Using Microworlds to Introduce Programming to Novices

Yannis Papadopoulos, Stergios Tegos
Department Of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
e-mail: {itpapado,stegos}@csd.auth.gr

Abstract—**Introduction to programming is a traditional and compulsory course for first year Computer Science students. However, research in the field of computer science education has highlighted that many students lack on problem solving and computational thinking skills. These issues provoke questions in the academic community as to whether the learning methodology of introductory to programming courses is appropriate. As regards the pedagogical procedure in general, the teacher-centered model was amended with the introduction of Interactive Learning Environments, which promote distributed learning. In this paper, we examine the fundamental skills needed for introductory to programming courses. Additionally, we focus on some of the most popular microworlds and evaluate them in terms of educational suitability.**

Keywords- microworlds; computer supported education; programming learning; computational thinking

I. INTRODUCTION

In the last decade, introduction to programming has been established as a compulsory course for both secondary education and first year Computer Science students. However, there are many challenges that have not been addressed so far. For instance, the high rate of failure in the introductory programming courses has been a topic deeply investigated [1]. It may be the case that the traditional learning methods, which include programming from an early age, may not be suitable for novice learners who have not yet developed problem-solving and computational thinking skills.

A traditional programming class usually encourages the students to participate in the learning activity by developing a program in a specific language, just after they have been introduced to a respective textbook [3]. Nevertheless, learning programming does not only involve reading educational textbooks and comprehending object-oriented concepts [20]. It also requires that the students possess fundamental computational thinking skills along with some basic mathematic knowledge which are the main requisites for developing programming skills. Furthermore, another issue has derived from the traditional programming courses that include only textual programming. Nowadays students are grown up with computer based graphical environments, thus, text-based programs and command line are neither impressive nor familiar to them [4].

Against the above, we argue that, although the above methods can be useful in specific occasions, there are more effective ways to start learning programming. In fact, considering all these concerns, many researchers in the field of computer science education turned their attention to the utilization of new educational technologies such as Interactive Learning Environments (ILEs).

In this paper, we present the role of interactive learning environments in distributed educational settings. In particular, we focus on microworlds which are a subgroup of interactive learning environments and highlight their educational value compared to the traditional teaching approach of programming. Finally, we examine the theoretical principles behind these tools and qualitatively evaluate them in order to explore their educational suitability in the context of second and higher education.

II. BACKGROUND

A. Introduction to Interactive Learning Environments

An Interactive learning environment (ILE) is an educational tool which aim is to facilitate teaching and support learning programming by leveraging the use of more advanced computer interfaces that include interactive simulations, dynamic visualization and adaptive support [5].

Typically, ILEs support distributed learning because learners can use them from home and interact through emails, chat, forums, etc. This social interaction between students is a very important factor for learning. In fact, the learning communities have highlighted that learning outcomes can be enhanced by engaging students in "multiple forms of interaction: learner-learner, learner-group, learner-content, and learner-instructor" [15]. From that perspective, students can bring different perspectives to a problem-solving situation by using an educational tool to participate in activities where deliverables can be uploaded, reviewed, discussed and altered by others.

B. The birth of microworlds

Classic constructivism is a learning theory that was introduced by Piaget. It disintegrates learning into four stages according to the age of each learner [26]. In particular, Piaget's approach argues that "learning by doing" can be an effective way for learning and focuses on the interaction between the learner and his/her environment [27]. This approach is claimed to be suited to today's knowledge society because knowledge is more distributed and easier accessible than some years ago [27]. Another scope of constructivism is constructionism, which is based on the work of Harel and Papert which emphasizes on situated learning and "knowledge in action" [27]. Papert associated ICT with constructionism with the scope of delivering better education [30]. This association gave birth on Turtle Graphics, which was the first microworld that

978-0-7695-4825-8/12 $26.00 © 2012 IEEE
DOI 10.1109/PCi.2012.18

was developed, based on an educational programing language named LOGO.

Since the 70s, when Turtle graphics was developed, a wave of microworlds has been emerged. These educational tools focus on interactivity as a key point for understanding abstract programming concepts in a more efficient way. Additionally, they promote a new learning methodology of object oriented programming by creating classed, objects and functions through a graphical interface.

