A DRAWING ENVIRONMENT FOR BEGINNERS' LEARNING OF PROGRAMMING AND C

George Kripotos
Computer Engineer
Dept. of Computer Engineering and Informatics, Patras University, 26500, Rion, Patras, Greece

Maria Kordaki
Adjunct assistant Professor
Department of Computer Engineering and Informatics, Patras University, 26500, Rion Patras, Greece

ABSTRACT

This paper describes the design, features and advantages of a drawing application -the L.E.C.G.O-Paint application- that is part of a constructivist multi-representational computer environment designed for beginners' learning of Programming and C (the LECGO environment; Kordaki, 2007). L.E.C.G.O-Paint supplies tools that provide support for students to express their own intuitive graphical solutions to given problems by using hands-on experience and to receive feedback in the form of the correct interpretation of the said solutions in Multiple Representation Systems (MRS), such as: natural language, imperative, pseudo-code and code in C. Specifically, LECGO-PAINT provides a number of tools for: a) drawing various shapes b) editing shapes, c) using metrics, d) using loops, e) providing feedback in the previously mentioned MRS, f) receiving help and g) performing typical file operations, such as: save, print, open, etc.

KEYWORDS

Programming, C, Secondary Education, Computer Learning.

1. INTRODUCTION

The rapid growth of computing and technology and their impact on modern life have created a pressing need to provide academic coherence in computing and also to improve the level of public understanding of computer science as an academic and professional field. Elementary and secondary schools have a unique opportunity and responsibility to address this need. To this end, a four-level model curriculum of CS for K-12 has been proposed (ACM, 2003). The aim of this curriculum is to better address the need to educate young people in the important subject area of CS and thus better prepare them for effective citizenship in the 21st century. One of the four goals of this curriculum is to introduce the fundamental concepts of CS to all students, beginning at elementary school. In the context of this curriculum, high school graduates will be educated to use computer science skills (especially algorithmic thinking) in their problem-solving activities in other subjects. Thus, the learning of programming is acknowledged as a central element of this curriculum.

However, programming is 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 learners' problem-solving skills can be encouraged. In fact, programming is a complex task including understanding of the task at hand, method finding and coding (Brooks, 1999). According to Winslow (1996), it can be divided into four steps: a) understanding of the problem at hand, b) definition of a solution to that problem, initially in any form, such as text-based or math-based, and in a computer compatible form such as pseudo-code and flow-chart, c) translation of the solution into a selected programming language, and d) testing and debugging of the resulting program. It is worth noting that students encounter serious difficulties in performing all the steps mentioned above (Kurland & Pea, 1989; Lemone & Ching, 1996; Du Boulay, 1986; Soloway and Spohrer, 1989; Winslow, 1996; Robbins, Rountree and Rountree, 2003). Programming problems come from a wide range of problem domains and understanding the problem domain is also critical (Anderson and Soloway, 1985). Many students may lack

the necessary domain background but there are also many other students who can solve a problem by hand but lack the ability to express the solution in computer terms (Winslow, 1996). In addition, novices seem to know the syntax and the semantics of individual statements and constructs but they do not know how to combine these features into valid programs (Soloway and Spohrer, 1989; Winslow, 1996). Most importantly, students are also rarely aware of the problems that can be solved by a computer and the benefits to be had from using programming (Lemone & Ching, 1996). Good performance in programming also implies the ability of learners to use various and new representation systems to express their problem-solving strategies in order to progress smoothly to the formation of the appropriate code (Komis, 2001; Brooks, 1999; Kordaki, 2007). To this end, the role of using pseudo-code before writing code is essential, despite the fact that students prefer to skip the careful thought processes required in program design and use trial and error techniques in their algorithm design (Garner, 2007).

Programming in schools is currently supported by professional integrated software environments. Although they feature capabilities and aids for experienced users, these tools are useless for novices: they do not support algorithmic solutions, provide insufficient feedback and are extremely complicated (Freund, & Roberts, 1996). Thus, a student's frustration with programming often depends less on the target programming language and more on the programming environment in use. To this end, it is worth noting that the role of designing appropriate feedback is crucial for student learning in the context of computer learning environments (Hummel, 2006) and especially those dedicated for the learning of such a complex task as programming, indeed in any educational setting (Cohen, 1985).

