USER-CENTERED COMPUTER SCIENCE
High-ceiling and Low-floor Approaches to Programming Languages and Algorithms

Stefano Federici
Dept. of Teaching and Philosophy, University of Cagliari, via Is Mirrionis, 1, Cagliari, Italy

Keywords: Computer science, Computational science, Algorithms, Algorithm animation, Sorting, Scratch, BYOB.

Abstract: Understanding computer science algorithms is one of the steepest obstacles when learning computational science. In this paper I will describe a novel approach to learning standard programming languages and basic computer science algorithms that is based on BYOB, one of the more relevant extension of Scratch, a programming tool developed by MIT. In the proposed approach, students can build several algorithms by themselves without having to cope with all the knowledge about programming usually needed when using a standard programming language.

1 INTRODUCTION

Computational Science education is recently getting a lot of attention from the US government: the President's Information Technology Advisory Committee (PITAC, 2005), the Defense Advanced Research Projects Agency (DARPA, 2010), the Association for Computing Machinery and the Computer Science Teachers Association (ACM and CSTA, 2010) all insisted that a large effort must be put in increasing the strength of current educational practices in computer science. The reason is that from 2000 to 2007 enrollments in Computer Science (CS) courses in United Stated dropped by more than 60%. And this despite the fact that "Computational science – the use of advanced computing capabilities to understand and solve complex problems – has become critical to scientific leadership, economic competitiveness, and national security" (PITAC, 2005).

One of the prominent reasons of this failure is that computer science programs are for the most part still anchored to what computer science was twenty years ago: bits, bytes, Pascal, C, Assembler.

If these topics are all relevant to what the core of CS is (computation, programming, automation) there is no reason why they should be used still today to introduce students to computer science.

The -maybe surprising- fact is that there is no need to teach computational skills by using this old-fashioned path from bits to, for example, the C programming language. All computational concepts can be taught today in a much easier way by using the newest introductory tools to computation developed by outstanding research centres as the Massachusetts Institute of Technology, the Carnegie Mellon University, the University of Kent. All these tools focus on the creative potential behind computation by lowering the minimum level that allows students to get interesting results. In parallel, interesting visualization tools have been developed in recent years to help students in understanding one of the more complex parts of computational science: algorithms. What is still missing is the point in which the new introductory tools to computation meet the theory of algorithms and complexity.

In this paper I will briefly survey the newest tools for introduction to computer science, then, after discussing the topic of visualization tools, I will describe a first effort in developing a new environment that can introduce students in natural way to understanding computation and complexity.

2 NEW ENVIRONMENTS FOR TEACHING AND LEARNING CS

One of the major obstacles to make students appreciate computer science has been until recent

Federici S..
USER-CENTERED COMPUTER SCIENCE - High-ceiling and Low-floor Approaches to Programming Languages and Algorithms .
DOI: 10.5220/0003347903400348
In *Proceedings of the 3rd International Conference on Computer Supported Education* (CSEDU-2011), pages 340-348
ISBN: 978-989-8425-49-2

years the very *high-floor* and *low-ceiling* of previous approaches to the teaching of programming languages, that is, while the entry level of knowledge and effort to accomplish anything is high, the meaningfulness of the output is very low.

The very first example that every student would find in their books is the famous "Hello World!" program used by Kernighan and Ritchie in their "The C Programming Language" (Kernighan and Ritchie, 1978):

```
#include <stdio.h>
main() {
   printf("hello, world\n");
}
```

This code will simply display the words "hello, world" on the computer screen. But even understanding this minimal program requires the students to get to a *high floor* by acquiring a lot of esoteric knowledge about libraries, main programs and procedures, programming languages syntax, functions, strings, and control characters. Getting appealing results with programs written in the C programming language is far from the possibilities of an introductory CS course. In this sense CS based on C is really *low-ceiling*: you can't go very far when trying to get interesting results.

Things didn't get better when C was replaced by Java. The "Hello World!" program written in Java is even more complex, involving also knowledge about classes and object oriented programming:

```
class HelloWorld {
  public static void main(String[]
                                 args) {
    System.out.println("Hello
                              World!");
  }
}
```

If the floor of Java is even higher, at least the ceiling is a bit higher too compared to C: a lot of free and open source libraries are available in Java and the interested student can get interesting results by applying itself to the discipline. But this is exactly the point: before the teacher will be able to show to the students how to get appealing results, the interest of the most part of the students will have already faded out. That is why specific tools to introduce students to programming have been developed in recent years and have started succeeding in getting students interested in computer science. Just to complete the "hello world!" example, this is the program that gets the same result (that is showing the "hello, world" words on the screen) when using Scratch (Resnick et al., 2009):

Figure 1: "Hello, World!" program in Scratch.

