AlgoPath: A New Way of Learning Algorithmic

Estelle Perrin
CReSTIC
IFTS – University of Reims
Charleville-Mézières, France
estelle.perrin@univ-reims.fr

Sébastien Linck
IFTS – University of Reims
Charleville-Mézières, France
sebastien.linck@univ-reims.fr

Frédéric Danesi
DINCCS
Charleville-Mézières, France
frederic.danesi@dinccs.com

***Abstract*— This paper presents a new way of learning algorithmic: AlgoPath is a virtual world in which variables are represented by 3D figures carrying a backpack and the sequence of instructions is represented by a stone path. The interface of AlgoPath helps students to avoid common mistakes. The world of AlgoPath gives them a mental representation of algorithms. Students are more prone to learn because AlgoPath changes the level of difficulty. They can forget the off-putting syntax and grammar of algorithmics. AlgoPath is ludic and students feel they are more playing than learning.**

Keywords- 3D-based training; education; algorithmics; ludic teaching

I. INTRODUCTION

Learning how to create algorithms can sometimes be frustrating for students in the urge of using computers. We now live in a world where numericals are everywhere: playing is numerical, reading is numerical, and socializing can be numerical. Asking students to handle a sheet of paper and a pencil in order to learn the basic concepts of programming is kind of out of phase. Bringing back numericals in the process of learning can be perceived reassuring for students and can increase their interest and their concentration.

But numericals cannot solve everything. They have to be attractive enough or they can be classified as being as boring as a sheet of paper and a pencil. On the contrary, students can learn while playing with a good numerical game. Nowadays games are virtual and teenagers agree that the most important characteristics in a game are those that contributed to its realism [1].

In our university, we do have students that have to learn algorithmics although they don't have an extended scientific background. One of our under-graduate courses is dedicated to websites – how to design, how to add on modules, how to write for the web, etc. Some of the students are more familiar with communications while others are more familiar with informatics. But they all have to attend a course dedicated to algorithmics. The students whose mind is not shaped to rationalize and to think in a way that fits logics, sometimes have difficulties to catch a good mental representation of an algorithm [2] and sometimes are not motivated to do so. As hard teachers can keep trying to explain, a variable or a loop, for example, is and remains abstract and some students just do not conceptualize it. As a consequence, these students will not even bother learning the syntax of algorithms and they will fail without a doubt. Algorithmics is just something they are not motivated for. So they need something attractive to help them learn and avoid errors. Using computer games is one means to encourage learners to learn. Even if game-based learning is not the most efficient learning method per se, games enhance motivation and increase students' interest in the subject matter [3]. Indeed, there have already been quite a lot of research projects towards the development of software games for education that aim at increasing the students' motivation and engagement while they learn [4][5].

In Section II we will review visual programming dedicated to algorithmics and we will show that their interfaces are traditional while the game-based environment is more enjoyable for the users than a traditional environment [6]. In Section III, we will show that algorithms can be represented as a virtual world in which 3D figures walk along a path. In Section IV, we will briefly show the implementation of an example from scratch and we will finally conclude and reveal future work in Section V.

II. OVERVIEW

In this section, we study various graphical algorithms and programming editors. All of them have advantages but also different disadvantages. We will focus on two characteristics of these editors: (1) the appearance of the editor (see Section "A") and (2) the representation of the programming code or algorithm (see Section "B").

The software studied is designed to help students learning algorithmics whatever their age. Some of the software, mostly those dedicated to algorithm editors, are developed by standalone or amateur developers. The others are created by researchers or programmers. For example, Algoris is a software created by an amateur who makes small programs for pleasure [7]. He developed Algoris to help young and older students to learn algorithmics with a visual method. Scratch was developed by the Lifelong Kindergarten Research Group at the MIT Media Laboratory [8]. It was designed to introduce concepts to children in mathematics and computer science [9].

A. Appearence

The colored visualization of a program or algorithm, whatever the kind of code it is, is an important point to help students to understand.

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-177-9

Colors can be used as an attention-getting element in program interfaces. Algoris and Scratch follow this method by using a colored enhancement in the text and the representation of their algorithms [10]. Both editors have rounded and colored interfaces but although they look like identical, they have different aims. The first one is to learn algorithmic to create real codes. The second one is to simplify the creation of software. In first case the aim is to learn, in the second the aim is to learn unaware of doing so.

