Computers & Education 54 (2010) 69–87

Contents lists available at ScienceDirect

Computers & Education

journal homepage: www.elsevier.com/locate/compedu

A drawing and multi-representational computer environment for beginners' learning of programming using C: Design and pilot formative evaluation

Maria Kordaki *

Department of Computer Engineering and Informatics, Patras University, 26500 Rion Patras, Greece

ARTICLE INFO

Article history:
Received 18 August 2008
Received in revised form 12 July 2009
Accepted 13 July 2009

Keywords:
Programming
C
Interactive learning environments
Multiple representation systems
Secondary education

ABSTRACT

This paper presents both the design and the pilot formative evaluation study of a computer-based problem-solving environment (named LECGO: Learning Environment for programming using C using Geometrical Objects) for the learning of computer programming using C by beginners. In its design, constructivist and social learning theories were taken into account. The general design has taken into consideration models of the learning process and subject matter as well as potential learner behaviour in dealing with fundamental tasks. The main emphasis has been placed on the role of: (a) multiple external representations in student learning, (b) motivation, through performing problem-solving activities taken from the familiar and meaningful context of drawing, using simple geometrical objects, (c) the active participation of students in their own learning by using hands-on experience, (d) appropriate feedback on the actions taken by students, to aid their self-correction, and (e) holistic, activity-based, multi-media, multi-representational and multi-layered content for the learning of basic concepts of programming using C. LECGO was pilot evaluated in the field through a qualitative and comparative study where nine 12th grade (18-year-old) students participated. In fact, students faced three similar yet not identical sets of four tasks across three learning environments, namely; paper and pencil (p–p), Turbo C and LECGO. The data emerging from this field evaluation study indicates that students gain better results within LECGO than in both the p–p environment and the typical programming environment of Turbo C, while performing similar activities.

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 (CS) as an academic and professional field (National Research Council Committee on Information Technology Literacy, 1999). To this end, a four-level model curriculum of CS for K-12 has been proposed (Association for Computing Machinery, 2003). 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 taught to use CS skills and concepts – especially algorithmic thinking – in their problem-solving activities in diverse 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 problem-solving skills can be encouraged in learners. In fact, programming is more a mental skill than a body of knowledge (Hadjerrouit, 2008). It is a complex task, including understanding of the task at hand, method finding, coding, testing and debugging of the resulting program (Brooks, 1999). Here, it is essential to note that, students encounter serious difficulties in performing any or all of the above (Allwood, 1986; Christiaen, 1998; Du Boulay, 1986; Hadjerrouit, 2008; Lemone & Ching, 1996; Pacheco, Gomes, Henriques, de Almeida, & Mendes, 2008; Robins, Rountree, & Rountree, 2003; Soloway & Spohrer, 1989; Winslow, 1996). Many CS students cannot grasp the most fundamental concepts of programming, unable to produce even the most basic programs (Eckerdal, 2009). Programming problems come from a wide range of problem domains and understanding the problem domain is also critical (Anderson & Soloway, 1985). Moreover, novices seem to lack the ability to

* Tel.: +30 2610 996932; fax: +30 2610 990006.
E-mail address: kordaki@cti.gr

0360-1315/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compedu.2009.07.012

combine the individual statements and constructs of a programming language into valid programs (Eckerdal, 2009; Soloway & 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 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 (Brooks, 1999; Komis, 2001; 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 of little use 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 (Freund & Roberts, 1996). To this end, it has been acknowledged 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).

Well-known examples of such environments for the learning of programming in computer language C are BACCII (Calloni & Bagert, 1994, 1997), THETIS (Freund & Roberts, 1996) and 'Karel the Robot' (Pattis, Roberts, & Stehlic, 1995). Despite the incorporation of one or more of the features mentioned in the previous paragraph, together with fundamental principles of modern social and constructivist theories of learning (Jonassen, 1999; Vygotsky, 1978), these environments either fail to emphasize learner ability and the need to express their knowledge in different representation systems or offer possibilities to solve only a limited set of problems. In addition, the reported environments lack the appropriate feedback to help novices meaningfully correct their mistakes.