There is no need to know about what programming is and to know how computer programming evolved from very slow and expensive computers to very fast and cheap netbooks.

In the following, brief survey we will review the more well-known and successful tools for teaching CS today, namely Alice, Greenfoot, StarLogo TNG and Scratch.

2.1 Alice

Alice has been continuosly developed at Carnegie Mellon University since 1999. The purpose of Alice (http://www.alice.org/) is to offer to CS students the possibility to create some very nice 3D animations and programs in a "movie director" way and at the same time to learn the basic computational concepts.

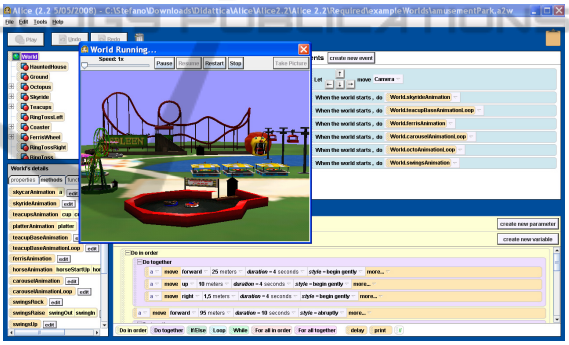

Figure 2: The *Alice* programming environment.

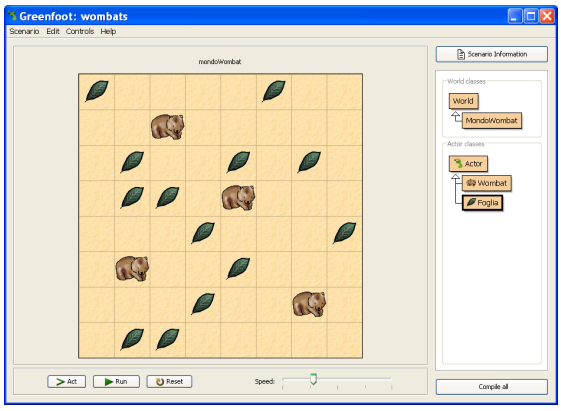

Figure 3: The Greenfoot programming environment.

2.2 Greenfoot

Greenfoot (http://www.greenfoot.org/) has been developed by researchers of the University of Kent

and Deakin University since 2006. The main goal of Greenfoot is to introduce students to the Java programming language with a smooth transition.

2.3 StarLogo TNG

StarLogo TNG (http://education.mit.edu/projects/starlogo-tng), by Eric Klopfer of MIT (Klopfer, 2008) has evolved from of StarLogo. StarLogo, in its turn, is an extension of the Logo programming language (Harel and Papert, 1991) created by Seymour Papert in 1967. It can be used to create 3D animations and programs similar to the ones that can be developed with Alice.

Figure 4: The StarLogo TNG programming environment.

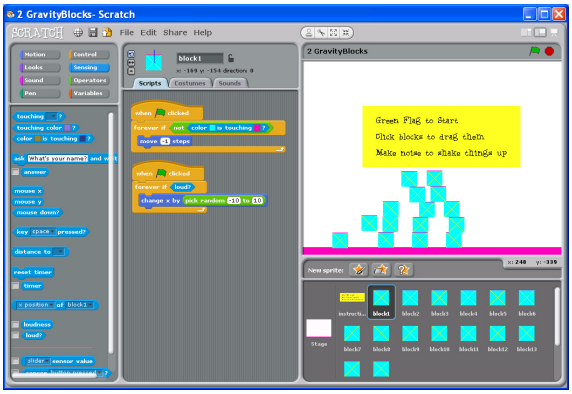

Figure 5: The Scratch programming environment.

2.4 Scratch

Conceived in 2007 by Mitchel Resnick, director of the Lifelong Kindergarten Group of MIT, Scratch (http://scratch.mit.edu/) is the youngest and the most successful of programming environments designed for people that have never successful coped with computer programming.

The Scratch philosophy encourages people to study projects created by other members of the Scratch community (composed of more than half a million people) and remix them to create their own projects. In 2 years, the number of open source projects freely available for downloading from the Scratch website rose from less than 70.000 to more than 1.3 millions.