Various studies indicate that there is a need for a novice-oriented programming environment. It is suggested that the success of such an environment is mainly dependent on its ability to: a) encourage active learning, b) emphasize design of the algorithmic solution - using pseudo-code - rather than the syntactic rules of the specific programming language at hand, c) provide usable coding tools such as programming patterns and models for selecting and combining language structures, d) support problem solving settings, e) visualize the program and its output, f) encourage the solution of appropriate tasks and g) provide meaningful feedback (Freund, & Roberts, 1996; Soloway and Spohrer, 1989; Winslow, 1996; Brusilovski, Calabrese, Hvorecky, Kouchirenko, & Miller, 1997; Wallingford, 2001; DiGiano, Kahn, Cypher, & Smith, 2001; Garner, 2007).

Well-known examples of such environments for the learning of programming in computer language C are BACCII (Calloni, B. & Bagert, 1994; 1997) THETIS (Freund, & Roberts, 1996) and 'Karel the Robot' (Pattis, Roberts & Stehlic, 1995). Despite the incorporation of one or more of the principles mentioned in the previous paragraph, together with fundamental principles of modern constructivist theories of learning (von Glasersfeld, 1987; Vygotsky, 1974), these environments either fail to emphasize learner ability and the need to express their knowledge in different representation systems or offer possibilities to solve a limited set of problems. In addition, the reported environments lack the appropriate feedback to help novices meaningfully correct their mistakes.

In an attempt to exploit all the above, an open problem-solving computer learning environment was designed to support secondary level education students in their learning of programming and C (Kordaki, 2006; Kordaki, 2007). This environment is named L.E.C.G.O (Learning Environment for programming and C using Geometrical Objects). The design methodology and the general architecture of L.E.C.G.O are further discussed in Kordaki (2007) while the structure of the learning materials provided (patterns and models) has been presented in Kordaki (2006). L.E.C.G.O. supports the development of students' problem-solving skills by allowing them to perform meaningful activities within the motivating context of drawing using geometrical objects. In this paper, a basic component of L.E.C.G.O – namely, the L.E.C.G.O-Paint component - is presented. L.E.C.G.O-Paint provides support for learners to express their own drawing solutions to given problems by using hands-on experience and to receive feedback in the MRS provided by L.E.C.G.O. In the following section, the architecture and the features of L.E.C.G.O are briefly presented. Next, the design and the features of the L.E.C.G.O-Paint are demonstrated and then this design is discussed in reference to other reported environments for the learning of programming and C. Finally, proposals for future plans are presented.

2. L.E.C.G.O AS A LEARNING ENVIRONMENT

L.E.C.G.O. was designed (Kordaki, 2006; Kordaki, 2007) to be a possible learning environment for twelfth grade students (18 years old) and for first-year University students. The programming language C was selected as a learning subject as this is a modern language with great capabilities which could also become a solid background for the learning of object-oriented programming. The design of L.E.C.G.O. was the result of a synthesis of three models: a) the learning model, based on modern social and constructivist theories of learning (von Glasersfeld, 1987; Vygotsky, 1974) acknowledging the active, subjective and constructive character of knowledge, and the crucial role of tools and of MRS in knowledge construction To this end, the role of digital media in playing a determining role in the whole learning context was acknowledged (Noss & Hoyles, 1996). b) the subject matter model, based on the literature on basic aspects and structures of programming and C, and c) the learner model, based on the literature on how students learn essential aspects of programming. As regards the role of MRS, it is acknowledged that computer learning environments providing a variety of RS of different cognitive transparency could encourage students to select from among them the most appropriate tools to express their knowledge (Ainsworth, 1999; Zikouli, Kordaki & Houstis, 2003; Kordaki, Miadides and Kapsampelis, 2007). These different RS can provide students with opportunities to express their inter-individual and intra-individual variety. It is noteworthy that most learner difficulties are found in the gap between their intuitive knowledge and the knowledge they need to express themselves in the RS proposed for use.

