Beginners' programming attempts to accomplish a multiple-solution based task within a multiple representational computer environment

Maria Kordaki

Dept of Computer Engineering and Informatics
Patras University, 26500, Rion Patras, Greece
e-mail: kordaki@cti.gr

Abstract: This study considers the role of a drawing multiple-representation based computer environment (named L.E.C.G.O: a Learning Environment for programming and C using Geometrical Objects) in beginners' programming attempts to accomplish a multiple-solution based task demanding the use of algorithmic structures, graphic functions and arrays. In fact, nine 18-year old students participated in a comparative learning experiment where they were asked to face similar multiple-solution based tasks in three learning contexts: paper and pencil (p-p), Turbo C and L.E.C.G.O. The analysis of the data shows that all students successfully faced the task at hand within L.E.C.G.O., at the same time using various algorithmic structures and graphic functions. However, fewer students effectively performed within p-p and Turbo C. On the whole, students were given support to effectively perform the said task within L.E.C.G.O. by the opportunities provided to form intuitive graphical solutions and gradually move on - through multiple representation systems - to form formal solutions in C. Students were also given support by the multiple aids provided by L.E.C.G.O. to emphasize the algorithmic solution to the task at hand, at the same time avoiding the cognitive load stemming from the memorization of the syntax of the programming language C.

Introduction

Programming is an important but complex task, including understanding, method finding and coding (Brooks, 1999). According to Winslow (1996), it can be divided into four steps: a) comprehension of the problem at hand, b) definition of a solution to that problem, initially in any form, such as text-based, math-based, pseudo-code and flow-chart, c) translation of that form 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; Christiaen, 1998). Good performance in programming also implies the ability of learners to use various and new representation systems to express their problem-solving strategies (Brooks, 1999; Komis, 2001).

Students often begin programming with a "big handicap": they do not know how a computer works. This lack of knowledge results in many other "deep misconceptions" as well as "surface errors" that appear as programming bugs (Du Boulay, 1986; Soloway & Spohrer, 1989a; Kurland & Pea, 1989; Mayer, 1989). In addition, after students find a solution, they usually tend to omit the "algorithmic solution" and immediately confront the problem (Allwood, 1986). However, there is a big gap between the informal solutions they find and the more formal computer-oriented solution that is required (Winslow, 1996) and the bigger the gap, the more difficult it is for the student to surpass it (Christiaen, 1998). Students also cannot easily identify primitive constructs, such as conditionals, recursions and loops, and entities, such as variables, counters and conditions, in their initial solution; nor can they express them with computer-based structures. However, it is argued that the majority of "surface errors" are caused by these "plan-composition problems" rather than by semantic misconceptions of the language (Spohrer and Soloway, 1989b). Students also have semantic and syntactic misconceptions of almost every basic structure and concept of a programming language (Lemone & Ching, 1996; Putnam, Sleeman, Baxter. & Kuspa, 1989; Samurcay, 1989; Christiaen, 1998; Komis, 2001).

Specifically, the variable is thought to be static and is confused with its algebraic notion. Consequently, the assignment command is interpreted as equality (Samurcay, 1989; Lemone & Ching, 1996; Komis, 2001). The initialization, updating and testing of variables are also difficult for students, especially when they are encountered inside loops and conditions (Samurcay, 1989). Regarding conditionals, students fail to understand that the conditions are static yet the entangled variables change and, as a result, the same condition can have a different result at each repetition of the loop. Moreover, students seem to interpret "*While… do*" statement as a kind of exception within the program (Lemone & Ching,

1996). There are also important difficulties in understanding the way input and output commands function (Putnam, Sleeman, Baxter. & Kuspa, 1989) and, according to Du Boulay (1986), students find it difficult to manipulate arrays. Finally, students find it difficult to comprehend and use the online help provided by the typical programming environments (Allwood, 1986).

