible, and call into question the modern ideas of justice, equality and democracy, throughout four key dimensions of the educational function, all of which intersected by antinomies and uncertainties: ethical-political socialization, differences, inclusion, innovation.

The Conference has been an opportunity to present and discuss empirical and theoretical works from a variety of disciplines and fields covering education and thus promoting a trans- and inter-disciplinary discussion on urgent topics; to foster debates among experts and professionals; to diffuse research findings all over international scientific networks and practitioners' mainstreams; to launch further strategies and networking alliances on local, national and international scale; to provide a new space for debate and evidences to educational policies. In this framework, more than 600 participants, including academics, educators, university students, had the opportunity to engage in a productive and fruitful dialogue based on researches, analyses and critics, most of which have been published in this volume in their full version.

ISBN 978-88-944888-1-4

Premise

Since 1973, with Chile's Dictatorship as a neoliberal 'laboratory', it's more than 45 years that the Global Education Reform Movement has transformed educational systems all around the world through a discourse rooted on epistemic and ideological hegemonies. A new 'truth' of the homo economicus as able to rationally and freely pursue its interest as self-entrepreneur is relentlessly spreading: the Human Capital paradigm then connects individualistic choices and personal skills to impose diverse educational tracks through a Life-Long-Learning investment. Thus, the restructuring of the Education State, thanks to policies of privatization, competition and high stakes accountability, has implied a new ethics challenging social justice ideals.

The massification of educational systems in Europe and worldwide, together with the increasing demand for their democratization, have profoundly challenged traditional teaching models: the lecture, the magister teacher and the specific spatial-temporal devices aimed at disciplining students according to the needs of a Fordist capitalist society and to the reproduction of class inequalities. Starting particularly from the Fifties in schools, and more recently in higher education, new teaching-learning configurations have been explored and developed: situated and participatory didactics aimed at involving students in a reflexive relationship with knowledge and social reality; new ways of hybridizing formal and informal learning; new pedagogies exploiting the possibilities inscribed in new medias and digital technologies. These practices, sometimes radically, revers theory and practices in order to develop student-centred learning processes. The thematic sessions within this stream explore the challenges, tensions, ambivalences and potentialities of pedagogies and didactics innovations involving school and university teachers, students, as well as their surrounding environments: the physical, architectural, material and technological spaces that constitute a crucial component of situated learning processes.

The relation between education systems and policy making changed in the last decades, consequently to three innovations sharing the common paradigm of evaluation, namely: the establishment of national/international large-scale testing, the diffusion of systems assessing schools' and the raising interest for efficacy and cost-effectiveness of education interventions. These innovations have been highly debated from different and controversial perspectives. The aim of the conference stream is to collect papers focused on actual uses of different forms evaluation, in order to overcome previous ideological oppositions, contributing to move the debate into a more pragmatic and fruitful phase.

Further issue: How is digital technology changing education? Online schools and classes are becoming widely available; backpack of many high school and college students, instead of physicals textbooks, are now carrying iPads and various forms of devices connected to online; teachers now have more ability to personalize lessons, instructions, and projects for each group or student; by using devices and programs to distribute classwork and assignments, they can even personalize lessons and focus on the work of each student; increased opportunities and constraints for students to collaborate together from a variety of places becomes possible; free online classes called "MOOC's" otherwise known as Massive Open Online Courses are becoming widely popular. Finally, a mounting set of variegated pressures to produce pedagogical innovation in teaching and learning is being addressed to teacher and school staffs. Even the governance of school system and school-daily life as a whole is undergoing a wide process of digitalization. But what does the increase in digital technology and approach mean for the current times? Although many advantages come with digitalized learning, there are also disadvantages that researchers, educators, academics and professionals are aware of, including and not limited to minimal to zero face-to-face interaction in the classroom and the lack of ability to work in person with study partners and teachers. Any conversation that does not include the potential dangers of the widespread use of technology would not be complete. Therefore, the stream focuses also on the interplay between learning theories and technologies. Both learning theories and tools are composed of multiple attributes, and they refer to many aspects and facets which render educational technology highly complex. Evolution in both theory and technology reflects no clear successive breaks or discrete developments, rather, waves of growth and accumulation. Evolutions in society and education have influenced the selection and use of learning theories and technologies; learning theories and technologies are situated in a somewhat vague conceptual field; learning theories and technologies are connected and intertwined by information processing and knowledge acquisition; educational technologies shifted learner support from program or instructor control toward more shared and learner control; and learning theories and findings represent a fuzzy mixture of principles and applications.