The general architecture of L.E.C.G.O. is depicted in Figure 1 and also demonstrated on the L.E.C.G.O. home page. The aforementioned architecture is divided into two main parts: a) that presenting the appropriate content for learning fundamentals in programming and C (Kordaki, 2006). The content is presented in a five-layered hyperlinked structure including: i) complex examples (1^{st} layer), ii) simple examples (2^{nd} layer) iii) broad information about programming in C (3^{rd} layer), iv) fundamentals in programming (4^{th} layer) and v) broad information at various locations on the WWW (5^{th} layer). The provision of such content gives students the opportunity to study various plans and basic constructs in C as well as some models of programming in C, hyperlinked with fundamental concepts and constructs of the programming language C, and b) that dedicated to the learning activities necessarily performed by students if they are to learn fundamentals in programming and C. This latter part includes tools for algorithmic solutions of problems in MRS such as: i) graphical RS, providing opportunities to solve problems graphically using the LECGO-Paint application tools. The features of this application will be described in this paper. ii) text-based RS providing possibilities for translating the graphical solution given by a student in LECGO-Paint into natural language; iii) imperative RS, providing specific expressions in the imperative which could be used for translation of the solution at hand into the imperative; iv) pseudo-code RS, to translate the solutions expressed in the previously mentioned RS into pseudo-code; v) C language-based RS and vi) the graphic output of the written programs.

L.E.C.G.O. provides students with opportunities to: a) acquire hands-on experience while actively constructing their own graphic problem-solving strategies for the problems at hand, using tools that support the direct manipulation of computational objects on a computer screen, b) express their solution strategies in MRS, starting from intuitive-'anthropocentric' representations and non-necessary programming solutions and gradually moving to more computer-oriented programming solutions, c) deal with a variety of familiar and meaningful problems within the context of drawing using basic geometrical objects. Drawing was selected as a context for the learning activities as it would motivate learners to become actively and passionately involved in their own learning (Zikouli and Kordaki, 2004). In addition, drawing using geometrical objects was selected to give students the chance to learn about the graphic functions in C. The aforesaid activities can be of graded difficulty and can also be solved without the extra cognitive load that might stem from the demand to perform complex geometrical constructions. The role of engaging learners in meaningful and enjoyable learning activities is acknowledged as crucial for the learning of any subject (Nardi, 1996), let alone the learning of fundamental Computer Science (CS) concepts and skills at all levels of education (Bell, Witten, and Fellows, 2002), c) overcome the cognitive load of the syntactical rules of programming in C by using appropriately designed computer-based authoring tools, d) receive information about basic patterns-constructs of programming in general and of the programming language C, e) study essential examples-models in C and d) receive various types of appropriate assistance to correct their mistakes.

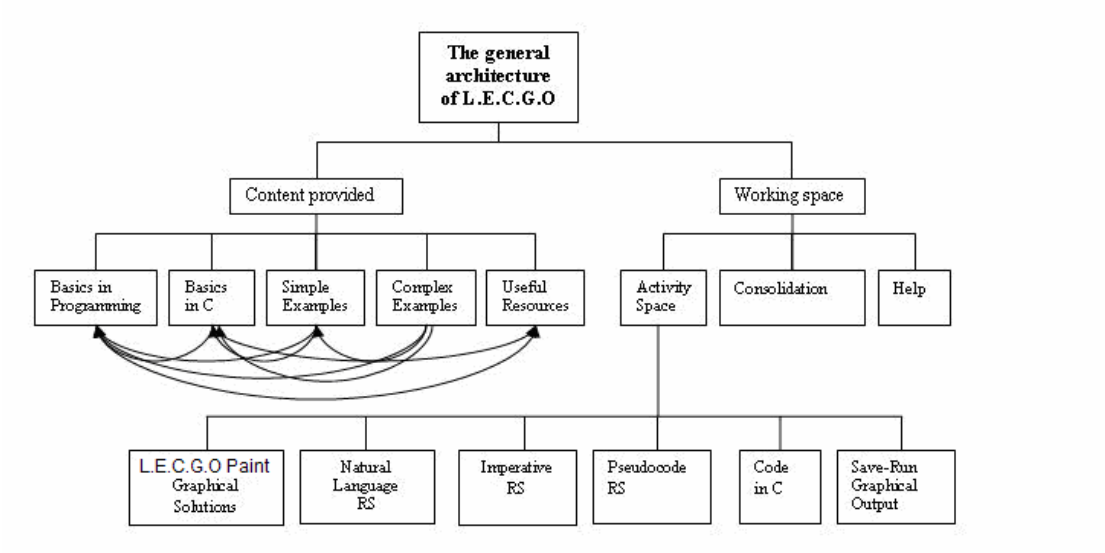

Figure 1. The general architecture of L.E.C.G.O.