However, various studies indicate that a need for a novice-oriented programming environment still exists. 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 assistance in terms of appropriate information and also meaningful feedback (Freund & Roberts, 1996; Brusilovski, Calabrese, Hvorecky, Kouchirenko, & Miller, 1997; DiGiano, Kahn, Cypher, & Smith, 2001; Garner, Haden, & Robins, 2005; Sangwan, Korsh, & LaFollete, 1998; Soloway & Spohrer, 1989; Wallingford, 2001; Winslow, 1996; Garner, 2007).

In fact, appropriately-designed computer learning environments can provide learners with great and unique opportunities for the learning of all learning subjects during the entire learning process (ACM, 2003; Jonassen, 1999; Noss & Hoyles, 1996). To this end, many researchers acknowledge the critical role of constructivist educational software in student learning in terms of: (a) acquiring hands-on experience, at the same time reflecting and translating this experience in multiple representation systems (MRS), (b) expressing their knowledge, including their mistakes, taking advantage of MRS of different cognitive transparency, (c) exploring dynamic visualization of complex procedures such as specific algorithms and programs, (d) obtaining hyperlinked, multilayered and multi-media information, including specific programming language constructs and plans, and (e) receiving appropriate intrinsic and extrinsic feedback in MRS (Ainsworth, 1999; Jonassen, 1999; Kordaki, Miatidis, & Kapsampelis, 2008).

In an attempt to exploit all the above, an open problem-solving computer learning environment has been constructed to support secondary level education students in their learning of programming using C. This environment is named LECGO (Learning Environment for programming using C using Geometrical Objects) and its design was based on social and constructivist perspectives (Jonassen, 1999; Vygotsky, 1978) regarding knowledge construction, with an emphasis on the role of: (a) algorithmic logic instead of the syntactical rules and grammar of a specific programming language, (b) performing meaningful learning activities within the motivating context of drawing using geometrical objects, at the same time avoiding the extra cognitive load that usually comes from complex problem domains, (c) hands-on experience in solving programming problems and subsequently in the role of reflection and translation of this experience in MRS, (d) constructing algorithmic solutions to problems in various Representation Systems (RS), starting from intuitive, more 'anthropocentric' solutions, and gradually moving to more computer-oriented solutions, (e) appropriate help, including language plans and constructs in a multi-layered hyperlinked content, using the learning activity as a basic structural element, and (f) appropriate feedback on student learning. The features provided by LECGO are not only theoretically documented but have been pilot tested in the field using actual students to verify their impact on student learning. Such a computer environment for the learning of programming using C has not yet been reported.

The main research aims motivating this study are:

- to explore ways of designing educational software that effectively supports beginner learning of programming using C (mainly at secondary education level)
- to examine the impact of the proposed software on student conceptualization of basic aspects of programming in C

The following section features the rationale behind the design of LECGO and its general architecture and features are then demonstrated. Subsequently, a pilot formative evaluation study of LECGO using real students is reported. The results from this study are then discussed in comparison to the results that emerged from the same students' involvement in similar tasks performed in the p–p environment and also in the context of Turbo C. Finally, conclusions are drawn.

2. The rationale behind the design of LECGO

In order to meet the first research aim of this study – namely, to explore ways of designing educational software that effectively supports secondary level education student learning of programming using C – it emerged that it was crucial to address certain vital questions:

- What is the subject to be learned, in terms of the basic concepts that have to be learned by the students?
- What is the target group of learners that LECGO will address?
- What is their profile?
- How they can learn the specific learning subject?

Consequently, it became essential to find an appropriate context of meaningful and appropriate activities for the previously mentioned students. Addressing these questions is not an easy task, since it implies some knowledge of learner profiles, in terms of how they learn the specific subject, and some awareness of the epistemological issues referring to knowledge construction and the learning subject under consideration. However, in the case of the construction of educational software, providing theoretical answers to the above questions will not suffice. Due to the fact that educational software is a technological artifact, the translation of these theoretical answers into software specifications is necessary. At this point, it is worth noting that this translation is not unique but reflects the designers' views on these theoretical issues.