Also, in the presence of an educational tool such as microworlds, the traditional workload of the teacher (e.g. preparation of handwritten exercises, presentation of theoretical concepts) is dramatically decreased. Furthermore, these systems can assist teacher to be innovative by engaging students in creative interactive activities [11].

C. Computational thinking

Computational Thinking (CT) is a fundamental mental tool that refines our ability to process information. In her work, Wing [6] stated that computer science is not pure programming and routine thinking but additionally requires fundamental skills.

Nowadays, there is a constant effort to make known to public and particularly to prospective computer scientists that computer science cannot be 'translated' to programming. On the other hand, completely separating CT from programming involves the danger that the students will perceive programming as a low-level dull task rather than as a creative process. Many researchers (e.g. [21]) consider that Computational Thinking can be defined as:

- A way of finding solutions to computer-related problems
- Thinking abstractly to deal with problems and find solutions
- Thinking algorithmically to create a sound solution to a problem
- Is a mental tool to solve complex human problems

James J. Lu et al. [22] underlined that "Programming into Computer Science is what proof construction is to mathematics and what literary analysis is to English." It was also stated that, "Substantial preparation in computational thinking is required before students enroll in programming courses." [22]. In addition to these, many professors argue that CT should be integrated to the curriculum of all levels of education before college in order to facilitate students to unfold "abstract thinking skills" [7].

D. The challenge of teaching programming

Over the past several years, researchers [9] have shown that when the students deal with object-oriented programming there are many things that they should focus on, such as the code, the model, the objects, the files and the display.

However, there are many efficient and innovative ways of teaching the fundamental skills for programming that do not only contribute to a state of the art teaching methodology but also attempt to motivate students by making learning a fun process. There are many graphical programming tools have a high educational value. These tools are not only more attractive compared to textual programming, but they also encourage students to concentrate in more important and abstract concepts. Some of these tools are proposed as a part of a learning methodology in the primary school where the students are in the appropriate age of starting thinking computationally and developing problem solving skills [8].

Furthermore, programming can be taught in a course either by learning a pseudo language or a "real" programming language (Java for example). The Table I bellow presents the basic programming skills which are required in order to use a programming language in the context of problem solving activities [19]:

TABLE I. PROGRAMMING BASIC SKILLS

Types of variables	Control flow structures	Object orientation
Characters	Selection	Object
Numbers	Repetition	Inheritance

If we were to define programming in steps the first one would be to develop the program logic to solve a specific problem. The second step would be to write the program in a specific language, the third one to compile the program in order to transform it into a machine language and the finally to test it and debug it.

This whole procedure may seem advanced for a novice learner to assimilate. Generally, logic is the most fundamental part of the process. Additionally it is the first step to develop a complete program. Although, many high level languages such as C, C++ and Java, have syntax and semantic rules that pretty much look alike, writing the statements in a specific programming language requires the knowledge of the structure of the language and sometimes some memorization. Hence, regarding the complex syntax of programming languages which may be useful for professionals but have no pedagogical value, a lot of learners have difficulties in understanding fundamental concepts such as how to manage a data structure or to produce an algorithm that resolves a problem.

Additionally, the traditional learning methodology is sometimes ineffective due to the inability to adapt to different learning styles. Also, another problem arises from the lack of engagement of the traditional, teacher-centered learning methodology. On the contrary, based on constructivism, the activities of students and the social interactions between peers have the same impact on learning such as knowledge itself [16].

In our study, we argue that many of the above issues can be addressed by the utilization of microworlds which can be beneficial for introducing novice learners to programming. The incorporation of these educational tools in introductory programming courses can facilitate learners to develop problem-solving skills and implement their solutions in a specific programming language [1]. We consider the focus of our work to be twofold: a) to review some of the most popular microworlds and b) evaluate them in respect to their educational value. Our evaluation is based on the following axes: flexibility in the problem-solving domain, textual programming capabilities, syntactical errors inclusion, educational suitability (as regards to teaching the fundamentals of programming) and the support of official student communities.