This help is provided in four basic modes: i) as ready specific expressions that could be used to describe, in natural language, a specific algorithmic solution to the task at hand, ii) as ready structures and functions in pseudo-code, provided in the form of buttons, iii) as ready structures and functions in C, also provided in the form of buttons, and iv) as feedback in terms of the correct interpretation of a student's graphical solution to the given problem in all the previously mentioned RS provided by L.E.C.G.O. The design of L.E.C.G.O. could be modified for the learning of any programming language.

3. THE L.E.C.G.O-PAINT APPLICATION

The need for the construction of the L.E.C.G.O-Paint application emerged from a field study where a preliminary version of L.E.C.G.O. was piloted using real students. In this version, students were provided with drawing support by the tools provided by Cabri-Geometry II (Laborde, 1990). As it emerged from this study, these students expressed the need to receive feedback from the system to inform them about the correct interpretation of their drawing attempts in all the RS provided by L.E.C.G.O.. As a result, L.E.C.G.O-Paint was designed as both a drawing and a feedback tool. It resembles Microsoft's "Paint", but the difference is that this tool is programming oriented and has been designed in such a way as to meet the needs of programming in Turbo C. This language and its "graphics.lib" library give the user the ability to draw various basic shapes and of course to use all C functions and structures. L.E.C.G.O-Paint corresponds to all the instruction set of this library in a user friendly way. The general interface of L.E.C.G.O-Paint consists of a drawing area, a top and left toolbar and a main menu toolbar at the top of the window (see Figure 2a). The specific features of L.E.C.G.O-Paint can be classified in the following seven main categories: a) drawing shapes b) editing shapes, c) using metrics, d) using loops, e) providing feedback in the MRS provided by L.E.C.G.O, f) help and g) typical file operations, such as: save, print, open etc. These categories are further discussed in the next section of this paper.

Drawing and Editing shapes. L.E.C.G.O-Paint provides the learner with the opportunity to draw shapes such as: line, rectangle, cycle, ellipse, polygon, filled polygon and filled ellipse. The outline of every shape can vary, e.g.: solid line, dashed line, dotted line or center line (consisting of dashes and dots). Moreover, the width of every line can be normal or thick. As far as the shapes with a filling are concerned, the user can use one of the following filling patterns: slash, dashed, backslash, thick slash or thick backslash. Finally, a number of colors can be used as foreground, pattern and background colors. In Figure 2a, the drawing of a railway carriage - using basic geometrical shapes, such as: two filled ellipses, for rectangles and two lines - is presented. In this Figure, different kinds of outlines of the drawn shapes were used.