2.5 Weaknesses and Strengths

Scratch is at present by far the most used among all programming environments for real beginners. Even if the projects created with Scratch cannot be as complex as the ones that can be created with the other environments (Alice, Greenfoot and StarLogo TNG all have the performance of Java behind them) nonetheless the appeal of Scratch is by far the highest, that is, the ceiling for Scratch may not be as high as for the other languages, but the floor is consistently lower.

Greenfoot is really a simplified programming environment specifically designed for Java (so that you have to know a good part of the Java programming language before taking advantage of it) and has a quite high floor. The reason for the success of Scratch when compared to Alice and StarLogo TNG is in the simplicity of the programming environment of Scratch.

3 SCRATCH: AN OUTSTANDING PROGRAMMING ENVIRONMENT

The programming environment of Scratch indeed is really essential: any non-programmer can start programming Scratch without even noticing that programming is taking place.

3.1 A Start-and-Go Programming Environment

The interface of Scratch is basic: essential menus, just a few icons. Nonetheless, all the important elements are visible: the cat sprite on the stage on the right hand side, a list of colored blocks on the left hand side. It is natural trying to click one of the colored blocks and seeing that the effect of what is written on the block immediately transfers to the sprite on the stage.

All the blocks are visible in the left pane, clearly organized in logic categories (movement, look, sound, pen, control, sensing, operators, variables) each identified by a different colour.

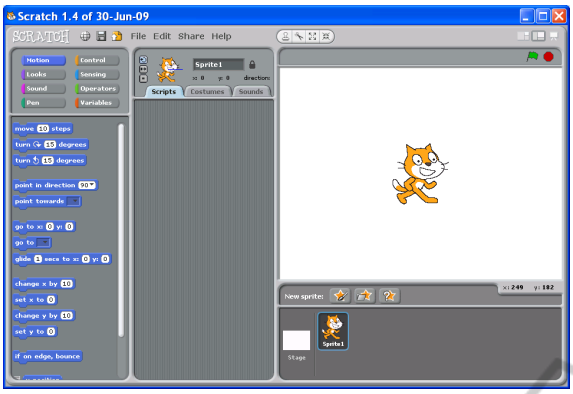

Figure 6: An empty Scratch project.

3.2 Natural Language Programming

Another element of great value of Scratch is its ability to model each "instruction" (each colored block) in a very natural wording: "move 10 steps", "change score by 1", "point towards mouse-pointer". As in Alice and StarLogo TNG, the student does not need to know in advance the programming language or its syntax. All blocks are indeed available in the block repository on the left hand side. The way in which blocks can combine together is constrained by their forms: only when a white halo appears around a free slot the dragged block can stick to the other blocks. All this prevents syntax errors. So students have only to pay attention to the meaning (the semantics) of what they are doing.

3.3 A Multilanguage Programming Language

Scratch, the language as well as the interface, is available in 50 different languages spoken worldwide, lowering the floor for non-English speaking people, who often have to learn a foreign language (English) at the same time that they are learning a new, formal way of expressing their knowledge.

3.4 A Very Low Floor

All these features greatly lower the floor, in terms of base knowledge, that a non programmer has to reach when confronted with programming: very little knowledge is necessary to get interesting results, when compared to standard programming languages. All the difficulties (unknown user interface, unknown formalism, awkward syntax, syntactic errors) have been carefully avoided by a thoughtful design.

Variables, one of the most difficult element of programming languages, are often not needed in Scratch. The basic sprite properties (position and direction, size, costume, volume of the sprite's "voice") are all available as "monitors", rounded blocks whose values can be immediately shown on the stage by clicking their checkboxes.

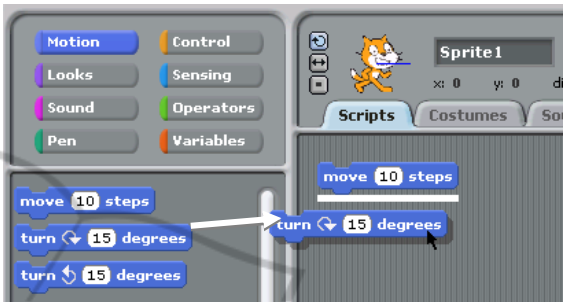

Figure 7: Snapping blocks together. The white line between the "move 10 steps" block and that the "turn 15 degrees" block means that the two blocks can fit together.

As highlighted above, the shape of the instructions of the Scratch programming language (the Scratch blocks) is reminiscent of the shape of Lego blocks. Scratch (and also Alice and StarLogo TNG) is a drag-and-drop programming language. The blocks can be snapped together by dragging and dropping them on the central grey area, the "Script" area (Figure 7). The script area is where Scratch programmers create their programs. Scratch's sprites are indeed like actors that act on the stage by following the scripts provided by the scriptwriter (the programmer). To run the scripts it is possible to click them or to click the green flag button at the top right corner of the Scratch window.