To answer the above questions, three theoretical models were constructed; namely, the learner model, the subject matter model, and the learning model. The design principles of LECGO arose as a transformation of the theoretical considerations of these models. Computer modeling has been acknowledged as a valid methodology in the design of educational software (Kordaki et al., 2008; Mellar & Bliss, 1994). Specifically, the construction of the learner model was based on the literature on how students learn essential aspects of programming, while the formation of the subject matter model was based on the literature on basic aspects and structures of programming using C. The learning model was based on modern social and constructivist theories of learning, as well as on the crucial role of tools and of MRS in knowledge construction (Ainsworth, 1999; Jonassen, 1999). An early description of these models is presented in Kordaki (2007). The entire modeling process, including the theoretical considerations taken into account during the construction of these models, as well as their interpretations in terms of operational specifications of the software, is presented in the following section of this paper.

LECGO was designed to be a possible learning environment for beginners in programming using C; in fact, for 12th-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.

2.1. The learner model

Students often begin programming with a "big handicap": they do not know how a computer works (Eckerdal, 2009; Hadjerrouit, 1998; Wulf, 2005). 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, 1989). Indeed, students try to explain the functionality of the computer by using anthropomorphic expressions (Dagdilelis, Satratzemi, & Evangelidis, 2004; Soloway & Spohrer, 1989). Thus, lacking a schema that can incorporate new concepts and explain others, students begin programming with an important innate disability.

A considerable amount of research has reported serious student difficulties in understanding the algorithmic logic in problem solving (Komis, 2001; Soloway & Spohrer, 1989). In fact, it seemed that students persist in referring to their previous knowledge of problem solving; emphasizing other methodologies from learning other subjects, such as mathematics, physics, etc. In programming, students have difficulty understanding the whole dynamic context of an algorithmic methodology that focuses on the formation of an appropriate process to obtain the preferred results. Furthermore, learning programming language constructs and problem solving through programming demands a solid foundation in algorithms and other essential mathematical skills. In a nutshell, mathematical maturity/immaturity seems to have an impact on learning programming language constructs (Hu, 2006; Pacheco et al., 2008).

According to Winslow (1996), programming 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 pseudocode and flow-chart, (c) translation of the solution into a selected programming language, and (d) testing and debugging of the resulting program. It is acknowledged, that students encounter serious difficulties in performing all the steps mentioned above (Allwood, 1986; Christiaen, 1998; Du Boulay, 1986; Eckerdal, 2009; Hadjerrouit, 2008; Lemone & Ching, 1996; Pacheco et al., 2008; Robins et al., 2003; Soloway & Spohrer, 1989; Winslow, 1996).

As programming problems come from different problem domains, many students may lack the necessary domain background but, most importantly, there are also many other students who can solve a problem with pen and paper but lack the ability to express the solution in computer terms (Winslow, 1996). In addition, some students know the syntactical rules of specific statements and constructs of a specific programming language but do not know how to combine these features into valid programs (Eckerdal, 2009; Soloway & Spohrer, 1989; Winslow, 1996). Some other students, after finding a solution, usually tend to omit the "algorithmic solution" and confront the problem directly (Allwood, 1986; Kolikant & Pollackm, 2004).

Research in computing education also suggests that learning to program is hard and that students often have trouble identifying 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 (Eckerdal, 2009; Guzdial & Ericson, 2005). There is a big gap between the informal solutions they give 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). Soloway and Spohrer (1989) argued that the majority of "surface errors" are caused by these "plan-composition problems" rather than by semantic misconceptions of the language. Students also have semantic and syntactic misconceptions of almost every basic structure and concept of a programming language (Britos, Rey, Rodríguez, & García-Martínez, 2008; Christiaen, 1998; Komis, 2001; Lemone & Ching, 1996; Samurcay, 1989; Soloway & Spohrer, 1989).