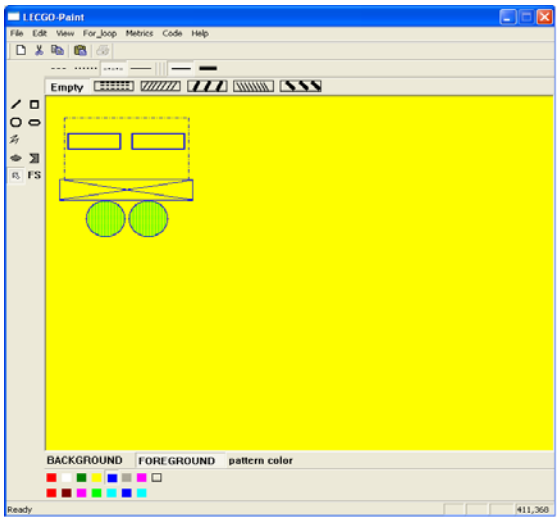

Figure 2a. The general interface of L.E.C.G.O-Paint

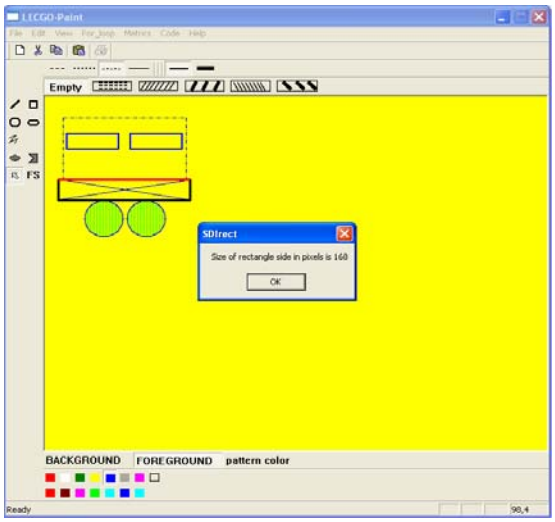

Figure 2b. Example of using the features of 'metrics' menu

Drawing can be performed by using the mouse, based on click and mouse drag. The user should select the desired shape and adjust the line width, color, pattern, etc. If the user presses the Shift button while drawing a shape, a canonical shape is drawn, e.g. a square when drawing a rectangle, a circle when drawing an ellipse, etc. While drawing a shape, the shadow of the shape appears, so that the user can have a preview of the shape. Apart from drawing, the user can *edit* the shapes by using the move, cut, copy and paste functions. All actions can be undone, even from a saved project.

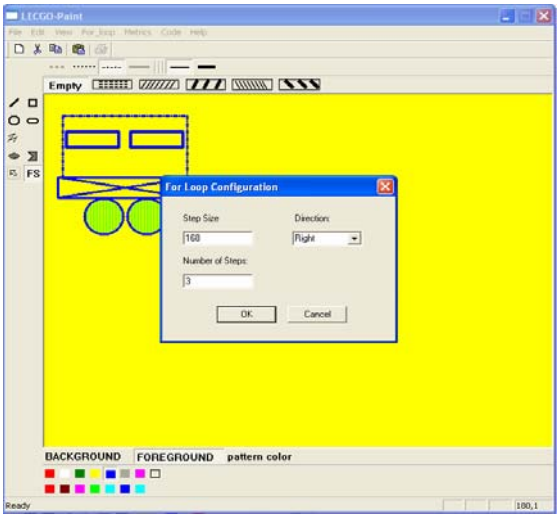

Figure 3a. Selecting a group of shapes and filling the For_loop dialog box

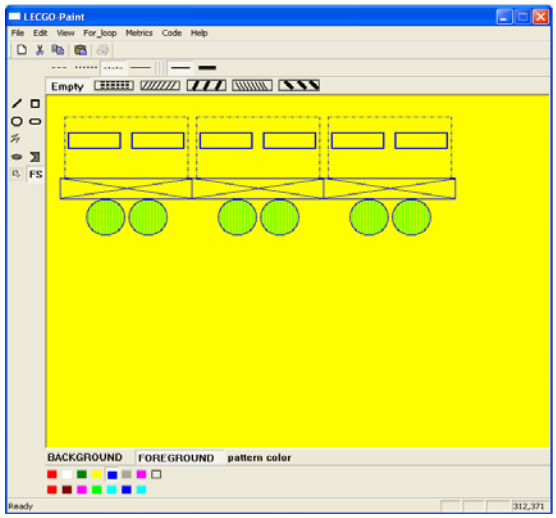

Figure 3b. An example of using the For_loop to create a train

*Metrics***.** The 'Metrics' menu consists of specific features providing the learner with information about specific characteristics of a shape, e.g. the coordinates of the critical points of a shape or its size in pixels (see Figure 3). This information can be helpful to the learner as it can be used in forming the solution in the MRS used in L.E.C.G.O. and especially in the formation of the appropriate pseudo-code and of the code in C. In addition, the information provided by the features included in the 'Metrics' menu, such as the size of a pattern can be used to fill the appropriate field of the for loop dialog box.

