Available online at www.sciencedirect.com

ScienceDirect

Procedia - Social and Behavioral Sciences 00 (2012) 000–000

Procedia Social and Behavioral Sciences

www.elsevier.com/locate/procedia

WCES 2012

Diverse categories of programming learning activities could be performed within Scratch

Maria Kordaki [a] *

[a]Ass. Professor, Dept of Cultural technology and Communications, University of the Aegean, University Hill, Mytilene,81100, Greece

Abstract

This study presents a set of categories of learning activities which could be performed by the students using the tools of the well-known educational software Scratch (www.scratch.mit.edu) that is dedicated for the learning of programming by novices. Specifically, eleven categories of learning activities that could be performed within Scratch were formed, namely: (a) Free creative activities, (b) Solving a specific problem, © Multiple solution tasks, (d) Experimentation within working Scratch projects, (e) Modification of working Scratch projects, (f) Working on a complete Scratch output and a correct but incomplete part of its code, (g) Working on a complete Scratch output and a mixed form of its code, (h) Working with a complete Scratch output and an incorrect part of its code, (i) Working with the complete code of a Scratch program and predicting its output, (j) Black-box activities, and (k) Collaborative learning activities. Computing teachers can use these categories of activities in their attempts to design appropriate every day classroom settings for the learning of programming by novices within Scratch.

*Keywords:*Programming, activities, Scratch, novices;

1. Introduction

The teaching of programming is one of the main challenges in Computer Science, despite it being not only an essential topic proposed for a K-12 curriculum, and a fundamental subject in studying Computing at Tertiary level but also a ‘mental tool’ of general interest (Satratzemi, Dagdilelis & Evaggelidis, 2002) where problem-solving skills can be encouraged in learners. In truth, programming is more of a mental skill than a body of knowledge (Hadjerrouit, 2008). However, students seemed to encounter serious difficulties during the whole programming process (Soloway and Spohrer, 1989; Winslow, 1996; Robbins, Rountree and Rountree, 2003; Hadjerrouit, 2008; Eckerdal, 2009). Specifically, as programming problems come from a wide range of problem domains, many students may lack the necessary domain background to face the problems at hand (Winslow, 1996). Students also face serious difficulties in understanding the algorithmic logic used in programming problem solving (Soloway & Spohrer, 1989). Most importantly, novices are rarely aware of the problems that can be solved by a computer and the benefits to be had from using programming. Furthermore, students often begin programming with a “big handicap”: they do not know how a computer works (Eckerdal, 2009). In addition, some students know the

* Corresponding Author name Maria Kordaki. Tel.: +030-2610-993-102
E-mail address: m.kordaki@aegean.gr

syntactical rules of specific statements and constructs of a specific programming language but do not know how to combine these features into valid programs (Soloway and Spohrer, 1989; Winslow, 1996; Eckerdal, 2009). Novices have also difficulties to identify primitive programming constructs such as conditionals, recursions and loops as well as entities such as variables, counters and conditions - in their initial solution; nor can they express them with computer-based structures (Eckerdal, 2009). In fact, there is a big gap between the informal solutions they give and the more formal computer-oriented solution that is required (Winslow, 1996). Students also have semantic and syntactic misconceptions of almost every basic structure and concept of a programming language (Britos, Rey, Rodríguez, & García-Martínez, 2008). There are also important difficulties in understanding the way input and output commands function (Soloway and Spohrer, 1989). Finally, students encounter serious difficulties to comprehend and use the online help provided by the environment (Allwood, 1986). Thus, there has been strong interest in the adoption of effective methods to improve the ability of students to solve problems requiring programming solutions (Robbins, Rountree and Rountree, 2003; Lahtinen, Ala-Mutka, & Järvinen, 2005). However, despite the fact that programming in schools is currently supported by professional integrated software environments; those programs solely feature capabilities and aids for experienced users (Freund, & Roberts, 1996).

