



**HAL**  
open science

## When sharing computer science with everyone also helps avoiding digital prejudices

Marie Duflot, Martin Quinson, Florent Masegla, Didier Roy, Julien Vaubourg, Thierry Viéville

### ► To cite this version:

Marie Duflot, Martin Quinson, Florent Masegla, Didier Roy, Julien Vaubourg, et al.. When sharing computer science with everyone also helps avoiding digital prejudices: Escape computer dirty magic: learn Scratch!. SCRATCH, Aug 2015, Amsterdam, Netherlands. hal-01154767

**HAL Id: hal-01154767**

**<https://inria.hal.science/hal-01154767>**

Submitted on 27 May 2015

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - ShareAlike 4.0 International License

# ESCAPE COMPUTER DIRTY MAGIC: LEARN SCRATCH !

## WHEN SHARING COMPUTER SCIENCE WITH EVERYONE ALSO HELPS AVOIDING DIGITAL PREJUDICES.

Marie Duflot, Martin Quinson, Florent Masseglia, Didier Roy,  
Julien Vaubourg, Thierry Viéville.<sup>1</sup>

Accepted to : <http://www.scratch2015ams.org> as a «practice report».

### *Abstract*

*We, computer scientists, have to increase human knowledge, e.g. help to better understand what is mechanical intelligence. We further have to contribute to generate good goods out of it. But we also have the duty to share this knowledge with everyone. To be sure that no one endure, whereas each one benefits from the derived technology. And we observe that the main challenge is : free people from preconception, misconception and prejudices about computers and robots. And the main enemy is “belief in artificial intelligence phantasms” ! To escape from such deleterious magic, creative computing with Scratch is our best friend, unplug activities our best lever and playful robotics our best partner. Let us tell it as a story..*

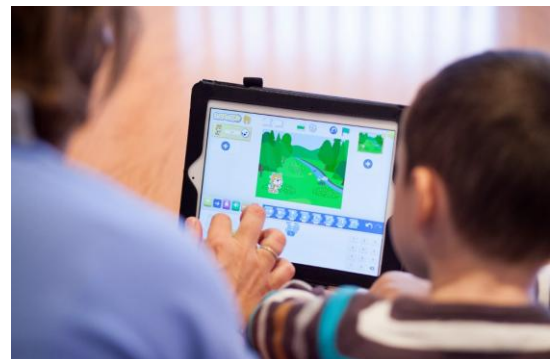
**Keywords** Scratch, creative programming, computer science, popular education, preconception.

**Acknowledgments** We are especially thankful to Martine Courbin-Coulaud and Valérie François regarding the contents engineering of Pixees and Véronique Poirel for the organization of the CodCodCoding action.

### **1. Introduction: why sharing useless computer science concepts with my father or mother in law**

France is quite a nice country. However, regarding digital literacy, France is a retarded country. Believe it or not, but we do *not* teach computer science to our kids, not even computer technology, only hardly to use common software. The reason is that we collectively do not even realize that there is something to learn here, since nobody has taught us *what it really is*.

And the consequence is that since computing is not



---

<sup>1</sup> Inria, BP 105, 78153 LeChesnay Cdx, France. Contact: [mecsci-contact@inria.fr](mailto:mecsci-contact@inria.fr)

supposed to be science, while computers work amazingly and do fascinating things, this must be ... magic. Not “nice magic” (the one you can play and have fun with), but “deleterious magic” (the one you endure and you are scared of). Exaggeration? Imagine people believing in entities created by humans but that can “breed” and become more clever than humans, thus (this is a minimal requirement if you are such an entity) dominate the world and (go ahead) eradicate the humankind. Some B soap-movie? No. A real claim by a tremendously clever guy<sup>2</sup>, very good in sciences .. but apparently *not* in *computer* science.