3.5 A Reasonably High Ceiling

Despite its simple interface, Scratch can be used to create a lot of interesting projects very easily.

Even if Scratch has no 3D capabilities and the speed of its interpreter is slow, having an embedded image editor, a webcam capture tool, a sound recorder, the possibility of connecting projects over meshes, to send to and receive messages from other sprites, to add special graphic effects (colour, fisheye, wheel, brightness, ghost, etc), all this make designing even complex projects about music, art, simulation, games and a lot more really very simple.

Creating even a full chat project, like the one available at http://www.sitibs.com/branches/moodle/mod/forum/discuss.php?d=1499 (Federici, 2010), or a complex videogame like Donkey Kong (http://scratch.mit.edu/projects/Deweybears/20922) is

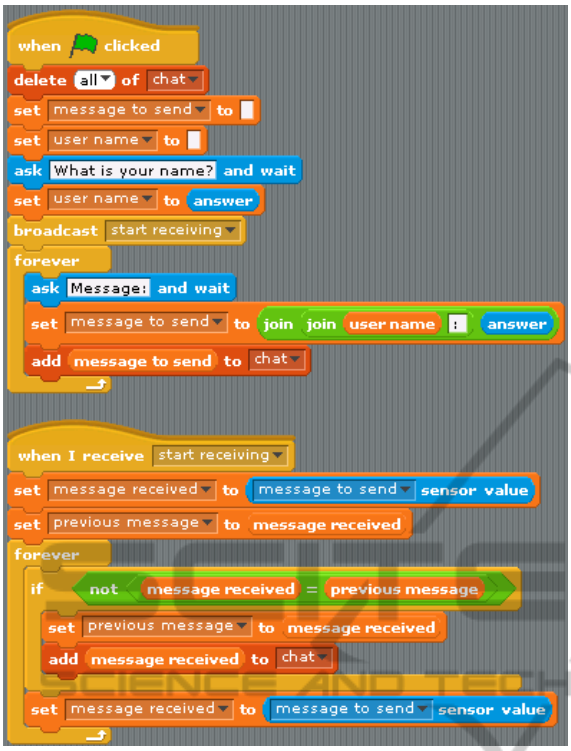

Figure 8a: A full bidirectional chat implemented in Scratch (code). Two PCs running Scratch are connected via "sensors" over a mesh.

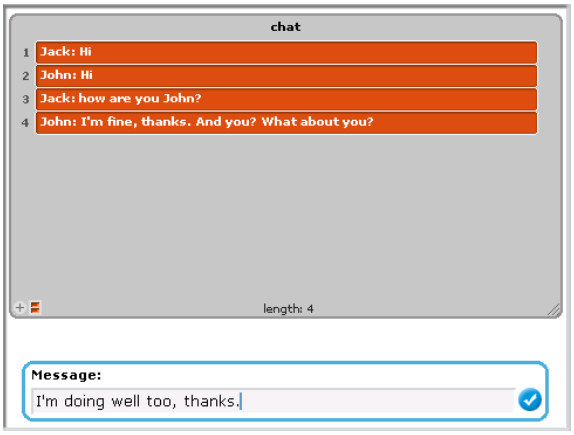

Figure 8b: A full bidirectional chat implemented in Scratch (screenshot).

really withing reach of begininng programming students.

Another feature of Scratch that is motivating to students is the possibility of publishing Scratch projects on the Scratch website just by clicking the "Share this project" button in the top left corner of the Scratch interface. Projects are then immediately usable online running inside a Java applet or (in beta testing at the time of writing) as a Flash application.

The future of Scratch 2.0 is indeed the mobile world. Besides the excellent port to iPhone (that has been temporarily removed from iTunes due to recent changes in Apple policy about legal applications), the new programming environment and player is being currently completely rewritten in Flash (http://wiki.scratch.mit.edu/wiki/Experimental_Viewer).

Furthermore, new versions of Scratch implemented in Javascript (http://github.com/mitsuhiko/iskratch/) and for the Android operating system and Nintendo DSi devices are currently under development (Slany, 2010).

4 FROM SCRATCH TO BYOB

Being designed to help people to get closer to computational concepts, Scratch designers have kept the Scratch language as simple as possible on purpose.