Scratch looks like a game. Its instructions are represented by boxes that fit like Lego bricks. These bricks are classified in categories (for example to move or to speak), which complicates the task of learning the program. Algoris has a form of circuit like a flowchart. A ball goes through boxes connected by pipes. It is a helpful representation to visualize what is inside the loop and what is done or not. Alice2 enhances the visualization thanks to a programming environment designed by building 3D virtual worlds [11].

Not all learning algorithm or programming software is beautifully designed.

But colors can be distracting too. With dark colors and shapes, LARP [12] and AlgoBox [13] seem less practical, stricter, and so more serious. Algobox is not graphic. The code is represented by text and is well made up. It follows the structure of an algorithm. LARP constructs algorithms in a flowchart form. It is pretty visual, but the user has to know the representations of every element. However its flowchart is not attractive to the eye. Jeliot, a java editor is divided into two parts [14]. The first part is a text editor (for programming purposes) with highlighted syntax keywords. The second part, called the "theater", is hidden by a blue curtain during the drafting stage of the code. At the beginning of the execution the curtain opens and reveals a four-part window. These parts represent the element classes of the program (method, expression evaluation, constant, instance, and array). The execution modifies adequately each part at each step of the program. The advantage of this software stops there, since the user has to type in the whole source of his algorithm.

Alvis is apparently a fairly classic program [15]. When you open it you first see a text editor. A whiteboard with a toolbox is placed next to it: this complementarity makes Alvis interesting. Alvis makes a parallel between the source code and the representation of the elements in memory: boxes are for variables and boxes containing boxes are for arrays.

One important standard of a good learning program is the appearance. Scratch combines the most attractive one and a playful display of the content of the variables. Alice2 makes a good evolution in representation by introducing 3D and can be seen as a gateway to the game world.

For some of the programs, there is a real time link between the graphical and the textual representation of the algorithm. It means that if the user changes the source, the representation of the memory will be changed instantaneously. Scratch, Jeliot [16], and Alvis [17] are the only ones which propose this automatically, but it is also proposed on-demand in the other programs.

B. Code Structure

AlgoBox and Scratch variable declarations are very simple. The user just clicks on a button called "Declare new variable" or "New variable". There is no graphical representation of a variable in AlgoBox but Scratch illustrates the concept of a variable with a box whose value is adjustable with a sliding scale. With Algoris or LARP, there is no declaration. Whenever a variable is needed in a program, it is set without your having to give a type. With Alvis, the drag and drop of a new variable from the menu to the editor opens a window assignment. To learn the notion of type and variables, Alvis seems better.

The Scratch conditional instructions seem the simplest and most visual implementation. The flowchart representation is quite visual on this point. One can easily understand how the algorithm works and what instructions are to be executed.

In Algobox, there is a button to create conditional instructions. Everything is explained in the initialization window, where specifications can be added at request.

With Alvis, there is no a graphical representation of the conditional instruction, but we are guided step by step to build it. Alvis helps the user in the establishment of a simple conditional instruction, but you have to modify the code to create a compound one. Finally with Algoris there are two boxes for "if" and one for "if / else".

The representation in Scratch or LARP is quite visual, it so it is easy to understand the course of a conditional instruction. Furthermore the definition of the conditions of the conditional instructions is designed like a puzzle.

The representation of loops in Scratch is interesting. They are represented in the form of pliers in which the instructions to be repeated are set. The problem is that the syntax is not usual, which is a problem for learning algorithms. Yet the visual aspect of the loops makes it easy to understand that they are the instructions to be repeated.

LARP is based on the usual algorithmic syntax, it's easy to understand what and each loop how does, but with flowcharts it is not easy to recognize what kind of loop it is.

With AlgoBox, there is still nothing graphic, but the initialization window is suitable and all the necessary explanations are given. For the loop, the variable that is chosen will be used as a counter. The starting and final values can be chosen by the user but not the increment which is necessarily of one.

In LARP, the code generation of a loop is well explained thanks to the initialization window where you define all the parameters available. The arrangement of loop representation in Scratch and Algoris helps to understand the sequence of loops.

All this algorithm and code editors, although very different by their languages or representations, are helpful for the user as they provide a representation of the code and a better highlighted syntax.