Page intentionally left blank

CONTENTS

From Practice to Learning: Computer Science the Other Way Round

Stefano Federici, *Università degli Studi di Cagliari*
sfederici@unica.it
Elisabetta Gola, *Università degli Studi di Cagliari*
egola@unica.it
Claudia Medas, *Università degli Studi di Cagliari*
claudiamedas1095@gmail.com
Andrea Zuncheddu, *Università degli Studi di Cagliari*
andrea.zuncheddu@unica.it

Keywords: *Computer Science, Learning strategies, Coding, Informal teaching*

Introduction

Computer science (CS), considered in the past as a formal discipline, has changed today due to the introduction of computational thinking in primary and secondary school. This change requires a new style of teaching that is both simplified and enjoyable. New tools for coding have paved the way to a different style of teaching and learning CS. This new way, in our view, can be a model for all of school pedagogy since it allows for construction of knowledge by the students themselves.

Learning CS by learning 'instructions' works when the self-motivation of the learner is strong. This method is not effective for less motivated students. Drop-rate in CS courses is alarmingly high (Beaubouef, Mason 2005; Computing Research Association, 2011; Nager, Atkinson 2016). Moving from 'instructions' to working smartphone apps or videogames, as the current pedagogy demands, is not straightforward.

A new approach to coding, mainly inspired by *MIT*'s Scratch programming language that directly starts from videogames and smartphone apps (Maloney et al., 2010), has transformed the formal CS education delivery mechanism to a discovery- and play-driven approach. Students see and manipulate visual objects (not numerical data), organize their behaviors by assembling drag-and-drop scripts (not awkward syntax-based code), and see every step of their 'program' visually updated in the programming environment.

This new informal approach, driven by need and self-discovery, can be a model of pedagogy that enables a better learning of all school curriculum.

1. The high dropout rate in standard computer science courses

The new alternative ways of teaching CS has even proved successful in learning other school topics (Federici et al., 2018); this in turn also has the potential of reducing the high drop-rate of CS courses.

In the last decades CS teachers have modelled their lessons on the style followed by their own teachers to teach them the art of making a PC.do what they want The first step was always creating 'variables' -elements that can store values- by assigning them initial values, then modifying those values and finally showing them on the PC screen. This can be done for example by the following very short code written in the Basic programming language

```
DIM A AS INTEGER
A = 1
A = A + 1
PRINT A
```

All this code does is to show the value 2 on the screen. When looking at this example -where we have a code that creates a variable named A, then stores the value 1 in the variable, then increases the value of the variable by 1 and, finally, prints its value on the screen (that is the value 2) - many students think 'why are we not simply writing PRINT 2?!'.

Of course, asking this question means that a lot of very important computer programming concepts -such as flexibility, dynamic behavior, reuse and automation- have not been transferred to them yet. So, what they really learn at this point is that they must memorize something that they are not really understanding in order to program a computer.

It is not surprising that the number of students that drop their CS majors in their first year of college is always high all over the world. The dropout rate is over 30% in Italy and in the United States. It is over 10% (that is the highest dropout rate in the first year of college in these countries) in Great Britain and China.

Since CS is extremely engaging to a lot of enthusiast self-made programmers and gets astonishing results -such as incredible high-quality-graphics videogames produced by the entertainment industry- there must be a better way to teach it.

2. Learning computer science the hard way

The standard approach in teaching computer science has remained the same for the last 30 years. The sequence of arguments in many schoolbooks is the same: starting with binary arithmetic (that is adding numbers by using only digits 0 and 1); illustrating the components of a PC, then the basics of operating systems, and finally computers networks. Students learn to engage with the computer only when they have gone through hundreds of pages of what a computer is. Unfortunately, what they see at this point is usually the following, cryptic piece of code written in the C programming language:

```
#include <stdio.h>
int main(int argc, char** argv) {
      printf('Hello, World!\n');
      return(0);
}
```

All this just to show up a simple greeting such as 'Hello, World!' on the PC screen. And when they make simple errors, like missing a single syntax element like a parenthesis or a semicolon, the result is simply nothing. The whole application crashes. This quickly frustrates the students. To even get a really uninteresting result, they have to learn (in advance) a lot of concepts (and memorize a lot of unknown sequences of characters and symbols) to avoid the risk of getting absolutely nothing.