Aiming to enhance students' learning about programming, the well known educational software Scratch (www.scratch.mit.edu) offers both a new programming language and a suitable environment for beginners. Scratch is based on the same concepts as other typical programming languages. However, Scratch does not support the learning of programming through writing long programs in text format using complex structures and following rigid syntactical rules which are mainly familiar to expert programmers (Ford, 2008). On the contrary, the Scratch environment motivates beginners to be involved in programming by exploiting their interest in games and play. Games are viewed as being the most ancient and time-honored vehicle for education (Crawford, 1982) and are also among the most enjoyable and motivating activities for the young (Prensky, 2001). Scratch is also easily accessible to everyone, as it is free of charge and is supported by an international community of users. Scratch also offers a context where social and constructivist programming learning settings can be supported. However, even if some progress has been made in solving some learning problems in programming using constructivist learning theories and specific educational software, the problems and difficulties associated with the learning of introductory programming by novices remain to be researched (Hadjerrouit, 2008). In fact, few educators systematically apply constructivism to the learning of computer programming (Berglund, Daniels & Pears, 2006).

To this end, an attempt has been made to formulate categories of essential types of learning activities which could be performed within Scratch taking into account social and constructivist learning approaches (Vygotsky, 1978; Jonassen, 1999). In fact, eleven categories of learning activities that could be performed within Scratch were formed. Computing teachers can use these categories of activities in their attempts to design appropriate every day classroom settings for the learning of programming by novices within Scratch. This is the contribution of this paper.

In the next section of this paper, a brief description of essential features provided by the Scratch environment is given, followed by a presentation of the aforementioned categories of learning activities which could be performed within the program. Finally, the paper ends with a summary and future research plans.

2. Few words about Scratch

Scratch is a "media-rich programming environment" (Maloney, Burd, Kafai, Rusk, Silverman, and Resnick, 2004) that empowers students—from day one—to implement animations, games, and interactive art. Although Scratch was designed "to enhance the development of technological fluency [among youths] at after-school centers in economically disadvantaged communities" (Maloney, Burd, Kafai, Rusk, Silverman, and Resnick, 2004) it is proposed as a gateway to languages like Java for students in higher education as well (Malan, and Leitner, 2007). It seems that, Scratch lowers the bar to programming, empowering first-time programmers not only to master programmatic constructs before syntax but also to focus on problems of logic before syntax (Malan, and Leitner, 2007). In fact, Scratch enables students to program with a mouse, presenting programmatic constructs as blocks (i.e., puzzle pieces) that only fit together if "syntactically" appropriate. In particular, Scratch's interface is divided into four areas, namely: a blocks palette, a scripts area, a selection area, and a stage. The blocks palette offers students eight categories of color-coded building blocks, puzzle pieces of sorts that collectively govern the sprites' behavior on the stage. Programming a sprite is as simple as selecting it in the selection area (after creating it with a click of a button) and dragging two or more blocks to the scripts area, where they will fit together if "syntactically" appropriate. Among these blocks are programming constructs such as statements, Boolean expressions, conditions, loops, and variables as well as support for multiple threads and events. Using these blocks students can program one or more "sprites" (i.e., characters) on a "stage," and the final result is some animation, game, or interactive art.

Thus, Scratch is highly capable of facilitating the design of learning activities that encourage learners to take a creative, investigative and exploratory perspective, to express their inter-individual learning variety, to make self-corrections, as well as to formulate and verify conjectures. Most importantly, authentic, meaningful and playful learning activities can be integrated within the context of this software. In particular, Scratch provides students with various opportunities for the learning of programming, such as: (a) interactivity and encouragement of multi-sensory, active, experimental learning, (b) emphasizing the design of the algorithmic solution rather than memorizing the syntactic rules of a specific programming language, (c) provision of a rich set of usable coding tools such as programming patterns and models for selecting and combining language structures, and also implementing diverse programming primitive constructs, such as conditionals, iterations, variables, etc, (d) provision of tools to construct a variety of linked, dynamic representations, numerical, textual, and audio-visual, (e) activation and use of prior-knowledge in order to advance, (f) avoidance of the cognitive load coming from having to acquire the necessary knowledge domain to face the school book-like typical programming problems, (g) problem solving support, (h) visualization of the program and its output in a real time execution, (i) encouragement of the solution of various appropriate tasks, (j) providing immediate feedback on student programming actions for self-correction, (k) learning in different ways from those often in use, in school settings, and (l) motivation. The aforementioned opportunities are acknowledged by many researchers as appropriate to be integrated into successful computer environments for the learning of any subject, as well as for the learning of programming (Freund, & Roberts, 1996; Soloway & Spohrer, 1989; Winslow, 1996; Garner, 2007).