Loops. In the LECGO-Paint application, For_loops are simulated in two ways: by repeating a pattern and by moving a pattern. As regards the repeating of a pattern (i.e. a group of shapes), L.E.C.G.O-Paint provides

a tool for selecting a pattern which can consisted of more than one shape. After the pattern is defined, the user specifies the number of repeats, the direction of the repeating pattern (right, left, up and down) and the number of pixels for every step. A visualization of every step of the loop is provided, so that it is clear which action takes place in every loop. For example, users can draw a pattern like a railway carriage and later, using the loop functionality, create a train. An example is shown in Figures 3a and 3b. Specifically, in Figure 3a, all shapes were selected to participate in the For_loop. Next, the loop sub-menu item was selected from the For_Loop menu and all the fields of the related dialogue box were filled with the appropriate info. By clicking ok, the train is drawn on the canvas of L.E.C.G.O-Paint (see Figure 3b). As regards the 'moving a pattern' feature, the user can move it with a delay of some seconds instead of copying this pattern to different places in a certain direction. A pattern is defined in the same way as described above, with the difference that every shape that is patterned is deleted before it is redrawn in another place.

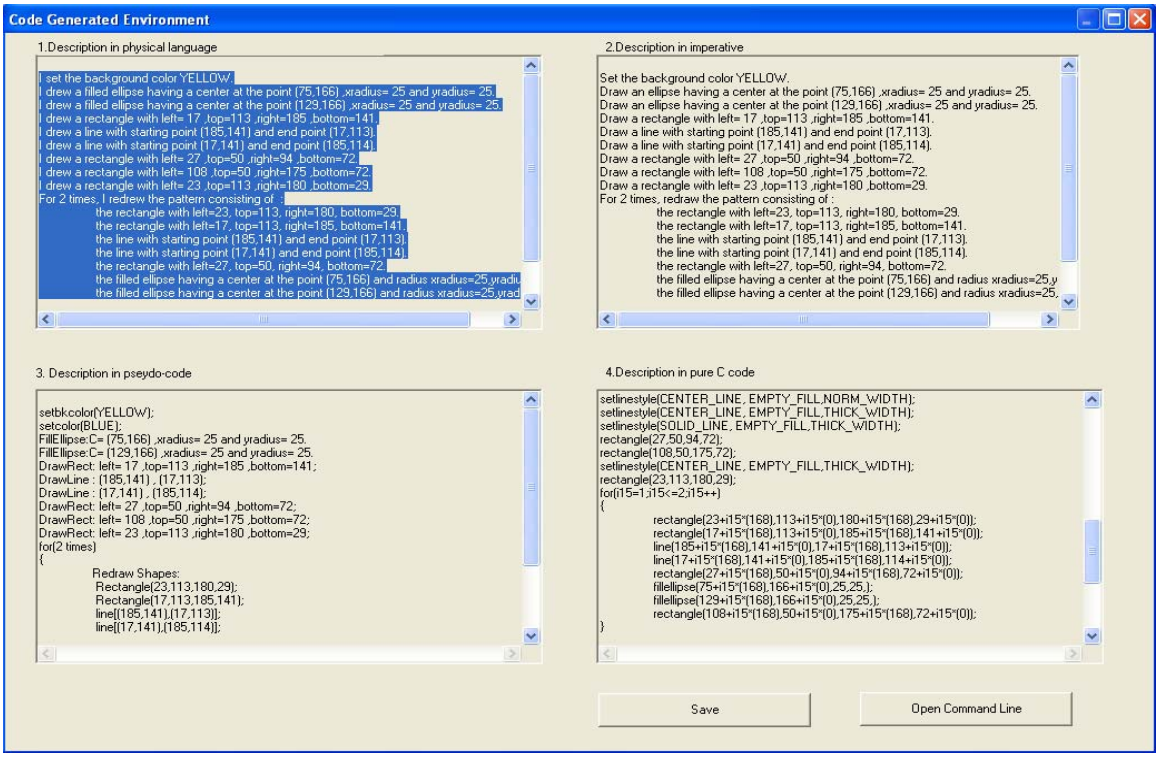

Figure 4. An example of the use of the 'Code Generated Environment'