Programming in schools is currently supported by professional integrated software environments. Despite their capabilities and aids for the experienced users, these tools are useless for novices with limited programming skills: they provide insufficient feedback, and they are also extremely complicated (Freund, & Roberts, 1996). Thus, the student's frustration with programming often depends less on the target programming language and more on the programming environment in use. Existing studies indicate that there is a need to support novices' learning by providing them with opportunities to: (a) emphasize algorithmic logic instead of the syntactical rules and grammar of a specific programming language (b) use simple coding tools for selecting and combining the language structures, c) be involved in problem solving settings, d) explore visualization of the program and its output, e) be involved in authentic and meaningful learning activities and g) receive meaningful feedback (Freund, & Roberts, 1996; Sangwan, Korsh & LaFollete, 1998; DiGiano, Kahn, Cypher, & Smith, 2001; Brusilovski, Calabrese, Hvorecky, Kouchirenko, & Miller, 1997; Kordaki, 2007).

In an attempt to exploit all the above, an open problem-solving computer learning environment was constructed to support secondary level education students in their learning of programming and C. The programming language C was selected as a learning subject, this being a modern language with great capabilities which could also become a solid background for the learning of object-oriented programming. This environment is named L.E.C.G.O. (Kordaki, 2006, 2007; 2008) and its features are briefly reported in the following section of this paper, followed by the context of a learning experiment. In this experiment, the impact of L.E.C.G.O. on student learning within the context of a task supporting the construction of multiple programming solution strategies is investigated. At this point it is worth noting that appropriate learning activities play a crucial role in motivating learners to be actively involved in their learning (Nardi, 1996). To this end, multiple-solution based activities performed within contexts providing multiple tools and multiple representation systems can play an essential role in encouraging learners to express their inter-individual differences regarding the concepts to be learned (Kordaki, 2003). The results emerging from this experiment are then presented and, finally, discussion points and conclusions are drawn.

Basic features of L.E.C.G.O

L.E.C.G.O. was designed to be a possible learning environment for basic aspects of programming and C by twelfth grade students (18 years old) as well as for first-year University students (Kordaki, 2006; 2007; 2008 submitted). The focus when designing L.E.C.G.O. was on the learning of basic algorithmic structures in C. L.E.C.G.O. provides students with opportunities to: (a) perform authentic, meaningful, open and graded learning activities, within the motivating context of drawing using geometrical objects, while at the same time avoiding the extra cognitive load that usually comes from complex problem domains. In fact, 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. In addition, drawing using simple geometrical objects was selected to give students the chance to learn about the graphic functions in C. The aforesaid activities can be solved in various ways, motivating students to express their inter- and intra-individual differences (Calloni & Bagert, 1994). (b) emphasize algorithmic logic instead of the syntactical rules and grammar of a specific programming language by providing ready language plans and constructs, c) acquire hands-on experience in providing graphic solutions to the problems at hand, using specific tools (d) express their solution strategies in MRS (Zikouli, Kordaki & Houstis, 2003), and (e) offer assistance with their problem-solving strategies so as to emphasize algorithmic logic. This support is provided in five modes: i) as ready specific expressions that could be used to describe, in natural language, a specific algorithmic solution to the tasks 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 , iv) as appropriate language plans and constructs integrated within a specifically designed content, and v) as visual feedback from the output of student programs, to be compared with their graphic solutions to the problem at hand, so as to verify the correctness of their programs. The general architecture of L.E.C.G.O. is demonstrated on its home page and is divided into two main parts:

a) That presenting an appropriate multilayered, hyperlinked, multimedia, multi-representation

system based content for learning fundamentals in programming and C (Kordaki, 2006). This content is structured in a hierarchical order of five hyperlinked layers, including: complex examples (1st layer), simple examples (2nd layer), broad information about programming in C, (3rd layer), fundamentals in programming (4th layer) and broad information at various locations on the WWW (5th layer).