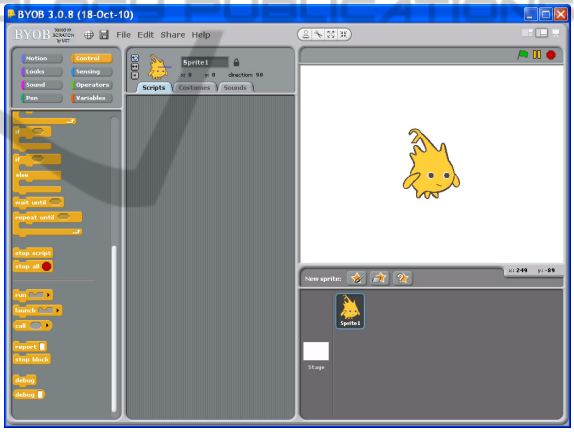

Figure 9: The BYOB environment.

So a lot of interesting things that could be done by using the extraordinary Scratch interface are not available: Scratch does not have the possibility to define new blocks, to read and write files, to have a full control on the elements of the tool. Even lists and string input have been added to Scratch only recently.

But Scratch is open source, so the language and the tool can be extended by proficient programmers of Smalltalk, the programming language used by Scratch's creators. This has led to the creation of a full variety of Scratch extensions, called "mods", that make available to Scratch programmers a lot of different features (a full list of mods is available at http://scratchmods.weebly.com/index.html).

The first and most famous among all mods is BYOB (http://byob.berkeley.edu/), an extension that added to Scratch the possibility of defining new blocks, to handle higher-order functions and to debug projects (Harvey and Monig, 2010).

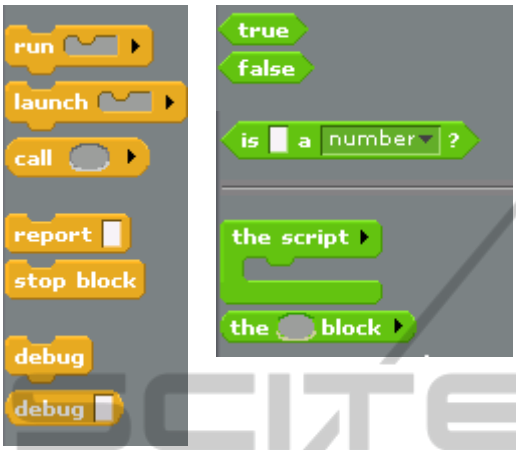

Figure 10: New BYOB's blocks.

Thanks to BYOB it is now possible to develop libraries of Scratch blocks that are seamlessly integrated in the original block categories. So, again, all the available instructions are readily available to the BYOB programmer, without having to know what is in the library in advance. All this by adding to Scratch just 12 new blocks (figure 10).

Thanks to the hard work of the Scratch community, the team in charge of developing the official releases of Scratch is getting feedback, help and support to make Scratch evolve in the directions that are considered mostly important to the community, at the same time keeping the language and the environment faithful to the original philosophy.

5 USER-CENTERED CS

The possibilities added by BYOB to Scratch are extremely powerful: a complete new language can be designed and developed in a relatively easy way even without having to be a proficient Smalltalk programmer.

In the next sections we will see examples of the two missing steps that have been developed with BYOB: the *miniC++* drag-and-drop programming language (http://www.sitibs.com/branches/moodle/mod/forum/discuss.php?d=1607) to learn C without the burden of C syntax and the *Animated Sort Library* (Federici and Stern, 2011) for understanding sorting algorithms in a way inspired by the excellent Algorithms in Action tool developed at the University of Melbourne (Stern et al., 1999).

5.1 MiniC++: a Minimal Drag-and-Drop Implementation of the C Programming Language

Brian Harvey supplied an extensive library "tools.ypr" included in the BYOB distribution.