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-177-9

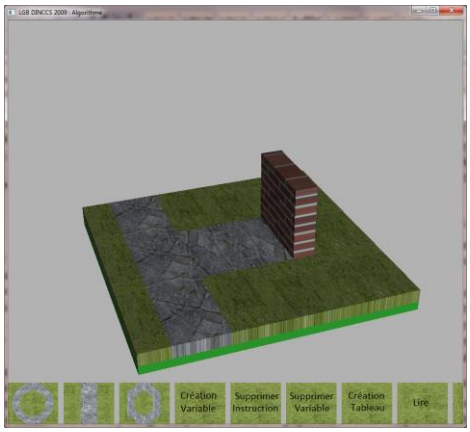

Figure 1. An empty algorithm in AlgoPath.

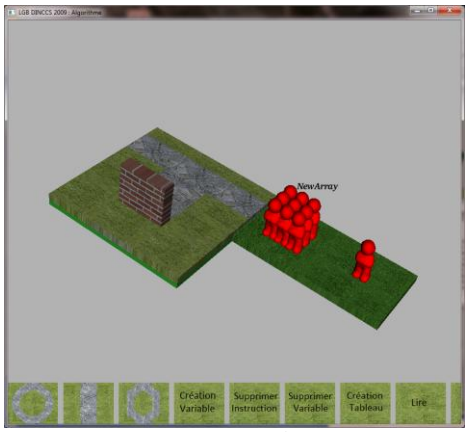

Figure 2. Creation of an array

Graphical representation or model is a good methodology to understand and interpret algorithms [18]. For some of them, the learning of a programming language is simplified by creating a new language between code and algorithm.

III. ALGOPATH

The main advantages of AlgoPath are:

- AlgoPath is attractive. Students face a virtual world which is like a game environment.
- AlgoPath helps them learn because it gives a mental representation of an abstract concept that is the algorithm.
- AlgoPath helps them avoid some common mistakes. Students can forget the off-putting syntax and grammar.
- AlgoPath lets them focus on the solving of the problem they have to automate without syntax or grammar errors. You will see that, at the end, a 3D figure just has to carry the proper value.
- AlgoPath changes the level of difficulty. It is easier to learn and to understand [19].

In AlgoPath, algorithms are a world reduced to a stone path lined with grass and trees (see Figure 1). In such a state, the algorithm represented by AlgoPath simply does nothing.

In the next sections, we will show how variables and basic instructions are represented and the last section will be dedicated to parameters passed by value or by reference.

A. AlgoPath Variables

In AlgoPath, a variable is a 3D figure (see Figure 2.). The user always begins with virtual declarations of variables because she/he cannot do anything else. Actually, no instruction can be added if no variable has been created. To do so, she/he just has to create a new 3D figure and give it a name. Then a new 3D figure stands in a platform next to the stone path lined with grass and folds its arms across its chest. Why does the 3D figure seem unoccupied?

It appears so because it has not been set yet. As expected, several instructions are available and are described in the next sections:

- The assignment.
- The conditional instructions.
- The loop instructions.

An array can also be created by the user of AlgoPath. For teaching purposes, only one dimensional and two dimensional arrays are available in AlgoPath. In that case, the array is shown by multiple 3D figures standing right next to the others.

B. AlgoPath Assignment

The assignment sets or re-sets the value associated to a variable. In AlgoPath, the user starts with specifying the right-value of the assignment with the keyboard and the mouse and then links it to a 3D figure. During this step, she/he just has to click on the appropriate 3D figure the software shows. An ordinary mistake our students do is to invert the right value with the left-value of an assignment. This cannot happen in AlgoPath. They also cannot use an unset variable in a right-value because the software only shows them the set variables during the specification of the right-value.

Once the variable has been set, the 3D figure carries a backpack. It also walks along the stone path (see Figure 3.). The difference between the green platform and the one with the stone path is that the green platform contains all the 3D figures that can be used in an instruction. The fact that a 3D figure standing in the green platform carries a backpack reminds the user that the associate variable has already been set.

When assigning an array, the user clicks on the corresponding 3D figure but this interaction is not sufficient. She/he also needs to choose which slot of the array is assigned. During this stage, the software shows the user multiple 3D figures: one for each slot of the array. The names of these 3D figures are the name of the array concatenated to their position in the array (zero stands for the first slot).