III. MICROWORLD TECHNOLOGIES

A. Popular microworlds

During our research on microworlds we observed that some tools support textual programming while others do not. Particularly, some tools are designed to support programming by dragging and dropping command blocks, with the view of providing a simple interface for novice learners and eliminating the complexity of syntactical errors. However, such tools can be perceived as disconnected from "real" programming languages [25].

In this section, we present some of the most popular microworlds that are freely available. Additionally, we evaluate them in terms of qualitative criteria in order to define their educational suitability and the possibility of integrating them as knowledge assisting tools in introductory to programming courses.

1) Karel the Robot. This framework supports the programming of the behavior of a robot which acts in a two-dimensional virtual environment. The learner can manipulate the microworld by adding walls, other robots or beepers in it. An extention to Karel is Jeroo, which integrates a text editor which supports code and errors highlighting and suggests the user to alter the code and recompile it [23]. Both of these microworlds are based on the general idea that students "wire in" code in Java, compile it and observe the behavior of the robot(s) on the screen. The main advantage of Karel is the visual feedback that provides which allows the student to test whether the program executes in a semantically correct manner.

2) Turtle Graphics. Turtle Graphics is a LOGO-based project, which was initially developed in the 70's with the scope of introducing young novice learners to fundamental programming concepts. The Turtle Graphics microworld derived from an MIT project of a turtle-shappedrobot that was programmable in Logo. Since then, Turtle Graphics has "migrated" in the computer screen as a virtual environment of two dimensions. In Turtle Graphics, the actor is a virtual turtle, which can be programmed to move and/or leave "traces" that produce different drawings. Programming with Turtle Graphics can introduce basic computational concepts such as iteration or recursion [10].

3) Marine Biology Case Study. The Marine Biology Case Study is built on Java. This framework provides a two-dimensional world, structured as a grid of positions. The actors that populate this world are fish and their behavior is defined by programming them in Java [24]. Similarly to the previous example, the marine biology framework provides immediate visual feedback, though it does not support direct object interaction. This tool has been integrated in the curriculum of secondary education in the USA (Advanced Placement Computer Science) [11] not only for educational purposes but also with the scope of testing students' knowledge in Java.

4) *Alice.* Recent research [12] has shown that although there are many candidate learners of computer science, most of them are men. This derives from the fact most women "have fewer opportunities to interact with computers" and "lack of encouragement from peers parents, and educators" [13]. Moreover, research claims that the decisions on whether to get involved or not with computer science are taken in their middle school years or even earlier. Based on those facts, developers from Carnegie Mellon University decided to build a programming tool that would encourage young girls that are "old enough to handle the complexity of computer programming but are less likely to have already decided against pursuing computer science" [13].

Alice is a framework, which provides a 3D programming environment suitable for children. It supports the development of animated scenarios and actors which can interact with each other.

Although most of the basic programming concepts are provided, Alice does not support textual programming. However, the absence of coding implies the absence of syntactic errors, thus making the tool more appropriate and appealing to children or novice learners who are interested in programming. Programming in Alice is facilitated by dragging and dropping commands in the editor and the outcome is an animation that is displayed on the GUI. Furthermore, instructional support is provided by the interactive tutorial of Alice.

In regard to teaching considerations, Alice is an educational programming tool that it is based on storytelling. Storytelling is something interesting to children and comes more natural than pure code, as it does not include difficult programming concepts, thus programming in such an environment is facilitated. The output of programming in Alice, is an animated scenario, which can be easily understood by children. Despite the advantage of storytelling development, a main drawback is the fact that decreases the flexibility in the problem-solving domain.

The figure below (Fig. 1) illustrates the different parts of Alice layout. Part A contains the tree structure domain of the objects. Part B consists of the microworld where we can observe from various angles the visual feedback that is provided when we compile a program. Part C is the editor of new events, part D is the commands' editor and part E constitutes the details window.