2.1. Learning Loops and Functions by formal teaching

After learning how to write something on the screen, the students discover the loop instruction `for`. They are shown a C program that adds up 10 numbers entered by the user:

```
#include <stdio.h>
int main(int argc, char** argv) {
      int number, sum = 0;
      for(int i=0;i<10;i++) {
            scanf('%d',&number);
            sum = sum + number;
      }
      printf('The sum is %d', sum);
}
```

The meaning of the looping operation is formally explained to them as «a sequence of statements which is specified once, but which may be carried out several times in succession. The code inside the loop is obeyed a specified number of times, or until some condition is met, or indefinitely»

They are not told that they can get the exact same result without using a loop, just by adding multiple times using the same sequence of instructions:

```
#include <stdio.h>
int main(int argc, char** argv) {
int number, sum = 0;
scanf('%d',&number);
sum = sum + number;
scanf('%d',&number);
sum = sum + number;
...
scanf('%d',&number);
sum = sum + number;
printf('The sum is %d', sum);
}
```

This process may not be convenient, but students get the same result without learning any further programming concepts. Students understand it well as this is an immediate extension of the intuitive concept of sequence of instructions, where instructions are executed from top to bottom. They still don't see the need to learn more, they get the desired result without putting in more effort.

Students face a similar problem when instructed that they can add two numbers by creating a `sum` function as in the following code

```
#include <stdio.h>
int main(int argc, char** argv) {
      int number1, number2, sum;
      scanf('%d',&number1);
      scanf('%d',&number2);
      sum = sum( number1, number2);
      printf('The sum is %d', sum);
}
int sum(int a, int b) {
      return( a+b);
}
```

instead of using the following, much simpler code, where they use the `+` operator to add `number1` and `number2`

```
#include <stdio.h>
int main(int argc, char** argv) {
      int number1, number2, sum;
      scanf('%d',&number1);
```

```
        scanf('%d',&number2);
        sum = number1 + number2;
        printf('The sum is %d', sum);
}
```

They carry out this function only after learning the formal definition of a function, that is «a sequence of program instructions that performs a specific task, packaged as a unit that can then be used in programs wherever that particular task should be performed» But why should we avoid repeating the same code more than once, or should we avoid using an existing operator and create instead a function? There are not many good reasons yet to do this. It then falls upon the teacher to present the same concepts in an interesting and, more importantly, logical manner.

3. Learning computer science the soft way

3.1. New environments to learn computer programming

In the new environment, specifically created to learn computer programming in an enjoyable and engaging way, such as Scratch, students do not need to create boring programs that merely handles numbers. Students can very quickly create multimedia interactive games by simply dragging and dropping instructions represented by colored blocks that clearly demonstrate their meaning expressed in a natural-language-like style (Figure 1). Everything is under the eye of the programmer. Making mistakes is simply not possible; they just have to snap blocks together. There is no syntax to be remembered.

The blocks are used to describe the behavior of interactive elements represented by colored pictures signifying both characters and the backgrounds in which they act. Here the students' use is not merely limited to numbers, such as 10 or 0, or to programs that use only mathematical formulas such as 'sum = a + b'. The elements handled by the programmer are 'physical'. They can see those elements and move them around, like they would do with real physical objects.

3.2. New environments to learn computer programming

Is it enough to replace numbers with multimedia objects and Keywords with draggable colored blocks? Scratch is appreciated by young kids (Federici et al. 2018). But is it easy and useful for everyone? Is it easy even for non-technical teachers should they introduce it in their classes as an educational tool?

We ran several experiments on groups of 10-20 k-1 to k-5 teachers of both humanities and hard sciences. The teachers, regardless of their group, discovered the important elements of Scratch in an unsupervised self-teaching session. All of them at the end of the session were able to build simple working projects (Federici et al., 2015).

Each session lasted 2 hours. This time is comparatively shorter with respect to that required to learn all the elements of a complex app by means of tutorials, manuals, etc. Moreover, by discovering the elements of the Scratch environment by themselves, the teachers could remember the function and position of each element.