Providing feedback. By clicking on the "Code' menu, another window pops up and the learner is automatically transferred to the 'Code Generated Environment'. Here, it is worth noting that, when learners try to form a drawing using the tools of LECGO-Paint, all their drawing actions are kept internally in a structure by the system. In addition, all the specific coordinates of the shapes used during the drawing process are also kept internally in the same structure by the system. Thus, at any moment and by clicking only one button, all learner drawing actions can be automatically translated in the MRS mentioned in LECGO, namely; natural language, imperative and pseudo-code and code in C. Learners also have the ability to run the automatically generated code in C that corresponds to their drawing actions. In fact, the feedback provided by the Code Generated Environment feature of LECGO-Paint provides learners with the correct interpretation of their drawing-graphical solutions to the given tasks in MRS. Learners can exploit the information given by this feedback and self-correct by comparing their translations of their own drawing solutions to a given problem (into the previously mentioned MRS) with the corresponding translations automatically generated by the system. The Code Generated Environment illustrates a dialog box, as is demonstrated in Figure 4. Specifically, in this Figure one can see the interpretation of a learner's action regarding the previously mentioned specific problem of drawing a train using a For_loop.

Finally, by pushing a button, the learner can compile the generated C code and run the resulting executable. In this way, learners can confirm that the code automatically generated by the application produces the same results as their hands-on drawings in the LECGO-Paint canvas.

Help. A context-based help file is attached to the LECGO- Paint application. It is a typical .chm file which provides information on all available operations. The structure of the help file follows the main menu toolbar of the application. For each topic, it gives fully detailed instructions and screenshots. Moreover, throughout the application, all buttons have tool tips and an explanatory text appears on the status bar. Finally, if the user tries to perform an invalid action, e.g. pasting without first copying, a message box appears featuring an appropriate informational text.

4. EPILOGUE AND PLANS FOR FUTURE WORK

An interactive computer application, namely the L.E.C.G.O-Paint application, that is part of a wider constructivist computer learning environment for the learning of programming and C, was presented in this paper. L.E.C.G.O-Paint is programming oriented and has been designed in such a way as to meet the needs of learning programming in Turbo C. This language and its "graphics.lib" library enable the user to draw various basic shapes and, of course, to use all C functions and structures. L.E.C.G.O-Paint has been designed to support learners becoming actively involved in their learning and to receive feedback in the form of the correct interpretation of their solutions in multiple representation systems such as: natural language, imperative, pseudo-code and code in the programming language C. In the context of L.E.C.G.O.-Paint, learners are provided with support to actively perform various meaningful and enjoyable learning activities in the context of drawing using simple geometrical objects. In fact, learners have the chance to actively form create their own drawings by using various simple and user-friendly tools. Specifically, to do the previously mentioned activities, various tools are provided to assist in the drawing and editing of various geometrical shapes as well as performing basic algorithmic structures. These tools implement the various graphical functions and structures of the programming language C. Next, learners have to interpret their drawing solutions in the MRS reported above by themselves and to compare the visual outputs of their programs with their drawings formed in the L.E.C.G.O-Paint application so as to implement self-correction. To this end, the L.E.C.G.O-Paint application provides learners with the opportunity to receive feedback in the form of the correct interpretation of their drawing solutions to the given tasks in the said MRS. By exploiting this feedback, learners have the chance to focus on their mistakes, to receive additional information about them and, finally, to make appropriate corrections and adjustments. Such an MRS–based activity and feedback system has not yet been reported. In the context of L.E.C.G.O-Paint, it is expected that learners can experience enjoyment and pleasure by being passionately involved in forming their own drawings and also progress in their learning of programming and C by receiving appropriate feedback to overcome their difficulties. To verify our expectations, field evaluation studies using real pupils are necessary.

REFERENCES

Ainsworth, S.E., 1999. The functions of multiple representations. *Computers and Education*, 33(2-3), 131-152.

Anderson, B. and Soloway, E., 1985. The role of domain experience in software design. *IEEE Transactions on Software Engineering*, Vol. SE-11(1), pp.1351-1360.

Association for Computing Machinery, 2003. *A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12* Task Force Curriculum Committee. Last retrieved on 25/03/08 from http://www.acm.org/education/curric_vols/k12final1022.pdf?searchterm=K-12

Bell, T., Witten, I. & Fellows, M., 2002. *Computer Science Unplugged*, http://www.unplugged. canterbury.az.nz

Brooks, R., 1999. Towards a Theory of the Cognitive processes in Computer Programming. *Int. J. Human- Computer Studies ,*51, 197-211.

Brusilovski, P., Calabrese, E., Hvorecky, J., Kouchirenko, A. & Miller, P., 1997. Mini-languages : a way to learn programming principles*. Education and Information Technologies*, 2, 65-83.