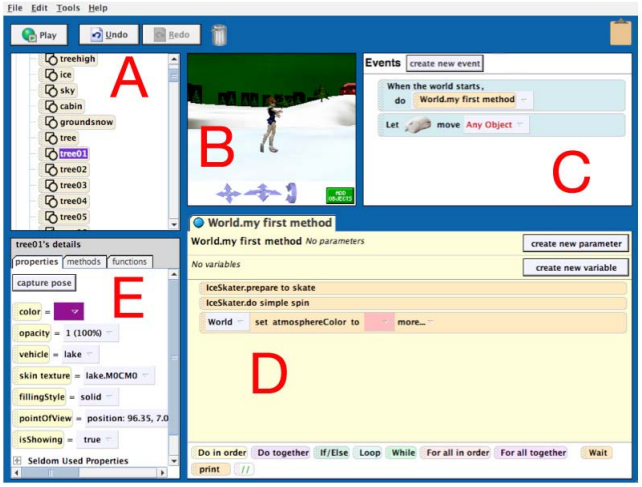

Figure 1. Alice GUI.

5) Scratch. Researchers from the MIT University cooperated with Lego and they managed to develop Lego Mindstorms. Additionally, they wanted the development in Scratch to have the same feeling for the youths as that of constructing with Lego bricks. In Scratch instead of bricks we use command blocks with the view of developing media-rich animations. A main advantage of Scratch is the lack of syntax errors. "Scratch blocks are shaped to fit together only in ways that make syntactic sense" [18].

Because of its wide acceptance, Scratch is now translated in more than 40 languages. Except from schools, Scratch has been integrated in the curriculum of some of the greatest academic institutions such as Harvard and Berkeley, as an antechamber for teaching programming. However, if we were to develop an introductory to programming course, an issue that naturally arises is what tool will succeed Scratch.

Regarding the future roadmaps of Scratch, developers intent to make it more "tinkerable", "meaningful" and social but at the same time to keep it simple to handle [18].

As shown in the figure below (Fig.2), Scratch's graphical user interface comprises of three panels. The first (A) includes all the available commands that the user can perform. These commands are separated in eight different categories based on their nature (e.g. variables, control flow structures, motion, etc.). The second part of the layout (B) constitutes the scripts panel where command blocks are combined. The last panel (C) contains the actual microworld which provides the visual feedback of the actors (or the sprits in Scratch terminology).

.

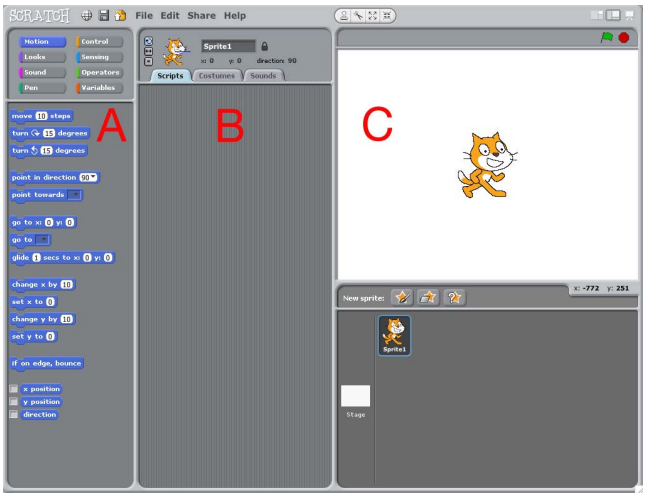

Figure 2. Scratch GUI.

6) *Greenfoot.* The platform of Greenfoot is a two dimensional environment, in which interactive projects can be developed. The Greenfoot GUI allows users to alter the behavior and the graphical representation of the objects by editing the java code from the text editor or by scripting new functions. Functions in Greenfoot can be triggered by various events (e.g. by clicking on the actor or the appropriate button). Additionally, Greenfoot supports the delivery of immediate feedback.

The main goal of Greenfoot is: "Teaching object-orientation at the school level" though the tool is also integrated to the CS1 level in many courses of academic institutions. According to the developers, the general objectives of Greenfoot fully align with Kolb's learning circle [17], which is the official learning framework in the UK [14]. The general goals of the tool are presented below [11]:

- "Experimentation and visual feedback": Important concepts can be visualized by the feedback the environment provides thus facilitating learners of no prior knowledge in programming.
- "Flexible scenarios": The system provides some basic, complete scenarios that are integrated. Furthermore one can create his/her scenario for teaching purposes or even fun.
- "Clean illustration of object-oriented concepts": Through visualization, the learner can fully comprehend the fundamental concepts of object-oriented programming such as classes, objects and inheritance.