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-177-9

Figure 3. A set variable and its assignment.

At any time, the user can double-click on the 3D figure and see the right value of the assignment.

C. AlgoPath Conditional Instructions

AlgoPath conditional instructions change the stone path (see Figure 4). Instead of being a straight line, it splits into two paths. There is a traffic light at the intersection of the paths which stands for the condition of the conditional instructions.

If the algorithm could be executed and the condition turned out to be true, the traffic light would turn out to be green. If the condition turned out to be false, the traffic light would turn out to be red. That is why, one path is covered with a green bush, the other with a red bush.

After the creation of a conditional instruction, the user has to click on both the green and the red bushes. By doing so, she/he will have to complete the unconditional instructions and the conditional instructions. At each step, a new empty stone path is shown and has to evolve. If she/he creates a new variable when the shown stone path is one of the conditional or unconditional instructions, then the associated 3D figure is only available in the conditional or unconditional instructions.

It means that the user will no longer see it when she/he leaves the stone paths of the conditional instruction. This specification helps the students learn the notion of the scope of a variable. They are able to see it before any program is run.

As for the right value of an assignment, the user can see the condition (by double-clicking on the traffic light), the conditional instructions (by double-clicking on the green bush), and the unconditional instructions (by double-clicking on the red bush) of a conditional instruction at any time.

D. AlgoPath Loop Instructions

As an AlgoPath conditional instruction, an AlgoPath loop instruction changes the stone path (see Figure 5). It becomes a two-entry traffic circle. At one of these two entries stands a traffic light. The position of the traffic light specifies the type of the loop. If the traffic light is at the beginning of the loop, the instructions of the loop may never be executed (it is a while loop). If the traffic light is at the end of the loop, a walker could be free to enter the loop. It means that the instructions of the loop could be executed at least once (it is a repeat loop).

After the creation of a loop instruction, the user has to double-click on the bush of the traffic circle. By doing so, she/he has to complete the instructions of the loop. Then a new empty stone path is shown and has to evolve. During this step, if a new variable has been created, the related 3D figure is no longer known when the user leaves the loop stone path. Again, the student faces the notion of the scope of a variable.

E. AlgoPath Input and Output

AlgoPath has two more instructions. One if the future user's algorithm wants to see the value of a variable (a slate is added in the stone path) and another it the future user's algorithm has to set a variable (the 3D figure shows an open book).

Figure 4. An AlgoPath conditional instruction.

Figure 5. An AlgoPath loop instruction.

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-177-9

F. Passing data to procedures and functions

In programming, there are two ways of passing data to procedures and functions: either by value (in that case, parameter in the call of a procedure or a function does not share memory with the parameter in the definition of a procedure or a function) or by reference (in that case, they share memory). In AlgoPath, students are able to visualize this process. As passing data by value passes a copy of the data for processing and as a data is represented by a 3D figure carrying a backpack, parameters by value are visualized by two 3D figures standing in front of each other, the first one showing its backpack to the second one, and the second one carrying its own backpack. On the contrary, as passing data by reference saves changes made to data and passes those changes back to the calling algorithm, parameters by reference are visualized by two 3D figures facing each other and holding the same backpack (see Figure 6).

The call of a function or a procedure is represented by a forest. Multiple 3D figures stand at the edge of the forest. They are the parameters of the function or the procedure. For educational purposes, a function never has parameters by reference but obviously passes a result to the calling algorithm. This result is represented by a 3D figure that stands at the exit of the forest. On the contrary, a procedure does not need such a 3D figure but may have parameters by reference.

When the user creates a new function or procedure, she/he must:

- Specify the name of the procedure or the function. With that name, AlgoPath knows if the user will have to create the definition of the function or the procedure in the next step.
- Specify the parameters. If the function or the procedure has already been defined, AlgoPath shows the user the parameters of the called algorithm. The user just has to drag and drop the correct 3D figures in front of the calling parameters. If not, AlgoPath expects the user to drag and drop the correct 3D figures and to choose the correct 3D figure that must stand in front of them to choose a parameter by reference or by value. If she/he chooses a 3D figure with its hands wide open then the parameter is by reference. If she/he chooses a 3D figure which carries its own backpack then the parameter is by value. If it is a function to be defined, a special calling parameter is automatically created with its hands wide open. It is the result of the function.
- Specify the instructions. This step is optional if the procedure or the function has already been defined. If not, the user has to double-click on the forest and a new empty stone path appears. In that case, the platform next to the stone path is already full of 3D figures that correspond to the calling parameters. If the calling parameter is by value then the called parameter has given it its value. So the relative 3D figure carries a backpack. If the calling parameter is by reference then the called parameter gives it its state. If the called parameter is set, the relative 3D figure carries a backpack. If it is not, it does not. The calling parameter representing the result of a function always folds its arms across its chest at the beginning of the definition of a function and means that it has to be set. AlgoPath does not allow the user to go back to the called algorithm if there is one 3D figure with its arms across its chest left. At any time, the user is able to change the type of a parameter either by double-clicking on the 3D figures at the edge of the forest or by double-clicking on the platform next to the stone path.

IV. IMPLEMENTATION

AlgoPath has been developed in C++ language using the VTK (Visualization Tool Kit) library. 3D figures have been made in clay and digitized into STL files using the 3DSom Software (see Figure 7). They have been smoothened and enlarged using Gom Inspect and 3D Reshaper.

A more complete example of an algorithm can be seen in Figure 8.

V. CONCLUSION AND FUTURE WORK

This paper shows that it is possible to translate algorithms into a virtual world where 3D figures are data storage locations, backpacks carried by 3D figures are values and, bushes and forests are a set of instructions and paths are the way instructions are supposed to flow.

We have also shown that AlgoPath can help students avoid common mistakes (such as using a variable in an expression before it is set or forgetting to assign a parameter passed by reference) through an easy and intuitive interface. It helps them learn because AlgoPath gives them a way to have a mental representation of algorithms. AlgoPath is ludic because it is a world close to games and students feel they are playing more than learning. As a consequence, it is easier to learn algorithmic with AlgoPath.

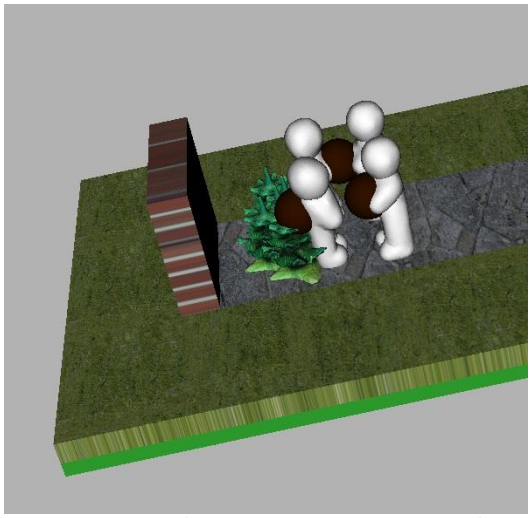

Figure 6. Parameters by reference (in back) and by value (in front).

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-177-9

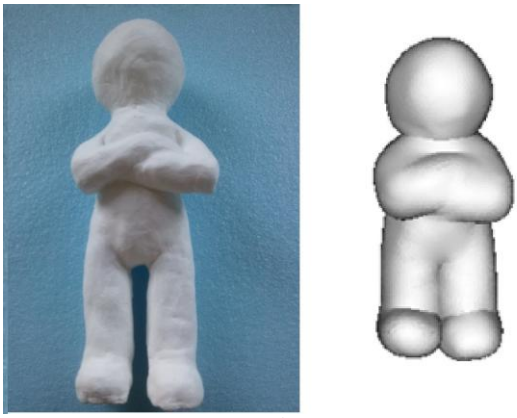

Figure 7. Clay 3D figure and the virtual one.

However, some improvements need to be done. In particular the results of the ongoing user study may point inconsistencies in the interface or in the process.

Anyhow this process must be extended by adding an execution that could be simulated by a bus taking on board every 3D figure along the stone path and delivering them at the end of the algorithm with the correct values.

AlgoPath could turn into a more serious game if the wanted algorithm was chosen by the teacher and was a data of AlgoPath so AlgoPath could help the students find the correct algorithm.

ACKNOWLEDGMENT

We thank Marie François and Estelle Bodart for their valuable help during the study and the implementation of AlgoPath.