b) That dedicated to the learning activities that could be actively performed by the students in order to learn fundamentals in programming and C. This part includes tools for algorithmic solutions to problems in MRS, such as: i) *Drawing-visual* RS, providing students with opportunities to express their intuitive knowledge by forming graphical drawing solutions to the tasks at hand using the tools provided by the well-known educational software Cabri-Geometry II (Laborde, 1990) and also easily trace the coordinates of the drawn shapes so as to incorporate them into their programs; ii) *Free-text* based RS, providing possibilities for reflecting on the graphic solution given by a student in Cabri and translating it into the familiar symbolic system of natural language, iii) *Text-based* RS, using the imperative, providing specific expressions which could be used for translation of the solution at hand into the imperative. The provision of this RS has been decided upon to help students move from the 'I' or 'we' situation to one in which the student gives directions to the computer; iv) *Pseudo-code* RS, to translate the solutions expressed in the previously-mentioned RS into pseudo-code. Students are helped to form their programs in pseudo-code using a set of buttons. Each of these buttons shows a basic algorithmic structure or a graphic function in C; v) *Representations in C*. Here as well, students are encouraged to form their programming solutions in C by using a number of buttons showing the corresponding code regarding basic components of a program in C, such as program-skeletons, algorithmic structures and basic graphic commands; and vi) *The graphic output* of the written programs. On the whole, students are provided with 'transitional' RS to gradually move from intuitive graphic solutions to solutions using pseudo-code and, ultimately, to solutions using the programming language C. Figure 1 shows the solution in the previously-mentioned RS of the following simple drawing problem: '*develop a program in C to draw three circles with the same centre and three different radii'*.

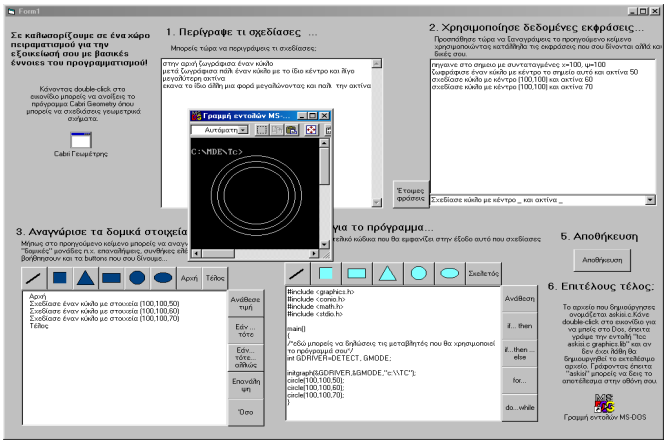

Figure 1. The general interface of the activity space integrated within L.E.C.G.O

The context of the learning experiment

This study is part of a wider experimental, comparative evaluation study of L.E.C.G.O (Kordaki, 2008) aiming to: a) provide evidence of student programming constructions using the tools and RS provided by L.E.C.G.O. and b) to compare these constructions to those realized by the same students while performing similar tasks in the paper and pencil (p-p) environment and in the typical programming environment of Turbo C. The second aim was selected to clarify the specific contribution of L.E.C.G.O. to student learning about basics in programming and C in comparison to the aforesaid environments. In terms of methodology, this research is a qualitative work which can also be characterized as a case study (Cohen and Manion, 1989). The learning experiment took place in a technical secondary school located in Patras, Greece, in which 9 twelfth grade students participated. These students did not have any knowledge about C; however, the previous year, they had attended a course on the programming language Pascal. The students in this sample had mixed learning achievement in terms of the previous year's school grades.

Students were set four learning tasks within p-p, Turbo C and L.E.C.G.O. Students worked individually within each of these learning contexts. To confront these tasks, students were provided

with paper-based information to be used in both p-p and Turbo C while, within L.E.C.G.O., the integrated content was used. To perform the tasks in both Turbo C and L.E.C.G.O., students worked in a computer laboratory equipped with nine computers. The researcher provided appropriate information to answer student questions while at the same time taking care not to affect student solution strategies to the given tasks. The questions posed by the students and the answers they received from the researcher were recorded and are presented in the "Results' section of this paper. Students were asked to complete the first set of four tasks in p-p and one month later performed these tasks again within Turbo C. The performance of tasks within L.E.C.G.O. was realized two months later. This was decided upon to prevent students from memorizing solving strategies used in one learning context and transfering them to the other two contexts. The duration of student involvement in the three learning contexts was according to their needs.