The Greenfoot GUI (Fig. 3), mainly consists of the microworld (A). The second part (B), provides a tree view of the all the current classes, which can be edited by clicking on the respective tabs. Additionally, all the classes are produced by the two supper-classes of the system, GreenfootWorld and GreenfootObject, which cannot be modified. Greenfoot also includes an integrated debugger and a Java compiler. Part C consists of the buttons that control the animation (act, run, speed) and the button (Compile all) that compiles the program.

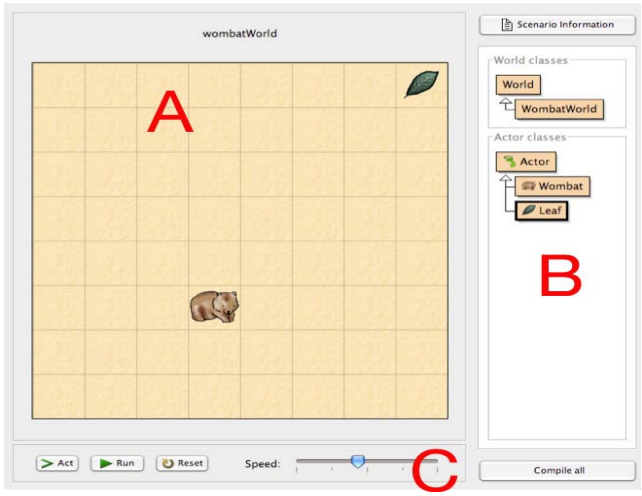

Figure 3. Greenfoot GUI

B. *Educational suitability*

In the previous section, we have presented of the most popular microworlds taking into account their implementation in real classroom settings (Table II). Considering this, we argue that the integration of such interactive tools in programming courses for novice learners can be very beneficial.

In our research we observed that some of these tools such as the "Turtle Graphics" and "The Marine Biology Case Study" can facilitate educators in introducing the fundamental concepts of programming to learners. However, although their

ability to instantly provide visual can assist learners to semantically verify their actions, a major drawback of these tools is the lack of interaction between objects (see Table II).

The microworld Alice, which does not provide object oriented development possibilities, is based on storytelling scenarios. Hence, Alice does not seem to provide the flexibility required in the domain of problem-solving. Another issue, which arises from the survey in [29], reveals that a small part of the sample (5%) experienced difficulties in perceiving the "depth of the objects" in the 3D environment of Alice.

In the context of programming learning, Al-Bow and Debra [28] revealed that novice learners often overestimate their programming abilities. Additionally, observations from [29] indicate that learners with no prior programming experience have difficulties on typing commands.

Considering the above studies ([28], [29]), we argue that Scratch is suited as an educational tool on a very introductory level, as it does not include textual programming and object orientation concepts, which are advanced for novice learners. Furthermore, its highly graphical interface is considered to be flexible since it allows the interaction between objects as well as an immediate visual feedback mechanism. Moreover, a survey from [31], indicates that the 80% of the sample appreciate the simplicity of Scratch in terms of developing a project. Additionally, recent research findings [25], [32] revealed that not only Scratch outperformed LOGO-like environments in the context of teaching variables and conditionals but also had a positive influence on 76% of the users. To conclude, another asset of Scratch is the existence of an active community, where all scratch users can upload their projects, vote on other projects, download or edit them.

Karel the Robot and Greenfoot, which are both developed in Java, provide all the characteristics of the programming language including syntactical errors. Considering this fact, many argue that these tools may not be well suited for novice programming learners because of their advanced syntax complexity. However, such microworlds can assist teachers to introduce their students to abstract programming concepts such as object orientation, inheritance, etc. In particular, a post-test that was performed in a CS1 level course [32], indicated that students who used Greenfoot achieved a better comprehension on the concepts of class, object and method. In addition, another research [28] underlined that Greenfoot efficiently supported students to comprehend classes and objects, if-else statements and loops. Thus, we argue that Greenfoot can be a valuable tool for facilitating the transmission from a non-textual microworld (e.g. Scratch) to a "real" programming language.