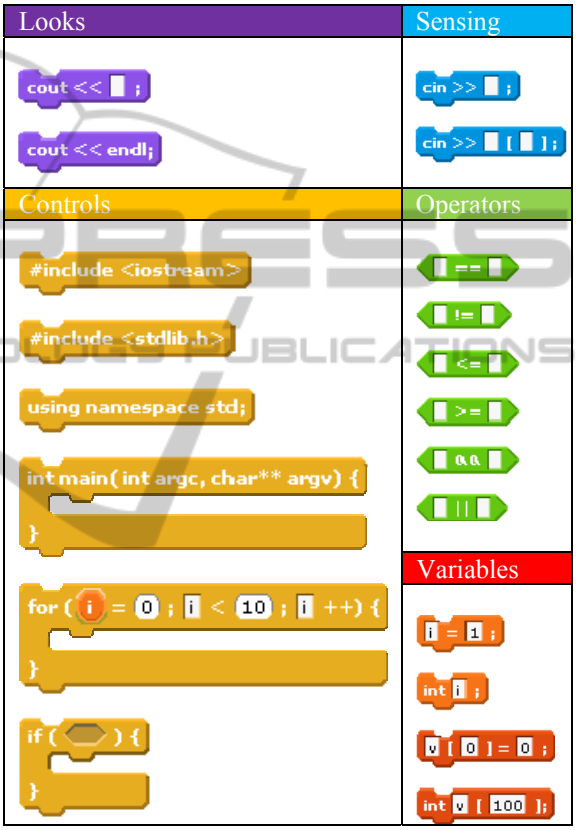

Figure 11: Some of the new blocks for mini C++.

By extending this library a full set of new blocks have been added to the standard categories already available in Scratch (figure 11):

- **cout** in the *Looks* category (output)
- **cin** in the *Sensing* category (input)
- **include, switch case, do while, for, if, if else, main, repeat until, using, while** in the *Control* category (control structures)
- **!=, ==, >=, <=, &&,** || and mathematical operators in the *Operators* category
- = (assignment), **int**, **[ ]** (array setting and access) in the *Variables* category

The new blocks are available in the same left pane as the standard Scratch blocks, so that there is no need to know in advance which they are.

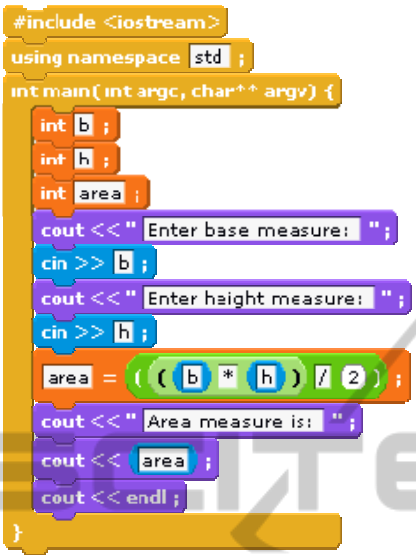

Figure 12: miniC++ program to calculate the area of a triangle.

By dragging and dropping this blocks in the script area it is possible to build simple C++ programs (figure 12).

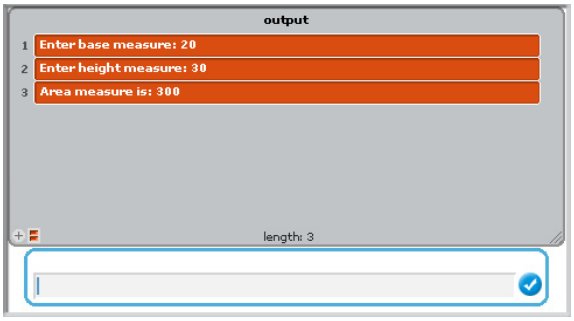

Figure 13: miniC++ program input/output.

MiniC++ programs behaves exactly as expected, showing the program output and asking for their inputs on the stage (figure 13).

5.2 An Animated Sort Library

One of the most successful technologies used to teaching basic algorithms (searching, sorting, graph algorithms, etc) is algorithm animation (Naps et al., 2002). Algorithm animation has been used for more than ten years and several excellent tools have been developed to allow teachers to build animations for their students. What is missing in this tools is a simple way of linking an algorithm developed by the student to the actual animation, a level of interaction defined as "constructing" (Greyling, 2009).

To overcome this gap, an Animated Sort Library has been developed in BYOB to provide a library of new blocks that, operating on several balls of different sizes on the stage and managing two magnets that get the balls and put them back, allow the student to implement several sorting algorithms (Federici and Stern, 2011).

5.3 Algorithms and Data Structures in miniC++

The last step of this excursus into a new curriculum for computer science based on drag-and-drop languages and simplified approaches is the full implementation of the Basic Sort algorithm in miniC++.

To understand how close the C++ algorithm for Basic Sort is to its drag-and-drop implementation let us have a loot at the "real" C++ algorithm (Figure 14a). Once every element of the C implementation of the Basic Sort algorithm using arrays and variables have been understood (inclusion of libraries, namespaces, main program, types, arrays, variables, input/output, loops, syntax) the miniC++ implementation (figure 14b) is straightforward.

From this point on, moving to a "real" programming language such as C or Java is only a problem of getting rid of all the possible syntactic errors: all the semantic part has indeed already been acquired by the student in advance.