Is such an idea refutable? Easily, with a first year computer class argument. Such a reproducible artificial intelligence must indeed at least be able to detect (and correct) its own bugs. Say, infinite loops. However, there is *no* (finite) algorithm that can detect all possible infinite loops in other algorithms. This is a proved theorem. So, such artificial intelligence is not even going to be able to avoid loop traps. Bad start to dominate the world, isn't it? More generally, computer science has studied in details the huge possibilities but also the strong intrinsic limits of computing.

However, we can change our mind, and start being *interested in* computer science, so that it starts becoming something we understand. Then, the fact we understand allows us to get control and manage it. At this point, there is a legitimate question: **How to?** How can we help doing that?

This is exactly the question we would like to address here.

Without theory. Simply the will to witness about what we observe when we have the pleasure and privilege to share such pieces of computer science with everyone. We first are going to report on what can be done before using Scratch, i.e. unplugged activities, then how we use Scratch and finally how, after using Scratch we can introduce small robots to get further on. No theory here, only concrete stories and some comments about the lesson we learned from this practice.

## 2. Starting to understand computing without any ... computer



Let us start with an experience where primary school teachers and computer scientists have shared some pieces of knowledge on the question of teaching the "science" in ... computer science. Of course, different teachers have different reactions to the idea of bringing this subject into their classrooms, since it is not yet explicitly in the curriculum. Some are enthusiastic while others are more doubtful, but still curious about the approach, that can help for other topics. Because there is one point they all have in common: They are able to match computer science teaching with activities and subjects they already teach

in their classrooms! Such a matching is certainly a decisive lever to help teachers take up computer science in its most transferable form to school. Here are some examples that have been effective.

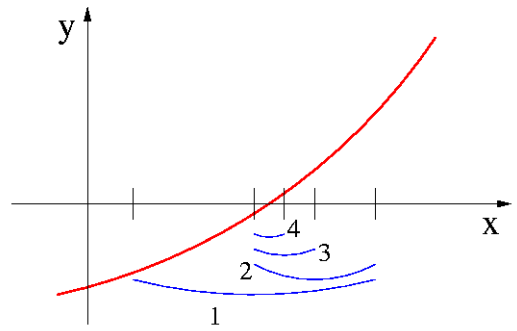
---

<sup>2</sup> Stephen Hawking, actually <http://www.bbc.com/news/technology-30299992>.

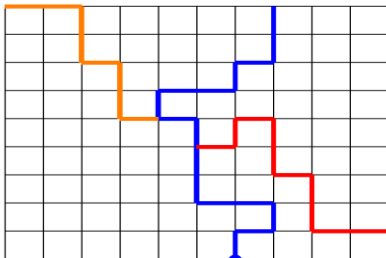
## The universal idea of dichotomy.

Next time you can talk with a primary school teacher about computer science in classrooms, try the following: Find some way to talk about dichotomy (e.g., how to find a word in a dictionary, or how to find the exact price between 0 and 1000€ just answering “more|less|exact” to any proposal). How dichotomy is wonderful compared to just enumerating the elements of a set. It is highly likely that the "target number" game is *not* unknown to such teacher.

She or he will probably explain that the "target number" is an activity where the teacher writes a number between 1 and 100, hidden behind the black board (yes, with chalk... as in the 20th century). The kids then have to "guess" the number. They try, and try again, one after the other and, of course, they don't find it. Well, to be quite correct, they won't try hard enough to find it, and eventually lose patience. Then, the teacher says "New rules! Now you are allowed to ask 'greater or lower than?', shall we try again?". And, gradually, the teacher explains the quickest strategy to find the target number. The first good news here is that teachers already have dichotomy in their classrooms. The second good news is that, now, with the computer science point of view, they will be able to talk about algorithms and complexity thanks to this activity. They may give a comparison between the exhaustive search. They may ask "Kids, how many tests do you need if you are not lucky?" or "With dichotomy, how many numbers have we permanently cut after the first test? And after the second test?". This is just an update on this activity. It requires some knowledge about algorithms but, with the motivation of reusing existing activities, it should be feasible.