As regards to the teacher use, Greenfoot provides a flexible and highly graphical framework in which a wide variety of scenarios can be implemented and shared to the users' community. Hence, Greenfoot enables the teacher to organize interesting activities which can differ from the typical or dull programming tasks.

Lastly, it should be noted that although all educational microworlds have been emerged within the theoretical framework of constructivism, Greenfoot is also supported by the theoretical framework of Kolb's circle [17].

IV. Conclusion

Regarding the general objectives of this report, we identified the fundamental skills needed for programming and argued that microworlds can be helpful educational tools that facilitate and support the learning process of programming. These interactive learning tools facilitate students in learning abstract programming concepts through working examples. Afterwards, students can gradually move to traditional programming languages by recalling and using the basic programming concepts that they were taught.

We suggest the use of Scratch as an introductory tool to programming for novices. In addition, we consider that Greenfoot can be efficiently used to scaffold learners' transition from microworlds to "real" programming languages.

Microworlds can account for the need of today's knowledge society for distributed learning. More importantly, these tools are appealing to all ages and make learning a process of "serious fun". We converge to the view that the utilization of microworlds can enrich introduction to programming courses by proliferating engagement to the students. In conclusion, microworlds can reduce the traditional teacher's burden of providing educational material as they integrate a range of scenarios in which many problems can be expressed.

References

[1] A. Gomes and A. J. Mendes, "An environment to improve programming education," Proc. of the International Conference on Computer Systems and Technologies, 2007.

[2] V. Allan, V. Barr, D. Brylow and S. Hambrusch, "Computational thinking in high school courses", Proc. of ACM SIGCSE, 2010.

[3] J. Cuny, "Transforming High School Computing: A Compelling Need, A National Effort". [Online]. Available: http://www.worldcat.org/isbn/0071362681

[4] M. Guzdial and J. Robertson, "Too much programing to soon? ", in Communications of the ACM, vol. 53, no 3, pp. 10-11, 2010.

[5] T. Crews, G. Biswas, S. Goldman and J. Bransford, "Anchored interactive learning environments", in International Journal of AI in Education, vol. 8, pp. 142-178, 1997.

[6] J. M. Wing, "Computational thinking and thinking about computing", in Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 366, no. 1881, pp. 3717-3725, 2008.

[7] P. B. Henderson, T. J. Cortina, O. Hazzan and J. M. Wing, "Computational Thinking", in ACM SIGCSE Bulletin, vol. 39, no. 1, pp. 195-196, 2007.

[8] P. Curzon "Serious Fun in Computer Science", Proc. of the 12th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '07), 2007.

[9] O. Berge, R. E. Borge, A. Fjuk, J. Kaasboll and T. Samuelsen, "Learning Object Oriented Programming", in Norsk Informatikkonferanse NIK'2003, Tapir Akademisk Forlag, pp. 37-47, 2003.

[10] M. E. Caspersen and H. B. Christensen, "Here, There and Everywhere – On the Recurring Use of Turtle Graphics in CS1", Proc. of the Fourth Australasian Conference on Computing Education, pp. 34-40, 2000.

[11] P. Henriksen and M. Kolling, "Greenfoot: Combining Object Visualisation with Interaction", in SIGCSE Bulletin, vol. 39, no. 4, pp. 204-223, 2007.

[12] J. Vegso, "Drop in CS Bachelor's Degree Production", in Compu ting Research News, vol. 18, no. 2, 2006.

[13] C. Kelleher, R. Pausch and S. Kiesler, "Storytelling Alice Motivates Middle School Girls to Learn Computer Programming", Proc. ACM CHI, 2007, pp. 1455-1464.

[14] D. A. Kolb, "Experiential Learning. Experiences as the source of Learning and Development", Englewood Cliffs, NJ: Prentice-Hall, 1984.

[15] Y. Dan and Y. Juan, "The Analysis of Education Information Systems Based on the Logical Overview", Proc. of the first international workshop on Education Technology and Computer Science, vol.3, 2009, pp. 34-37.

[16] N. Dabbagh, "Pedagogical models for E-Learning: A theory-based design framework", in International Journal of Technology in Teaching and Learning, vol. 1, no. 1, pp. 25-44, 2005.