3. Categories of essential learning activities which could be performed within Scratch

A diversity of learning settings could be formed within Scratch: First of all, students should be provided with opportunities to freely experiment with the tools of Scratch to create their own games, animations and interactive arts. Students should also be asked to solve a specific problem within Scratch. However, students should also be provided with the chance to perform various learning activities while working with both; complete working projects and incomplete projects within Scratch.. Thus, working within the aforementioned different learning settings, students could perform the following type of programming learning activities within Scratch:

(a) *Free creative activities*: Students should be asked to experiment with the tools of Scratch to formulate programs that produce an output (eg.a game, an interactive art, etc) of their preference. This kind of activity can stimulate students' motivation and creative thought that is not so common in real school practices. However, students have to be familiarized with the tools of Scratch and its possibilities by themselves. To this end, they can use trial and error techniques, perform various explorations on working examples presented in various websites as well as collaborate with other students around the world.

(b) *Solving a specific problem*: Here, students should be asked to provide a solution to a given problem within Scratch. Students have the opportunity to immediately see the results on their programming attempts on the 'stage' and make appropriate corrections.

© *Multiple solution tasks*: Here as well, students should be asked to formulate programming solutions to a given task 'in as many ways as possible'. This kind of tasks can challenge the formation of multiple perspectives regarding with a program design by the students (Kordaki, 2009). Promoting the development of multiple perspectives during learning settings is a learning approach that provides opportunities for the construction of flexible and meaningful knowledge structures at the same time encouraging the expression of learners' inter- and intra-individual differences (Kordaki, 2006). Multiple solution-based learning activities are also acknowledged as essential for the development of problem solving abilities, creativity and advanced mathematical thinking (Leikin, Levav-Waynberg, Gurevich, &

Mednikov, 2006). With the promotion of multiple perspectives, learners also have the chance to become aware that there are multiple approaches to an issue, which is the case in real-life situations. Learners have also the chance to explore each perspective to seek a meaningful resolution to the issue at hand and construct new meanings within the context of their own experiences (Dabbagh, 2005).

(d) *Experimentation within working Scratch projects*: Here, students should be asked to perform various experimentations with the specific programming constructs used in the formation of the code of a project (eg. omit specific blocks from the code of the program or change their position, change the value of some variables, etc) in order to provide explanations and justifications about the role of these constructs so that be able to understand them. This kind of activities could be used as a scaffolding element for students' familiarization with programming constructs and the Scratch tools so as to avoid some possible frustration from the cognitive load emerging from dealing with problems requiring the use of a 'new' methodology such as algorithmic logic.

(e) *Modification of working Scratch projects*: Here as well, students should be asked to modify the code of a working project in order to produce a slightly different output on the stage. This is a further step where students are called to actively construct a part of the solution of a problem. By being involved in this kind of activity, students can use the knowledge they acquired during other previous programming activities in order to advance in the modification of the program at hand. At the same time students can feel as sheltered by the context of the already working project in order to appropriately face the challenges of its modification.

(f) *Working on a complete Scratch output and a correct but incomplete part of its code*: By being involved in this kind of activity students can reflect on a working interactive art/game on the 'stage' as a guide in combination with the incomplete code provided and use their previous knowledge in order to provide the complete code. In order to succeed, Scratch provides students with the chance to correct their programming attempts through trial and error.