## The treasure hunt game to be played as a robot.



**TDGGTDTTGDDTGDGT**  
**TDGGTDTDGDDTGDGT**  
**TDGGTDTTGTDGTGDGT**

Treasure hunt is an activity where a map of the classroom is drawn. A treasure is hidden somewhere in the classroom, and the map gives the path, from a start point, to the treasure. The goal, for kids, is to follow the instructions given by the map and find the treasure. This activity looks like a perfect opportunity to talk about bugs and languages, two major notions of computer science. Indeed, instructions might be given in two languages. The first one is based on relative positions (e.g. "one step forward", "rotate to the right, a quarter turn"). The second one is based on absolute references (e.g. "one step towards North"). This activity may be funny since you will have lots of bugs (and improbable

situations). You may find that the bugs in an absolute reference frame stay closer to the treasure. But no mistake, a bug is a bug! In such a situation, the very interesting thing is that the kid is playing the role of a ... robot. With, say, three instructions (forward, right, left) and *no way* to do something more clever. As a consequence, when playing this game we discover that a robot cannot “magically be clever”, it can only perform what it has been programmed for. This is a crucial point, we must help kids discover the profound difference between *their human intelligence* and computer *mechanical intelligence* (i.e., algorithmic intelligence). A step further, this “unplugged” activity is an excellent preparation to Scratch programming: Consider “drawing a square” (i.e., repeat 4 times, forward and turn right). Performing the activity in unplugged mode really lightens what is going to be done with the Scratch cat when online.

**Playing as a robot again, with extra-ordinary kids.**

Following this idea, an activity has been created and experimented in a primary school with first year kids and with a class of kids with cognitive disorders. We managed to have them program, discuss on the interest of a programming language compared to another one, and even talk about complexity.

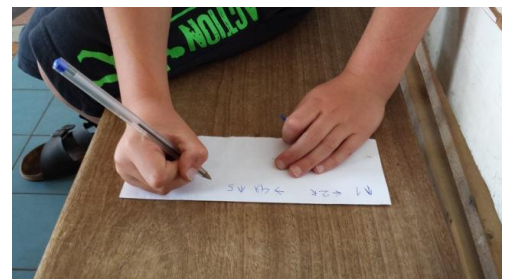


A landscape was drawn on a very large piece of fabric that was put on the ground. Regular dots were put on the landscape to form a “grid” and thus define the length of a step for the robot. Several areas (water/mountain/forest) were not supposed to be reachable so no dots were drawn on them. The kids were first given a “program” consisting of arrows in different directions that one kid was reading when another one was executing, taking the role of a

robot.

After some time to get used to the language, other programs were given, including some with bugs. The kids were globally puzzled. The robot was having doubts on the reader's job and vice versa, and it was interesting to see how difficult it was for them to finally tell that the error came for the professor who had written this program. It was then easy to make a parallel with programming, where the computer executes obediently the program, without making one extra step right to reach the bridge and not get into the river. The activity went on with several possible modifications of the language (allowing to turn and move forward instead of making steps in absolute directions, adding numbers to group together several steps forward, grabbing something on the ground with a new instruction).

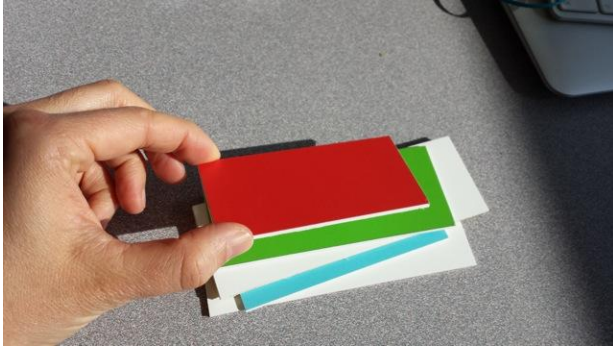
The last stage consisted in letting them write their own programs to accomplish different tasks. Several groups were each writing a program that was tested by another group. The kids thus needed to show their creativity as well as their ability to work together. Several questions arose in the teams, mainly concerning the different possibilities offered to achieve the same task. Was it better to move around and grab objects one by one or to go to a remote place and grab them all at the same time? This gave us another opportunity to link the activity with computer science by simply talking about complexity... with 7 years old kids.