6 EFFECTIVENESS OF THE APPROACH

Even students of university courses in CS can benefit by the creation of new, ad-hoc, drag-and-drop programming languages by grasping all the necessary concepts in a stepwise process.

They can start by concentrating on the important concepts of computer programming by using Scratch without being overwhelmed by the subtle and difficult elements that they should face if confronted with everyday programming languages. After having grasped the most basic concepts, students could then move to the corresponding instructions of regular programming languages by means of ad-hoc, drag-and-drop implementations of their preferred language. They will make use of the same basic concepts already learnt in Scratch without having to

```
#include <iostream>
using namespace std;
int main(int argc, char** argv) {

    int v[100], n, i, j, a;
    cout << "how many values? ";
    cin >> n;

    for( i=0; i<n; i++) {
        cout << "type a number: ";
        cin >> v[i];
    }
    cout << endl;

    for( i=0; i<n; i++) {
        for( j= i+1; j<n; j++) {
            if( v[i] > v[j]) {
                a = v[i];
                v[i] = v[j];
                v[j] = a;
            }
        }
    }

    for( i=0; i<n; i++) {
        cout << " value at position"
        << i << ":" << v[i] << endl;
    }
}
```

Figure 14a: C++ implementation of Basic Sort.

face the specific details of the syntax of the language. By using a simplified approach to study the -otherwise really complex- theory of data structures and algorithms (searching, sorting, graph algorithms, etc) and by making use of animation and visualization tools, students can actually *see* how the algorithms prepared for them by their teachers work. Even more importantly, students can *design* their own algorithms and look at how they work. Finally the students could see how the same algorithms and data structures can be implemented in a real programming language by using again a drag-and-drop implementation of the chosen language (figure 14b).

After having extensively used drag-and-drop programming languages and visualization tools, the students could then move to a user-friendly development environment for a regular programming language (such as, for example, Greenfoot for Java) and finally move to full-fledged programming environments such as Eclipse (http://www.eclipse.org) or Netbeans (http://www.netbeans.org).

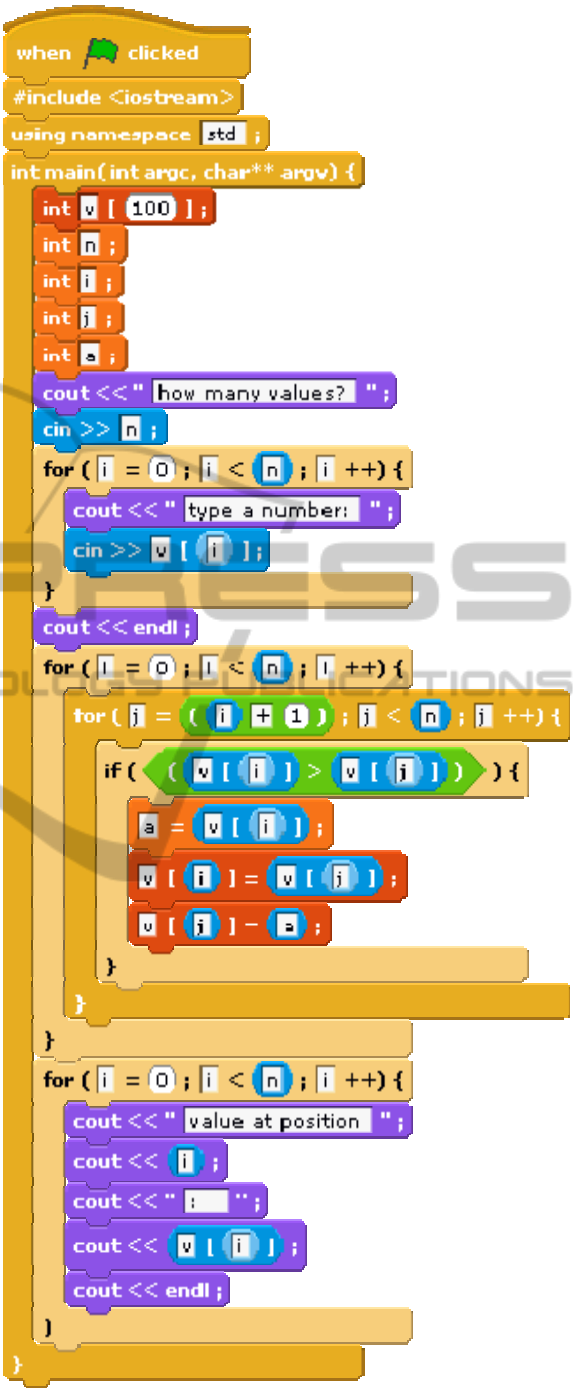

Figure 14b: miniC++ implementation of Basic Sort.