3.1. Classic vs new languages for computer programming

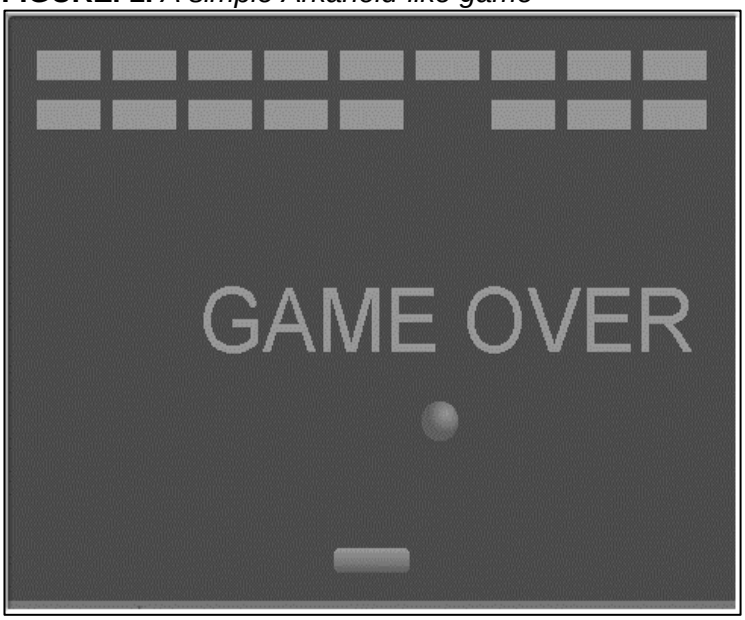
If the new environments and languages are so much better for computer science learners, why do CS teachers still stick to the old way of teaching computer science? The reasons seem historical. Computer science books have only been slightly updated in the years, mostly in their details less in the content. The

language has changed from BASIC or Pascal to C. The only real difference is just a matter of mere syntax.

3.2. From informal to formal teaching of loops and functions

Can the new programming environments, specifically designed for kids, be used to easily teach the fundamental concepts of computer science? The answer is yes. Starting from a naïve use of programming languages based on self-teaching, we can move to more thoughtful usages of programming languages that can allow students to build more complex projects with less effort. This move is guided by the teacher who can stimulate the students to learn new programming structures that lessen their burden. This can be done very easily by showing, for example, a simple game Arkanoid-like, were users must destroy several bricks by using a ball and a paddle (Figure 1).

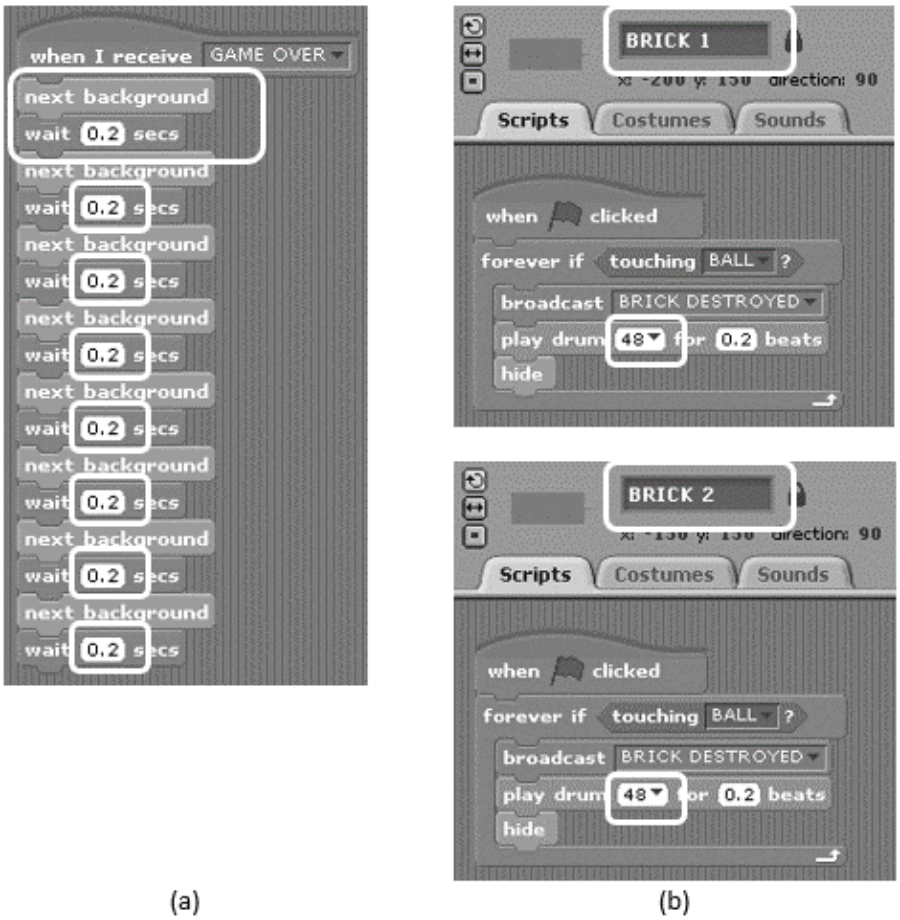
FIGURE. 1. *A simple Arkanoid-like game*