The feedback from the teachers was also very positive as, in addition to its interest related to computer science that was expected, the teachers found many other benefits in this activity, like cooperation between kids, the ability to move on a grid, identifying right and left (wristbands with colors and colored arrows were given to those who had troubles with differentiating them) among others. It is also worth saying that in this activity with such young kids, the possibility to move around was a very important factor. It really helped the participants to get into the activity, as well as to effectively see what a program was doing. In that respect an easy link can be made with Scratch activities that enable a cat to do the same things and even more... on a computer.

**The pancake sorting activity.**

A key point in writing a program is being able to formalize one's idea. Even when it is easy to play with tokens to sort them, it can be really tricky for people who don't know anything about programming to explain the method, a method that is clear and simple enough so that a computer can execute it, and that works for every initial situation. This process has been experimented on a "pancake sorting" game, created in 2012 by Martin Quinson and Jean-Christophe Bach, reusing a common unplugged algorithm. The activity is organized in three stages, described below.



A cook is making pancakes, supposed to be of non constant size, and wants them sorted with a nice pyramid shape, from the largest pancake at the bottom to the smallest at the top. Since he has no space left on his table and for obvious hygiene reasons cannot touch the pancakes with his hands, the only thing he can do is use his spatula, insert it in below any pancake in the stack, and turn upside down all the pancakes on his tool before putting them back on top of the remaining stack. In this

activity the participants (from age 6 to adults) were asked to first play the game themselves with plastic "pancakes" of distinct size and colors, and try to get a sorted stack of pancakes while respecting the only move allowed. After a few minutes they almost all succeeded to sort their stack of pancakes, even if a few of them admitted that they didn't know how they had done it. For those who were stuck at some point, usually close to the beginning, the role of the organizer was to lead them to identify what would be a good first step, here having the largest pancake at the bottom. The next stage was to let them do it using their eyes and their brain but not their hands. The stack was given to someone else who was supposed to execute the steps as asked. The instructions were usually of the form "flip the 3 topmost pancakes" or "take them down to the blue one and flip". Several times however the instructions were imprecise like "flip the whole stack" and then "no, the whole stack minus the bottom one". The bottom pancake was indeed already at the right place so for them it was obvious that the robot didn't need to take it any more. This stage confronts them with the necessity to be precise enough so that the robot executes what the player means. The role of the robot is important as well: he has to be as idiot as a computer can be, precisely **not** trying to guess what the player means, which is surprisingly a not so easy task. The last stage was then to tell them that they could now only use their brain, and that the robot alone could use his eyes and hands. The stack was thus shuffled and hidden from the player. The participants were puzzled by the fact that they could not have a look, not even at the initial configuration. It was then easy to explain that,



when interacting with a computer that is doing things in a microsecond, it is impossible to give the user a few seconds now and then to look at the configuration. A real programmer has to write an algorithm once, and let it execute to the end on its own.

In this activity the three stages are important. If skipping the first one the participants might not know where to start and just not do anything. The second stage is not formal yet but it permits both to put words on an idea and to detect errors in the method and in the "programming language". It is thus important to let them go through it without requiring a more formal/generic way of describing their strategy/algorithm. The third stage is the most difficult one for the

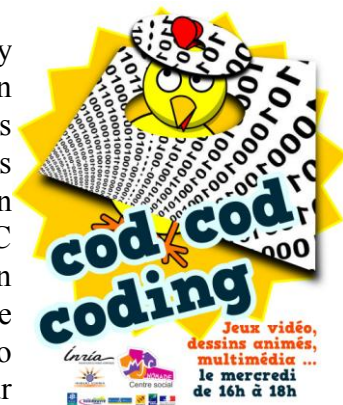
participants, but the errors detected during the second stage help them get an idea of what is to be done. If in trouble with this stage the player can go back to the stage where he could see the stack and be asked why he does a particular sequence of moves. The above activity can be enriched with a new constraint. Now pancakes have a nice side (the colored one) and an ugly one (white on our plastic pancakes). The player thus has to get them sorted by size but also to put all colored sides on top to have a nice pyramid. The solution doesn't change much but now when the biggest pancake is put on top of the stack, it has to be put upside down so that when the whole stack is flipped the colored side is on top.