(g) *Working on a complete Scratch output and a mixed form of its code*: Here, students should be provided with a mixed mode of the code of a specific Scratch project and should be asked to make the correct adjustments so that this code produces the given output. This kind of activity can act as a scaffolding element on students' attempts to formulate the correct code as they do not have to invent it from the scratch but have all its parts in their disposal and they have to think and experiment till they find the appropriate sequence of commands.

(h) *Working on a complete Scratch output and an incorrect part of its code*: In fact, the code provided in this type of activity should contain usual errors that students make during programming. Then, students should be asked to find the code mistakes and also make appropriate corrections so that this code produces the expected output given. This kind of activity is a little bit hard for the students to grasp, as it implies that they have to find the specific mistakes included in the given code and also to make corrections so that it produces the given output.

(i) *Working with the complete code of a Scratch program and predicting its output*: Making predictions is a high level cognitive activity (Marzano, Brandt, Hughes, Jones, Presseisen, Rankin and Suhor, 1988) that means that students are able to use their knowledge in an operative level. This kind of activity implies that students not only know the role of each specific programming construct used in the given code but that they understand what the synthesis of all specific programming constructs in the code really means.

(j) *Black-box activities*: Here, students should be provided with the output of a Scratch program (as it runs on the stage area) and should be asked to formulate the appropriate code producing it. Students can experiment with the output and reflect on their previous knowledge to construct the code. Indeed, it is a demanding kind of activity that calls students to use various thinking skills such as reversible thinking, analytical and synthetic thinking, as well as reflection, prediction, hypothesis generation and exploration. The development of this kind of thinking skills go hand by hand with the acquisition of problem-solving and decision-making skills that are potentially essential in both the personal and professional life of a learner (Dabbagh, 2005).

(k) *Collaborative learning activities*: Various collaborative patterns (eg. The Jigsaw collaborative pattern) should be used for students' collaboration in order to perform appropriate programming activities within Scratch (Theodorou and Kordaki, 2010). Students can collaborate with their classmates as well as with other students through Internet around the world in order to perform such activities.

4. Summary and future research plans

This paper presented a set of eleven essential categories of programming learning activities which could be performed within the well known educational software Scratch that is dedicated for the novices' learning of programming in a playful and creative way. These categories are: (a) Free creative activities, (b) Solving a specific problem, © Multiple solution tasks, (d) Experimentation within working Scratch projects, (e) Modification of working Scratch projects, (f) Working on a complete Scratch output and a correct but incomplete part of its code, (g) Working on a complete Scratch output and a mixed form of its code, (h) Working with a complete Scratch output and an incorrect part of its code, (i) Working with the complete code of a Scratch program and predicting its output, (j) Black-box activities, and (k) Collaborative learning activities. All except categories (a), (b), (d), (e) and (k), describe innovative learning activities within the context of Scratch. The performance of such kind of programming learning activities could encourage students' learning of programming as well as the development of their core thinking skills. The use of this diverse kind of activities in real classrooms to investigate their impact on students' programming thinking is in our future plans.

References

Allwood, C. (1986). Novices on the computer: a review of the literature. *International Journal of Man-Machine Studies*, Vol. 25, 633-658.

Berglund, A., M. Daniels and A. Pears, A. (2006). Qualitative research projects in computing education research: an overview. In Proceedings of the *Eighth Australasian Computing Education Conference* (ACE2006), Hobart, Tasmania, Australia, January 2006.

Britos, P., Rey, E-J., Rodríguez, D. & García-Martínez, R. (2008). Work in Progress – Programming Misunderstandings Discovering Process Based on Intelligent Data Mining Tools. In Proceedings of *38th ASEE/IEEE Frontiers in Education Conference*, October 22 – 25, 2008, Saratoga Springs, NY, F4H1-2.

Crawford, C. (1982). *The Art of Computer Game Design*. Retrieved from: www.vancouver.wsu.edu/ fac/peabody/game-book/ Coverpage.html

Dabbagh, N. (2005). Pedagogical models for E-Learning: A theory-based design framework. *International Journal of Technology in Teaching and Learning*, 1(1), pp. 25-44.