In this game the behavior of each character is described by a sequence of colored blocks. For example, when we want to animate an element by changing its look in each frame, we have a short sequence of two blocks, NEXT BACKGROUND and WAIT SECS, repeating many times (Figure 2, a). Similarly, the behavior of each brick is described by the exact same sequence of blocks (Figure 2, b).

Building the whole project is simple. We have to merely duplicate the same sequence of blocks many times. However, when we need to change a very simple feature of the project, for example when we want to make the frame-by-frame animation in Figure 1a a little slower, by changing the 0.2 value to 0.1, or we want to change the sound played when we ball hits a brick from 48 (Hid-Mid Tom) to 61 (Low Bongo) in Figure 1b, we have to repeat the same change many times. The teacher can then ask the students to experiment by looking for the perfect timing or the perfect sound in order to enable them to enact the same set of changes many times over.

FIGURE. 2. *Repeating blocks in a single sequence (a) or in multiple sequences (b)*



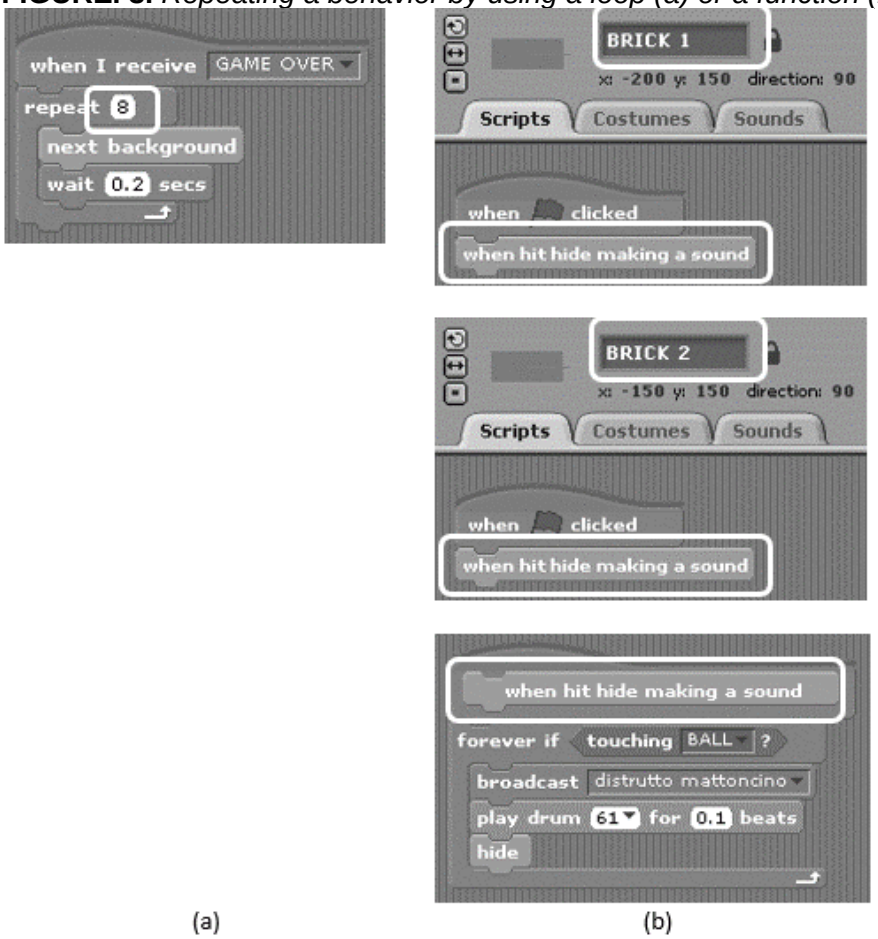
(a) (b)

At this point the teacher can remind them that in real life instructions are not repeated multiple times, for example when we describe a repeated sequence of actions in a recipe or the same sequence of actions in several recipes. It is thus quite reasonable to follow the same when we formulate instructions by means of a programming language. We can shorten a sequence of repeated blocks by using a REPEAT block (Figure 3, a) and we can shorten the repetition of a sequence of blocks in many characters by giving it a 'name' (Figure 3, b). So, in the Arkanoid project we are saying that a frame-by-frame animation is done by changing the picture (the 'background' in Figure 3a) many times and that the 'WHEN HIT HIDE MAKING A SOUND' behavior of each brick is identical for all bricks (Figure 3b). Making this kind of shortening is quite convenient when the number of repetitions get higher, as it happens in a complete project.