The aforementioned three sets of four tasks were holistic and open within the context of drawing, using simple geometrical objects. The learning tasks across the three learning contexts were similar but not identical. However, the aims of each task were different. Specifically, the aim of the 1st task was to encourage student learning about basic aspects in C, such as the structure and syntax of a serial program in C and the use of its basic graphic functions. The aim of the 2nd task was to encourage students to become familiar with the concept of iteration structures and also with the concept of the variable in programming. The 3rd task was more demanding than the others, challenging students to learn the use of iteration structures in combination with the control statements as well as the use of variables and of arrays in C. Finally, the 4th task provided students with the opportunity to construct different solution strategies by using a combination of various algorithmic structures and the appropriate use of variables and arrays. The study presented in this paper focuses on student programming constructions within the context of this last type of task. This task, set to be performed in the context of paper and pencil and Turbo C, was presented to the students as follows: '*Write a program that, when executed, will display a shape of your choice moving on your computer screen and leaving a specific trail behind it. Create a program that will enable this trail to draw the initial letters of your name and surname*'. In the context of L.E.C.G.O. students were asked to: '*Write a program that, when executed, will display a shape of your choice moving on your computer screen and leaving a specific trail behind it. Create a program that will enable this trail to be customized'*. The data resources included the field notes of the researcher during the whole experiment, as well as the students' work sheets, their paper-drawings, their computer programs, their written work in the RS provided by L.E.C.G.O. as well as the log-files that automatically recorded their actions.

Results

The data emerging from the previously-mentioned type of tasks are organized in terms of student programming problem-solving strategies performed within: i) p-p, ii) Turbo C 2.01, iii) L.E.C.G.O. and are shown in Table 1 in terms of: a) the program's completeness/incompleteness, i.e. full/partial programming description of the drawing solutions given by the student, b) correct/incorrect syntax, c) correct/incorrect coordinates used.

Table 1. Student programming problem-solving strategies to perform a multi-solution based task

Strategies	Description
P1	Forming a complete program, using correct syntax and correct coordinates. The program runs and provides appropriate results that match the students' initial drawings within Cabri.
P2	Forming an incomplete program, using correct syntax and correct coordinates Thus, the program runs but does not provide the appropriate results to match students' initial drawings within Cabri.
P3	Forming an incomplete program, using correct syntax and incorrect coordinates. Thus, the program runs but does not provide the appropriate results to match students' initial drawings within Cabri.
P4	Forming an incomplete program, using incorrect syntax and correct coordinates. Thus, the program does not run.
P5	Forming an incomplete program, using incorrect both; syntax and coordinates. Thus, the program does not run.
P6	Failure to form a program and use of neither graphical functions nor algorithmic structures. Sometimes a weak and verbal solution is formed or the skeleton of a program in C is copied from the learning materials given.

Next, the analysis of these strategies in each environment was performed in terms of: a) the type of programs constructed b) correctness of the written programs, c) student difficulties, d) student questions, e) graphic functions used and f) MRS used. Student programming problem-solving strategies to face this task within p-p, Turbo C and L.E.C.G.O. are presented in Table 2.