[17] The UK Professional Standards Framework for teaching and supporting learning in higher education. [Online]. Available: http://www.heacademy.ac.uk/assets/documents/rewardandrecog/ProfessionalStandardsFramework.pdf

[18] M. Resnick, J. Maloney and A. M. Hernández, N. Rusk and E. Eastmond "Scratch: Programming for All", in Communications of the ACM, vol. 52, no. 11, pp.60-67 , 2009.

[19] S. Kepser, "A Simple Proof for the Turing-Completenessof XSLT and XQuery", in Extreme Markup Languages, 2004.

[20] S. DeReamer, "Teaching Computer Science: A Neumont Philosophy", in ACM SIGSOFT Software Engineering Notes, vol. 35, no. 1, pp. 31-34, 2010.

[21] J. M. Wing, "Computational Thinking", in Communications of the ACM, vol. 49, no. 3, pp. 33-35, 2006.

[22] J. J. Lu and G. H. L. Fletcher, "Thinking About Computational Thinking", Proc. of the 40th ACM technical symposium on Computer science education., 2009, pp. 260-264.

[23] R. Borge and A. Fjuk "Using Karel J collaboratively to facilitate object-oriented learning", Proc. of the IEEE International Conference on Advanced Learning Technologies (ICALT'04), 2004, pp. 580- 584.

[24] Marine Biology Case Study. [Online]. Available: http://www.collegeboard.com/

[25] C. M. Lewis, "How Programming Environment Shapes Perception, Learning and Goals: Logo vs. Scratch", Proc. SIGCSE'10, ACM Press, WI, USA, 2010.

[26] W. Huitt and J. Hummel, "Piaget's theory of cognitive development", Educational Psychology Interactive, Valdosta, GA: Valdosta State University, 2003.

[27] B. Gros, "Knowledge Construction and Technology", in Journal of Educational Multimedia and Hypermedia, vol. 11, no. 4, pp. 323-343, 2002.

[28] M. Al-Bow and D. Austin, "Using Greenfoot and Games to Teach Rising 9th and 10th Grade Novice Programmers", Proc. of the 2008 ACM SIGGRAPH symposium on Video games, 2008, pp. 55-59.

[29] M. Conway, S. Audia and T. Burnette, "Alice: Lessons Learned from Building a 3D System For Novices", Proc. of the SIGCHI conference on Human factors in computing systems, 2000, pp. 486-493.

[30] I. Harel and S. Papert, "Constructionism", in Constructionism, Norwood, NJ: Ablex Publishing, 1991.

[31] G. Fesakis and K. Serafeim, "Influence of the familiarization with "scratch" on future teachers' opinions and attitudes about programming" and ICT in education", Proc. of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education (ITiCSE '09), 2009, pp. 258-262.

[32] S. Montero, P. Díaz, D. Díez and I. Aedo, "Dual Instructional Support Materials for introductory object-oriented programming: classes vs. objects", Proc. of the 2010 IEEE Education Engineering conference, 2010, pp. 1929-1934.

TABLE II. MICROWORLDS EVALUATION

	Karel the robot	**Turtle graphics**	**The marine biology case study**	**Alice**	**Greenfoot**	**Scratch**
Syntactical errors	Yes	Yes	Yes	No	Yes	No
Textual programming	Yes	Yes	Yes	No	Yes	No
Graphical representation	2D	2D	2D	3D	2D	2D
Target group	Higher Education	Primary and Secondary education	Secondary education	Secondary education	Higher Education	Primary education
Flexibility	Problem solving domain oriented, interaction between objects	Problem solving domain oriented, lack of interaction between objects	Problem solving domain oriented, lack of interaction between objects	Storytelling oriented, Interaction between objects	Problem solving domain oriented, Interaction between objects	Problem solving domain oriented, Interaction between objects
Educational suitability	Types of variables, Control flow structures, Object orientation concepts, Methods, Arrays	Types of variables, Control flow structures	Types of variables, Control flow, structures	Types of variables, Control flow, structures	Types of variables, Control flow structures, Object orientation concepts, Methods, Arrays	Types of variables, Control flow structures
Students community	No	No	No	Yes (http://www.alice.org)	Yes (http://www.greenfoot.org)	Yes (http://scratch.mit.edu)