These two ways of shortening repeated sequences of blocks are the two basic mechanisms of 'loops' and 'functions' that we have already formally described in the top-down teaching strategy described in the previous paragraph by using the C language. So, the concept of looping is to the students just 'repeating actions' instead of having to list the repetitions one under the other by wasting a lot of time and space. Similarly, the concept of function is 'giving a name' to a sequence of actions, instead of repeating the same sequence in several characters by wasting again a lot of time. The students can create complex projects without getting frustrated by having to learn complex concepts in advance. Programming can still be fun and rewarding. Students can then get rid of new sources of frustration when they will have to make the same change

multiple times by learning how to 'repeat actions' and how to 'give sequences of actions a name'. That is by learning loops and functions.

FIGURE. 3. *Repeating a behavior by using a loop (a) or a function (b)*



(a) (b)

4. Better performances when learning computer science in an informal way

Building games is not the only way of acquiring important skills provided by learning CS. Moreover, learning computer science is not done in isolation. CS principles can be learned by using computer programming as a tool to learn other school disciplines in an interesting and -more importantly- deeper way. Computer science can be learned by using it as the new 'pencil and paper'. Starting from a given task (like learning history, learning a foreign language, learning science, mathematics, etc.) students and the teacher can identify the basic elements of this task and can then learn how these elements are related to each other by building 'tangible' digital elements whose behavior is described by sequences of blocks. When they build a working simulation of a given school topic, even in collaboration with their teachers, students really learn the underlying laws of the topic under consideration. The results of these different pedagogy of school topics are extremely encouraging. It has been demonstrated in an experiment about learning exponentiation -that is learning that, e.g., 2^3 is 2x2x2- that not only the students are more engaged during the lessons but they acquire the concepts learnt in a better and, more importantly, durable manner in the process improving their retention (Federici et al., 2018). This alternate

way of teaching computer programming has been successfully applied in other school topics like foreign languages and history (Federici et al., 2019).

Conclusions

Learning computer science is best begun informally followed by formal instruction that enables one to easily learn technical concepts like loops and functions. Results of our experiments show that students enjoy this form of learning. Following this method, students can acquire both concepts and skills and get interesting outcomes. We have shown that using learning strategies based on computer programming gives very good results even on other school subjects like mathematics, language, and history. Computer science does not need to be an arid topic, where teaching is only based on numbers and abstract structures. It can be taught by using 'tangible' elements and simple structures thanks to new programming environments that, even if designed for kids, have been fruitfully used by students at all levels, from elementary school to college.

References

Beaubouef, T., Mason, J (2005), «Why the High Attrition Rate for Computer Science Students: Some Thoughts and Observations», *Inroads – The SIGCSE Bulletin*, 37(2), pp. 103-06

Federici, S., Gola, E., Brau, D., Zuncheddu, A. (2015), «Are Educators Ready for Coding? From Students back to Teacher: Introducing the Class to Coding the Other Way Round», *Proceedings of CSEDU 2015* Vol. 1, Lisbon, Portugal, pp. 494-500

Federici, S., Medas, C., Gola, E. (2018), «Who Learns Better: Achieving Long-Term Knowledge Retention by Programming-Based Learning», *Proceedings of CSEDU 2018,* Vol. 2, Funchal, Portugal, pp. 124-133

Federici, S., Sergi, E., Gola. E. (2019), «Easy Prototyping of Multimedia Interactive Educational Tools for Language Learning based on Block Programming», *Proceedings of CSEDU 2019*, Vol. 2, Heraklion, Greece, pp. 140-53.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E. (2010), «The Scratch Programming Panguage and Environment», ACM Transactions on Computing Education. 10(4)

Nager, A., Atkinson, R. D. (2016), «The Case for Improving U.S. Computer Science Education», *Information Technology & Innovation Foundation*; http://www2.itif.org/2016-computer-science-education.pdf