Table 2. Student programming problem-solving strategies to face a multiple solution-based task

Students' programming problem solving strategies to face a multiple solution-based task								
Paper and pencil environment			**Turbo C 2.01**			**L.E.C.G.O.**		
AS & GF used	**Strat. used**	**N/S.**	**AS & GF used**	**Strat used**	**N/S**	**AS & GF used**	**Strategy used**	**N/S**
Circle	P1, P2, P4, P5	1,1,1, 2	For-circle	P1	1	For-circle	P1:D1-NL-IM5-PC1-P1 P2:D2-P1 P3:D1-NL-IM3-P1 P3:D1-NL-IM2-P1	1 1 1 1
Line	P4	1	Circle	P1,P2, P3	3,1, 1	For-draw Drawpol.	P1:D1-NL-IM1-PC1-P1	1
Drawpoly	P5	1	None	P6	2	For- rect.	P1:D1-NL-IM2-PC3-P1	1
None	P6	2				Circle	P1:D1-NL-IM1-PC1-P1 P1:D1-NL-IM5-PC1-P1 P3:D1-NL-IM4-P1	1 1 1

a) Paper and pencil environment

Type and correctness of programs written by the students: The majority of the students constructed serial programs, although 2 students gave up. It is worth noting that only one student successfully managed to complete this task. The rest of them formed incomplete programs, using correct syntax and correct coordinates (1 student), incorrect syntax and correct coordinates (2 students) as well as incorrect syntax and incorrect coordinates (3 students). Two students also used verbal descriptions of the '*for-to'* structure for the repetition of a circle.

Student difficulties: Here as well, students faced difficulties regarding: a) the calculation of appropriate coordinates (4 students). Here, it is worth noting that students who drew a figure were helped to calculate correctly the coordinates needed to form their solution strategy (3 students), b) the syntax of the graphic functions used. Specifically, the wrong number of arguments was employed (2 students) in using the functions '*line*', and '*drawpoly*'. As regards the use of the function '*circle*', a value was correctly assigned to its arguments. Despite this, some (2 students) used both these arguments and their corresponding values to define the use of the said function. Other students used a Greek word instead of the appropriate word 'radius' (2 students), c) the use of repetition statements. Students tried to use serial programming instead of iteration statements, d) the changing of the coordinates of the centre of a circle so as to display a circle moving on the computer screen and leaving a trail behind it, to draw the students' initials. To realize this task, some (2 students) formed 34 statements assigning the same and constant values to the coordinates (x, y) for the centre of this circle.

Student questions: Student questions during their involvement in this task within p-p were focused on: a) the structure of a program in C, b) the syntactical rules of C, and c) how to calculate the correct coordinates to incorporate them in their programs.

*Graphic Functions appropriately used: T*he function '*circle*' was successfully used by only one student.

b) Turbo C

Type and correctness of programs written by the students: Only one student successfully used the repetition structure '*for-circle'* to face this task, while the rest formed serial programs. Two students were unable to face this task and gave up. The function '*circle*' was used within: a) complete and correct programs; within a correct repetition structure (1 student), b) complete and correct serial programs (3 students), c) incomplete serial programs using correct syntax and coordinates (1 student), and d) incomplete, serial programs using incorrect coordinates (1 student).

Student difficulties: About half of the students (4 students) demonstrated difficulties in estimating the appropriate coordinates but overcame these difficulties by trial and error. One student also overcame problems with the repetition structure by trial and error. These problems were to do with the number of repetitions and the quantum of the increment of a variable used.

Student questions: Student questions during their involvement in this task were focused on: a) the syntactical rules of C, b) the structure of a program in C, c) how to calculate the correct coordinates to incorporate into their programs, and d) how to use the condition statement and the repetition structures.

Graphic Functions used: To deal with this task, the '*circle*' function was successfully used by three students and in combination with the '*for-to'* structure by one student.

It is worth noting that students did not manage to overcome the difficulties reported above within both p-p and Turbo C, despite the interventions made by the researcher in the form of explanations and the provision of specific examples included in the book-like learning materials.

c) L.E.C.G.O.

The abbreviations presented in the 8th column of Table 2 were used for the description of the problem-solving strategies constructed by the students through the use of MRS provided by L.E.C.G.O. These abbreviations (AB/used) are explained in Table 3.

Table 3. Student solution strategies across MRS provided by L.E.C.G.O.