In addition to the ability to put one's ideas into words and formalize a solution to a problem, this activity has two other advantages. First it introduces two key ingredients in algorithms: conditional when deciding whether or not to flip the biggest pancake on top of the stack, and loops (or recursivity) when repeating the same three steps several times with a stack getting smaller and smaller. The second advantage is that this very simple algorithm is a good means to talk about complexity. Participants first think at how many moves they needed to sort the stack, and come up with the answer that it depends on the initial situation. The group usually quickly converges to a complexity in the worst case of 13 (if optimized) to 15 moves for five pancakes. It is then natural to extend that to an arbitrary number of pancakes and leading the discussion to the notion of complexity that takes into account the number of steps as a function of the size of the data.

But it is now time to open the computers, and enjoy creative computing thanks to Scratch.

### 3. Starting from Scratch to do more that scratch the surface

Since September 2014, a programming club named CodCodCoding, mainly based on Scratch activities is opened to a group of nine children, between seven and twelve years old (six boys for three girls). We offer a two hours session each week, during a whole school year, with two animators. This activity comes from a partnership between Inria (the main research center in computer science in France) and a French local youth cultural center (MJC Centre Social Nomade). As a result, one animator is a PhD student in computer science, while the other one is a youth facilitator. Thanks to these dual skills, we experimented various tips to teach Creative Programming to children. The Creative Computing Book proposed by the Scratch Ed. was our main resource for the animators. This is why Inria have translated this precious resource in French (available on <https://pixees.fr>) with the equally precious «Starting from Scratch» manual from the Royal Society of Edinburgh.



Though such manuals are very rich in terms of contents, in such a context children prefer handouts with only the five main instructions or steps for the current working session. Sticking to these five instructions in order and checking that it was correct proved out to be already a challenge for kids that do not learn computer science.



In this section we would like to report on a very interesting aspect of programming small projects with Scratch. Beyond computing, it is

a real lever to discover how to work together and how to learn experimental collective investigation.

One of the main difficulties for the children is debugging their own code. Writing a clean code is very important, because they have to be able to read it easily when a bug occurs. Moreover, they have often the responsibility to write something clear enough for someone else to carry on with it. The Scratch remix function is a very good opportunity to educate such free software concepts, and the clarity of the code is one of them. A perfect example of such difficulties is the chosen names for their messages in Scratch: “loloolloooooolllooo”, “roottlllffffffroorlfrotlf” or “trololololo”. Describing the meaning with the right words was already a challenge.

In order to help them regarding this issue we tried the Free Access Solution. During this session, they had to add a difficult functionality to their projects, without help from us. As a replacement, a computer with a Scratch project containing the solution, was freely available in the room. The main reaction of the kids was “oh, it's very easy if we have the solution, it's a cheat!”. In fact, they had not realized that the base of the project with the solution was not the same as theirs. As a result, they had to read and understand the code of the project to find the right blocks. This was a very good solution to help them to reread a source code and make them receptive to the importance of code clarity.

Once a project was completed by the group, we asked each child to present her or his project to others. He or she had to explain how to use the project, to show how it worked, and possibly to talk about the difficulties encountered. The other children had to evaluate the project, with reasoned and constructive comments. Knowing how to correctly review a work is very important, but very difficult. The audience must comment the work with constructive remarks beyond “so good” or “that sucks”, and not only through comparisons with their own version of the project (“mine is better”). We also noticed that, with ten kids around the same computer, everybody was talking at the same time and they were not listening to each other.