Calloni, B. & Bagert, D., 1994. Iconic programming in BACCII vs. Textual programming: which is a better learning environment?. *ACM, SIGCSE '94* 3/94, Phoenix AZ,188-192.

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 25/03/08 from http://www.ascilite.org.au/ singapore07/ pros/garner.pdf.

Cohen, V.B., 1985. A reexamination of feedback in computer based instruction: Implications for instructional design. *Educational Technology*, pp.33-57.

DiGiano, C., Kahn, K., Cypher, A. & Smith, D.C., 2001. Integrating Learning Supports into the Design of Visual Programming Systems. *Journal of Visual Languages and Computing,* 12, 501-524.

DuBoulay, B., 1986. Some difficulties in Learning to Program. *Journal of Educational Computing Research*, 2(1), 57-73.

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, pp. 300-304.

Hummel, H., G-K., 2006. Feedback Model to Support Designers of Blended-Learning Courses. *International Review of Research in Open and Distance Learning*, 7(3), pp. 1-16.

Komis, V., 2001. A study of basic programming concepts within a constructivist teaching approach. *Themes in Education*, 2 (2-3), 243-270.

Kordaki, M., 2006.΄Learning activity΄ as the basic structural element for the design of web-based content: A case study. In T. Reeves & S. Yamashita (Eds), Proceedings of *World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education (E-Learn 2006),* October, 13-17, Honolulu, Hawaii, USA, pp.88-96. Chesapeake, VA: AACE.

Kordaki, M., 2007. Modeling and multiple representation systems in the design of a computer environment for the learning of programming and C by beginners. In T. Bastiaens and S. Carliner (Eds), Proceedings of *World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education (E-Learn 2007),* October, 15-19, Quebec, Canada, USA, pp.1634-1641, Chesapeake, VA: AACE.

Kordaki, M. Miatidis, M. and Kapsampelis, G., 2008, in press. A computer environment for the learning of sorting algorithms: design and pilot evaluation. *Computers and Education.* doi:10.1016/j.compedu.2007.07.006.

Kurland, D. M. & Pea, R. D., 1989. Children's Mental Models of Recursive Logo Programs. In Soloway & Spohrer (Eds), '*Studying the Novice Programmer'*, 315-324, Hillside, N.J. Erlbaum.

Laborde, J-M., 1990. *Cabri-Geometry* [Software]. France: Universite de Grenoble.

Lemone, K. A. & Ching, W., 1996. Easing into C: Experiences with RoBOTL. *ACM, SIGCSE Bulletin,* Vol. 28, No. 4, 45-49.

Nardi, B.A., 1996. Studying context: A comparison of activity theory, situated action models, and distributed cognition. In B.A. Nardi (Ed.), *Context and consciousness: Activity theory and human-computer interaction*, Cambridge, MA: MIT Press.

Noss, R. & Hoyles, C., 1996. Windows on mathematical meanings: Learning Cultures and Computers. Dordrecht : Kluwer Academic Publishers.

Pattis, R.E., Roberts, J. & Stehlic, M., 1995. *Karel-The Robot, A Gentle Introduction to the Art of Programming*. NY: Wiley.

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, 'Information & Communication Technologies in Education'*, 289-298, Phodes, Greece.

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

Vygotsky, L., 1974. *Mind in society*. Cambridge, MA: Harvard University Press.

Wallingford, E., 2001. *The elementary Patterns Home Page*. Retrieved on 25/03/08 from http://www.cs.uni.edu/~wallingf/patterns/elementary/

Winslow, L.E., 1996. Programming Pedagogy. *SIGCSE Bulletin,* Vol. 28, No. 3, 17-22.

Zikouli, K., Kordaki, M. & Houstis, E., 2003. A Multi-representational Environment for Learning Programming and C. *3rd IEEE International Conference on Advanced Learning Technologies*, (pp. 459), July, 9-11, Athens, Greece, 2003.

Zikouli, K. and Kordaki, M., 2004. Forming an evaluation context for the learning of basic concepts of programming and C through the use of educational software. *4rth Pan-Hellenic Conference 'ICT in Education'(with International Participation).. Athens, Greece, September, 2005, pp*. 598-606.