AB/used	Student solution strategies across MRS provided by L.E.C.G.O.
P1	Path followed: Cabri ⇒ Natural Language⇒ Imperative⇒ Pseudocode⇒ C
P2	Path followed: Cabri ⇒ C
P3	Path followed: Cabri ⇒ Natural Language⇒ Imperative ⇒ C
D1	Using Cabri to draw a figure that is a complete solution to the given problem.
D2	Using Cabri to draw a figure roughly that is an incomplete solution to the given problem.
NL	Use of Natural language for a weak description of the graphical solution to the given problem.
IM1	Complete description of the graphical solution given by using the ready specific expressions in the imperative provided by L.E.C.G.O. Correct coordinates are also used, with the help of Cabri.
IM2	Incomplete description of the graphical solution given by using the ready specific expressions in the imperative provided by L.E.C.G.O. Correct coordinates are used (with the help of Cabri).
IM3	Incomplete description of the graphical solution given by using the ready specific expressions in the imperative provided by L.E.C.G.O. Incorrect coordinates are used.
IM4	Complete description of the graphical solution to the given problem in the imperative. Incorrect coordinates are also used (the assistance of Cabri is not taken into account).
IM5	Incomplete description of the graphical solution to the given problem in the imperative. Incorrect coordinates are also used (the assistance of Cabri is not taken into account).
PC1	Complete description of the graphical solution to the given problem in pseudocode. Correct coordinates are also used, with the assistance of Cabri.
PC2	Incomplete description of the graphical solution to the given problem in pseudo-code. Correct coordinates are also used, with the assistance of Cabri.

Type and correctness of programs written by the students: Here as well, more than half (6 students) successfully used the '*for-to'* structure in combination with the graphical functions: '*circle*' (4 students), '*drawpoly*' (1 student) and '*rectangle*' (1 student). The remaining students (3 students) correctly formed serial programs using the '*circle*' function. It is worth noting that all students formed complete and correct programs to deal with this task within L.E.C.G.O.

Student difficulties: In facing this task, students found it difficult to: (a) initialize the variables used (3 students), (b) understand the dynamic nature of a variable (1 student), (c) define the appropriate number of repetitions within a repetition structure (1 student), (d) correctly define the quantum of increment of the value of a variable used in a repetition or a sequential structure (1 student). These difficulties were overcome with the support that students received from the solved examples provided within the content integrated within L.E.C.G.O. as well as by comparing the graphical outputs of their programs to the graphical solutions they constructed within Cabri.

Student questions: Taking into account their whole experience within L.E.C.G.O., students managed to confront their difficulties on their own during their attempts to realize this task.

Graphic Functions used: During this task, students successfully used the combinations: '*for-circle'* (4 students), '*for-drawpoly'* and '*drawpoly*' (1 student), '*for-rectangle'* (1 student) and '*circle*' (3 students).

Student use of MRS provided by L.E.C.G.O: Here as well, there are four categories of MRS use by the students: (a) use of all the RS provided, at the same time using the Imperative RS in various ways. Specifically, some (2 students) formed complete solution strategies using the specific expressions provided and integrating correct coordinates of the shapes used. These students formed correct programs in C using the structures '*for-drawpoly'* and '*circle*'. Some others (2 students) formed incomplete descriptions - in the imperative - of their drawing without coordinates. However, all these students formed correct and complete programs using the structures '*for-circle'* and '*circle*', (b) use of

all RS provided except pseudo-code, at the same time using the Imperative RS in various ways. In fact, some formed incomplete descriptions of their drawing solution strategies using/not using appropriate coordinates (1 student/2 students). It is worth noting that all these students successfully managed to form programs in C using the iteration structure '*for-circle'*, (c) use of Cabri and immediately forming successful programs in C, also using the '*for-circle'* structure (1 student), and (d) use of all RS provided except both pseudo-code and Imperative RS, where one student formed incomplete descriptions of their graphic solutions without using any coordinates. Despite this fact, this student successfully constructed a program in C using the '*for-rectangle'* structure.