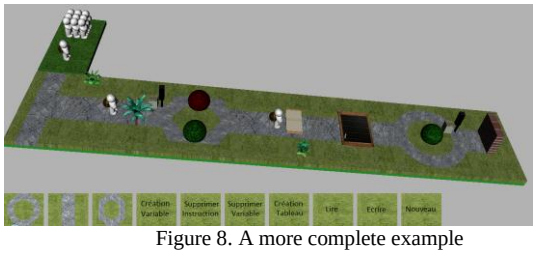

Figure 8. A more complete example

REFERENCES

[1] http://www.mediaawareness.ca/english/resources/research_documents/studies/video_games/vgc_preferences.cfm

[2] Ronit Ben-Bassat Levy, Mordechai Ben-Ari, Pekka A. Uronen, "The Jeliot 2000 program animation system", Computers & Education, vol. 40 (1), pp. 15–21, 2003

[3] Druckman D., "The Educational Effectiveness of Interactive Games. Simulation and gaming across disciplines and cultures: ISAGA at a watershed", Thousand Oaks: Sage, pp. 178 – 187, 1995

[4] Alan Amory, Kevin Naicker, Jackie Vincent, Claudia Adams, "Computer games as a learning resource", ED-MEDIA, ED-TELECOM 98, world conference on education multimedia and educational telecommunications, vol. 1, pp. 50–55, 1998.

[5] Cristina Conati, Xiaoming Zhou, "Modeling students' emotions from cognitive appraisal in educational games", Intelligent Tutoring Systems, pp. 944–954, 2002

[6] Viswanath Venkatesh, "Creation of favorable user perceptions: exploring the role of intrinsic motivation", MIS Quarterly, vol. 23, pp. 239-261, 1999

[7] http://www.mikesoft.fr/math/Algoris/Logiciel%20algoris.htm

[8] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, Y. Kafai, "Scratch: Programming for All", Communications of the ACM, vol. 52 (11), November 2009

[9] M. Resnick, "All I Really Need to Know (About Creative Thinking) I Learned (By Studying How Children Learn) in Kindergarten", Proceedings of the SIGCHI Conference on Creativity and Cognition, Washington, D.C., 2007

[10] Norma S. Pribadi, Maria G. Wadlow, Daniel Boyarski, "The use of color in computer interfaces : Preliminary Research", Information technology Center, Carnegie Mellon University,Pittsburgh, Pennsylvania, 1990.

[11] C. Kelleher Dennis and D. Cosgrove and D. Culyba and C. Forlines and J. Pratt and Y Pausch, Alice2: Programming without Syntax Errors, 2002

[12] http://larp.marcolavoie.ca/fr/default.htm

[13] http://www.xm1math.net/algobox/

[14] Mordechai Ben-Ari, Roman Bednarik, Ronit Ben-Bassat Levy, Gil Ebel, Andrés Moreno, Niko Myller, Erkki Sutinen, "A decade of research and development on program animation: The Jeliot experience", Journal of Visual Languages and Computing, vol. 22, oct. 2011, pp. 375-384, 2011

[15] Hundhausen, C.D. & Brown, J.L. What You See Is What You Code: A "Radically-Dynamic" Algorithm Visualization Development Model for Novice Learners." Proceedings IEEE 2005 Symposium on Visual Languages and Human-Centric Computing. Piscataway, NJ: IEEE.

[16] Andrés Moreno, Niko Myller, Erkki Sutinen, M. Lattu, V. Meisalo, J. Tarhio "A visualization tool as a demonstration aid", Computers & Education 41(2), pp. 133–148, 2008

[17] Christopher D. Hundhausen, Jonathan L. Brown, "What You See Is What You Code: A ''live'' algorithm development and visualization environment for novice learners",Journal of Visual Languages & Computing, vol. 18 (1), February 2007, pp. 22-47, 2007

[18] Richard E. Mayer, "Different problem-solving competencies established in learning computer programming with and without meaningful models", Journal of Educational Psychology, vol. 67, pp. 725-734, 1975

[19] John Maloney, Kylie Peppler, Yasmin B. Kafai, Mitchel Resnick, Natalie Rusk, "Programming by Choice: Urban Youth Learning Programming with Scratch", SIGCSE'08, March 12–15, Portland, Oregon, USA, 2008

Copyright (c) IARIA, 2012. ISBN: 978-1-61208-177-9