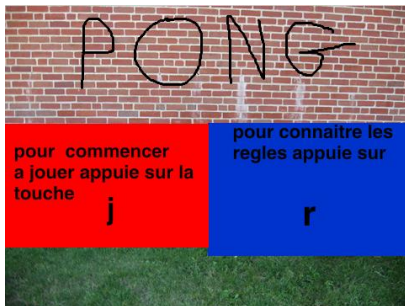


To solve this problem, we used the well-known Talking Stick (a piece of rope) trick, allowing someone to talk. There are two sticks, one for the one presenting the project, and one for the reviewer. This simple trick really changed the group behavior. It became a game for them, and each child with the stick played the right role, trying to say something clever and useful, while others quietly listened to him. Other children seemed to think about something even more relevant to say, as in a constructive contest. The remarks were sometimes composed of somehow meaningless words, but they were globally progressing in the right direction.

Sometimes, we observed a child “accusing” another one of cheating in doing the same thing as him. View something and reproduce it is not necessarily a cheat, because a creation is rarely completely new and often inspired from others. We all unconsciously copied some ideas from games we know for creating our own game. It was very difficult to convince them that it can be satisfying to be copied, because only good ideas are copied. On the other hand, we also stated that plagiarizing someone is not a good thing, especially if this person is not cited in the project. The difference between inspiration and plagiary is subtle, and we helped to draw the limit between



them. This point is important for education, beyond Scratch and the programming world. It opens the door to true collaborations.



When they create, the children do not have the final user in mind. They create their own game, for them, with their own rules and logic. For example, for moving a sprite with the keyboard, a children had associated the left arrow key to an upward movement, the right key to a left shift, up key to a downward movement and the down key to a right shift. This seems right to the children, who does not understand why he should change. Another example is the rules at the beginning of the game. Some rules are totally fool and impossible to guess without notice, e.g. you can win only if you touch the top-right corner of the scene, after catching all the items.

Sometimes, visitors or other animators were present in the cultural center. We took this opportunity to ask someone to try the children's game. Once a kid considered he had completed his game, he could do the next activity only if the tester was able to play his game alone and without difficulties to understand it. As a result, the children were forced to write rules at the beginning of the game, and chose the keys to use with a minimum of universal logic.

Shall we stay “in” the computer ? Not necessarily: We may also animate real objects to better understand the interaction between computing and the real world. This is our last step.

## 7. Playing with robots to fall in love with computer science

Lola, 8 years old: "I know programming, I do not like it. My brother showed me. It is only words everywhere. So boring. More than that, it is ugly. "

Ramzy, 7 years old: "What are you talking about? So strange! You just like IniRobot, you use it to *program* Thymio using VPL. But programming, you do not like! "

Lola, again: "Yeah. This is it. No matter if it bothers you :)"

[IniRobot](#)? It is a series of open source didactic robotics activities, written in French, and created by the Inria <https://flowers> team


[Thymio](#) II? It is an open-source tiny mobile robot toy, mainly developed by the LSRO Team of the EPFL.

[VPL](#)? The acronym for Visual Programming Language, the way Lola likes to program her robot.

**Carnet de missions IniRobot**

Documents ensembles    Indications sur la difficulté    Durée estimée    Niveau de Thymio    Niveau de langage    Activité débranchée


---

**MISSION 1 : C'est quoi ce truc ?** 

Il s'agit de découvrir le robot Thymio 2 en totale autonomie.

**Déroulement :** Le robot est donné aux enfants. La consigne est simple : ils doivent découvrir Thymio, sans aucune indication.  
 À la fin de cette activité, les enfants doivent savoir allumer le robot et constater qu'en appuyant sur les flèches, Thymio change de couleur.

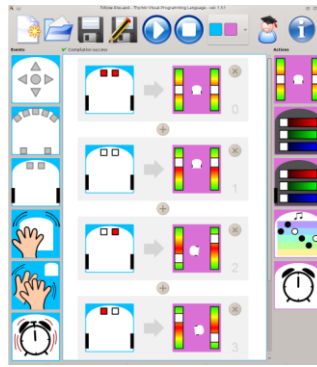
---

**MISSION 2 : Des couleurs et des comportements** 

Doc externe : Fiche à remplir

Il s'agit de découvrir les programmes pré-enregistrés dans le robot.