Discussion points and concluding remarks

This study presented a comparative learning experiment, investigating the role of MRS on beginners' programming attempts within the context of a learning task that could be accomplished in various ways, i.e. using various algorithmic structures and graphic functions as well as variables within arrays. Specifically, students were asked to face similar multiple-solution based tasks in three learning contexts: p-p, Turbo C and L.E.C.G.O. Analysis of the data shows that: (a) all students found being involved in the said task attractive, especially within L.E.C.G.O., because they had the opportunity to draw in the context of the computer and also to form their own solutions, (b) more students successfully performed the task at hand within L.E.C.G.O. (all students) than in p-p (1 student) and Turbo C (4 students), (c) more algorithmic structures and graphic functions were used by the students within L.E.C.G.O. than in both other environments. This was probably due to the fact that L.E.C.G.O. provides buttons automatically illustrating the code of various algorithmic structures and graphic functions, (d) students faced difficulties with the formation of iteration structures as well as with the appropriate use of both variables and arrays within p-p and Turbo C. However, they solved these difficulties within L.E.C.G.O. where they were supported by the ready examples provided, including language plans and constructs as well as appropriate animations. On the whole, the analysis of the data shows that programming is a complex task, especially for beginners. Consequently, they needed some motivation to be passionately involved. In addition, this complex task is difficult for beginners to perform within an unaided environment such as paper and pencil. However, beginners also ran into difficulties performing the task at hand within Turbo C because they were provided with support only for the syntactical errors they made. Contrariwise, students were provided with multiple aids in their programming attempts within L.E.C.G.O. In fact, students were given emotional support to become involved in the task at hand, because the formation of a drawing solution was easy and also part of its demands. Students were also encouraged to continue to be involved in the solution process, as the interpretation of the drawing solution in both natural language and the imperative exploited the use of RS familiar to them. To this end, students could smoothly progress to the formation of pseudo-code especially where various aids in the form of buttons were provided to help them to avoid syntactical errors. Finally, students were given support to successfully form their programs in C by studying the information provided in the form of ready examples and essential language plans and constructs and by using the ready templates provided.

References

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

Brooks, R. (1999). Towards a Theory of the Cognitive processes in Computer Programming. *International Journal of 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.

Christiaen, H. (1998). Novice Programming Errors: Misconceptions or Misrepresentations? *ACM, SIGCSE Bulletin*, Vol. 20, No. 3, 5-7.

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.

Du Boulay, 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,300-304.

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

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

Kordaki, M. (2003). The effect of tools of a computer microworld on students' strategies regarding the concept of conservation of area. *Educational Studies in Mathematics*, 52, pp. 177-209.

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. (2008, submitted). A drawing and multi-representational computer environment for beginners' learning of programming and C: Design and formative evaluation. *Computers and Education*.

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.

Mayer, R. E. (1989). The Psychology of How Novices Learn Computer Programming. In Soloway & Spohrer (Eds), '*Studying the Novice Programmer'*, 129-160, Hillside, N.J. Erlbaum.

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.

Putnam, R. T., Sleeman, D., Baxter, J.A. & Kuspa L.K. (1989). A Summary of Misconceptions of High-School BASIC Programmers. In Soloway & Spohrer (Eds), '*Studying the Novice Programmer'*, 301-314, Hillside, N.J. Erlbaum.

Samurcay, R. (1989). The Concept of Variable in Programming: Its Meaning and Use in Problem Solving by Novice Programmers. In Soloway & Spohrer (Eds), '*Studying the Novice Programmer'*, 161-178, Hillside, N.J. Erlbaum.

Sangwan, R. S., Korsh, J. F. & LaFollete, P. S. (1998). A system for programming visualization in the classroom. *ACM, SIGCSE '98* , Atlanta GA USA, 272-276.

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

Spohrer, J. C. & Soloway, E. (1989b). Novice Mistakes: Are the Folk Wisdoms Correct?. In Soloway & Spohrer (Eds), '*Studying the Novice Programmer'*, 401-416, Hillside, N.J. Erlbaum.

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.