The effectiveness of Scratch-based approaches to introductory CS at different levels (k-12 to university) has been already confirmed in several studies (Malan and Leitner, 2007; Humphries, 2010; Rizvi et al, 2011). The miniC++ programming language and the Animated Sort Library can pave the way to a new full curriculum for CS. This tools

are currently under testing in two introductory CS courses at the faculties of Education and Engineering of the University of Cagliari. Preliminary results are encouraging. If they will be proved effective tools to better understand CS, a full implementation will be soon completed.

7 CONCLUSIONS

Thanks to new advances in the development of tools for the support of CS learning, designing a new curriculum for CS is today a concrete possibility.

In this paper I have shown that basic concepts of computer programming can be taught by using intuitive drag-and-drop environments that provide the low-floor and high-ceiling of Scratch and, at the same time, represents a smoother transition to higher level languages such as C++ or to very technical topics such as sorting algorithms.

Similar approches can be taken for other high level languages or for other technical topics such as concurrency or languages and compilers. Moving to real programming environments is only done when everything else has been perfectly assimilated.

REFERENCES

ACM, CSTA, 2010. Comments on America COMPETEs Act Reauthorization In Response to the House Science and Technology Committee Survey on K-12 STEM Education Programs. http://www.acm.org/public-policy/competes_survey_final.pdf.

Dann, W. P., Cooper, S, Pausch, R., 2005. *Learning to Program with Alice*. Prentice Hall. ISBN 0131872893. http://www.alice.org/.

DARPA, 2010. *Computer Science - Science, Technology, Engineering, and Mathematics (CS-STEM) Education Research Announcement (RA)*. DARPA-RA-10-03.

Federici, S., 2010. Computers and Computer Science in the Communication Age. Presentation at *Comunica senza frontiere: e-learning communication workshop & showcase.* Cagliari. http://www.com.unica.it/data/
comunica_ senza_frontiere_2010.pdf.

Federici, S., Stern, L., 2011. *A Constructionist Approach to Computer Science Education.* In Proceedings of Global Learn 2011, Melbourne..

Greyling, J., 2009. *Developing a Set of Requirements for Algorithm Animation Systems*. In International Journal of Computing and ICT Research, Special Issue, Vol.3, No.1, pp. 83-89. ISSN 1818-1139.

Harel, I., Papert, S., 1991. *Constructionism*. Ablex Publishing. ISBN 978-0893917869.

Harvey, B., Monig, J., 2010. Bringing 'No Ceiling' to Scratch: Can One Language Serve Kids and Computer Scientists? In proceedings of *Constructionism 2010*.

Humphries, T., 2010. CS0 Programming Course using Scratch. *Software Engineering Educators' Symposium*, Santa Fe, New Mexico.

Kernighan, B. W., Ritchie, D. M., 1978. *The C Programming Language (1st ed.)*. Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-110163-3.

Klopfer, E., Scheintaub, H., Huang, W, Wendel, D., 2009. StarLogo TNG: Making Agent Based Modeling Accessible and Appealing to Novices. In Komosinski, M., Adamatzky, A., *Artificial Life Models in Software*. Springer. ISBN 978-1-84882-284-9.

Kölling, M., 2009. *Introduction to Programming with Greenfoot*. Pearson. http://www.greenfoot.org/.

Malan, D. J., Leitner, D., 2007. Scratch for Budding Computer Scientists. In proceedings of *SIGCSE '07*. Convington, Kentucky, USA.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger , S., Velázquez-Iturbide, J. Á., 2002. Exploring the Role of Visualization and Engagement in Computer Science Education. In *Working group reports from ITiCSE on Innovation and technology in computer science education.*

PITAC, 2005. *Computational Science: Ensuring America's Competitiveness.* http://lazowska.cs. washington.edu/computational.pdf.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y., 2009. Scratch: Programming for All. In *Communications of ACM*, 11. http://scratch.mit.edu/.

Rizvi, M., Humphries, T., Major, D., Jones, M., Lauzun, H., 2011. A CS0 course using Scratch. CCSC Conference. *Journal of Computing Sciences in Colleges*, volume 26, issue 3.

Stern, L., Sandergaard, H., Naish, L., 1999. *A Strategy for Managing Content Complexity in Algorithm Animation*. ACM SIGCSE Bulletin, Volume 31, Issue 3.