**Déroulement :** On explique qu'il existe des programmes pré-enregistrés dans le Thymio, que les flèches servent à faire défiler les comportements et que le bouton rond sert à valider. Il faut compléter la grille en donnant un nom à chaque comportement et entourer ou colorier sur le schéma les éléments touchés. Pour aider à donner un nom, on peut dire : « Si c'est ».



A sequence of robot programming activity using the Thymio II with the VPL interface.

With IniRobot, children have a series of tasks to perform. They gradually discover the robot mechanisms and learn how to program it through an easy to use purely visual interface. To realize a “mission” one connects proximity sensors, accelerometer, light and sound sensor to the wheel motors, colored LEDs, or sound synthesizer. Each group of kids can progress at its own pace through small activities. This goes on until the last challenge: Explore an unknown area avoiding obstacles. After inventing their own algorithm and programming it, they test the robot process, in order to adjust its exploration strategy. And robot visual and audio decoration is not to be forgotten, as it is a good way to recognize each robot, to discover programming, and to create funny and cute things. Then begins an amazing and unexpected choreography of robots on the given pathway, in alternance with interactions among themselves, and with obstacles. Furthermore, a color pen allows to mark the robot trajectory.

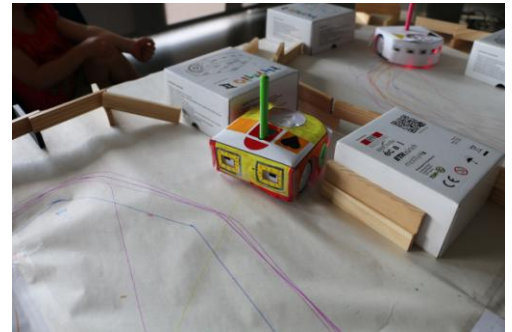
With such a didactic paradigm, robotics offers a micro-world of integrated learning, quite motivating. The key point is that one puts into action computational thinking, linking the digital world and the physical world. Using what we may call “tangible objects”. As such, it is an irreplaceable paradigm for teaching computer science, including investigation, learning by trials / errors, discovering cooperation in a small group, and so on.



Then, it appears that the existing visual programming language is quite attractive and easy to learn, but only includes sequences of event-driven construction. The only king of this kingdom is :

“On <event> do <action>”

As such, it is challenging, because it allows kids to think not only as a “piece of code” but make them smoothly familiar with reactive programming, which is the base of modern computing (e.g., Javascript in a Web page or Smartphone applications). Scratch includes such a paradigm, indeed. But also allows to make nice loops, manage variables, make use of timers and exchange messages. This is the reason why we have *also*<sup>3</sup> connected Thymio II, with the related IniRobot activities, to both the Scratch (<http://scratch.mit.edu>) and the Snap (<http://snap.berkeley.edu>) platforms (<http://www.inirobot.fr>). The first one, to be able to share activities with the biggest (and smartest :) creative programming environment. The second one, to be able to create “tractable” dedicated interfaces (e.g., oversimplified, or integrated on a Web page with other medias).



So Lola is going to be happy: she will program all what she wants with a cute and not ugly tool.

## 8. Conclusion: how to get one step further

The next step is to be able to help not only a few groups of kids, but push this initiative to a much larger scale. The lever is the training of teachers and activity organizers. To this end, resources, online support, best practice reports, and news sharing have been made available. We have translated available resources and an on-line MOOC formation is in progress.



<http://jecode.org> a community of actors.



<https://pixees.fr> a scientific community, online to help you sharing computer science with everyone.



*Al*  
l-inclusive open resources, and fully validated, made available.

<sup>3</sup> In fact, the development effort to connect Scratch (and Snap) to the smart open-source and open-hardware humanoid robot <https://www.poppy-project.org> printable on 3D printers, and Poppy ergo Junior, one of his cousins.

So that kids “no only play video-games but also build their own” (B.H. Obama II, 2013).