Eckerdal, A. (2009). *Novice Programming Students' Learning of Concepts and Practise*. Dissertation presented at Mathematics and Computer Science, Dept of Information Technology, Upsalla University, Sweeden, http://uu.diva-ortal.org/smash/record.jsf?pid=diva2: 173221

Ford, J.L. (2008). *Scratch Programming for Teens*. Canada: Course Technology PTR.

Freund, S. N. & Roberts, E. S. (1996). THETIS: An ANSI C programming environment designed for introductory use. *ACM, SIGCSE '96 2/96*, Philadelphia, PA USA, 300-304.

Garner, S. (2007). A program design tool to help novices learn programming. In ICT: Providing choices for learners and learning. Proceedings *ascilite* Singapore 2007. Retrieved on 25/07/11 from http://www.ascilite.org.au/singapore07/ pros/garner.pdf.

Jonassen, D. H. (1999). Designing constructivist learning environments. *Instructional design theories and models*, 2, 215-239.

Hadjerrouit, S. (2008). Towards a Blended Learning Model for Teaching and Learning Computer Programming: A Case Study. *Informatics in Education*, 2008, Vol. 7, No. 2, 181–210.

Kordaki, M. (2006). Multiple Representation Systems and Students' Inter-Individual Learning Differences. In P. Kommers & G. Richards (Eds.), Proceedings of *ED-MEDIA 2006* (pp. 2127-2134). Chesapeake, VA: AACE.

Kordaki, M. (2009). Beginners' programming attempts to accomplish a multiple-solution based task within a multiple representational computer environment. In (Eds.), Proceedings of *ED-MEDIA 2009*, June 22-26, 2009, Hawaii, USA. (pp. 3282-3295). Chesapeake, VA: AACE.

Lahtinen, E., Ala-Mutka, K. & Järvinen, H. (2005). A study of the difficulties of novice programmers. *In ITiCSE '05*: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, Caparica, Portugal, ACM Press, 14-18.

Leikin R., Levav-Waynberg A., Gurevich I. & Mednikov L. (2006). Implementation of Multiple Solution Connecting Tasks: Do Students' Attitudes Support Teachers' Reluctance? *Focus on learning problems in mathematics*, 28(1), 1-22.

Malan, D. J. and Leitner, H. H. (2007). Scratch for budding computer scientists. *SIGCSE Bulletin*, 39, 1, pp.223-227.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B. and Resnick, M, (2004). Scratch: A Sneak Preview. In *Second International Conference on Creating, Connecting and Collaborating through Computing*, pp. 104–109, Kyoto, Japan, 2004.

Marzano, J.R., Brandt, S.R., Hughes, C-S, Jones B-F., Presseisen, Z.B., Rankin, C.S. and Suhor, C. (1988). *Dimensions of Thinking: A Framework for Curriculum and Instruction*. VA: Association for Supervision and Curriculum Development.

Prensky, M. (2001). *Digital game-based learning*, Mc Graw-Hill, New York.

Robins, A., Rountree, J. and Rountree, N. (2003). Learning and teaching Programming: A Review and Discussion. *Computer Science Education* 13(2), pp. 137-172.

Satratzemi, M., Dagdilelis, V. & Evaggelidis, G. (2002). An Alternating Approach of Teaching Programming in the Secondary School. In Proceedings of *3rd Panhellenic Conference with International Participation, 'ICT in Education'*, 289-298, Phodes, Greece.

Soloway, E. & Spohrer, J. C. (1989). *Studying the novice programmer*. Hillside, N.J. Erlbaum.

Theodorou, C. and Kordaki, M. (2010). Super Mario: a collaborative game for the learning of variables in programming. *IJAR*, 2(4), pp. 111-118.

Vygotsky, L.S. (1978). Mind in Society. Cambridge, MA: Harvard University Press.

Winslow, L. E. (1996). Programming Pedagogy. *SIGCSE Bulletin*, Vol. 28 (3), 17-22