Specifically, the variable is thought to be static and is confused with its algebraic notion. Consequently, the assignment command is interpreted as equality (Christiaen, 1998; Komis, 2001; Lemone & Ching, 1996; Samurcay, 1989). The initialization, updating and testing of variables are also difficult for students, especially when they are encountered inside loops and conditions (Lahtinen, Ala-Mutka, & Järvinen, 2005; Samurcay, 1989).

Regarding conditionals, programmers fail to realize the interaction between the serial execution of the program and the condition (Komis, 2001; Lahtinen et al., 2005). The condition cannot easily be defined, especially when it uses logical variables or variables that change inside the body of the conditional (Lahtinen et al., 2005; Soloway & Spohrer, 1989). 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*" as a type of exception within the program (Lemone & Ching, 1996). Regarding recursion, studies have shown that repetition and loops are more understandable than recursion and that there is no transfer of knowledge from the latter to the former (Lahtinen et al., 2005; Soloway & Spohrer, 1989). There are also important difficulties in understanding the way input and output commands function (Soloway & Spohrer, 1989) and, according to Du Boulay (1986), students find it difficult to manipulate arrays. In addition, not only do inexperienced users have difficulties in using a programming language but, in contrast to those with experience, they also find it difficult to comprehend and use the online help provided by the environment (Allwood, 1986).

Furthermore, it has been acknowledged that, even if some progress has been made in solving some learning problems using constructivist learning theories (Ben-Ari, 2004; Wulf, 2005) 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).

With the above in mind, an attempt has been made to exploit the conclusions of the aforementioned studies through the formulation of a number of operational specifications for the design of LECGO so as to help beginners in programming overcome the previously mentioned difficulties. The specifications used for the construction of LECGO related to the design of such tools that would provide learners with opportunities: (a) to solve programming problems from such a simple and interesting domain as that of drawing, so as to avoid the cognitive load emerging from solving problems coming from other complex domains, (b) to explore practical solutions to these problems by using hands-on experience in order for them to be aware of their solving actions, (c) to interpret their hands-on problem-solving strategies in a number of RS – including text using natural language, natural language in the imperative, pseudo-code and code – so as to help them move gradually from informal solutions to more computer-oriented ones (Zikouli, Kordaki, & Houstis, 2003), (d) to receive help by exploring how basic constructs and plans of a specific language are used in various meaningful examples, (e) to explore dynamic visual representations of essential cognitive opaque programming constructs such as iteration structures, (f) to use specific authoring tools to write their solutions in pseudo-code and in C, thus avoiding the extra cognitive load stem from the need to memorize the specific commands of this language and to focus on the algorithmic solution of the problem at hand, and (g) to receive constructive intrinsic feedback to aid self-correction by comparing their drawing solutions to the outputs of their written programs.

2.2. The subject matter model

Traditional learning theories emphasize the learning of a programming language through its syntactical rules, starting with the easier ones and gradually moving onto more advanced matters, while modern social and constructivist learning theories emphasize the holistic approach of problem solving using algorithmic logic (Komis, 2001). Thus, the teaching of programming using C should stress the basic structures and not the syntactical rules of the language.

The following hierarchical network (Fig. 1) depicts the process of solving a problem using C. Each level of the process includes its subprocesses until it reaches the elementary ones, which can be performed using the specific commands of the target programming language. The division of the process has three objectives: (a) to allow students to realize the structure of the programming process in C, (b) to develop their own problem-solving strategies using elementary processes and the related commands, and (c) to make it possible to transform these elements into aids or templates within a programming environment.

Taking into account the results of the modeling process performed in the previous section (Section 2.1), and to help students acquire the appropriate knowledge needed in the process of problem solving in C, it was decided to provide them with: (a) various templates to be used as elementary authoring tools to construct a solution in both pseudo-code and C, while avoiding the cognitive load of recalling the specific syntactical rules and the vocabulary of this language and (b) appropriately-structured information using the learning activity as a basic structural element. This information is expressed in the previously mentioned MRS, including text, pseudo-code and code, hyperlinked with appropriate visual animations of the programming constructs used. Students can elect to study the appropriate examples that reflect

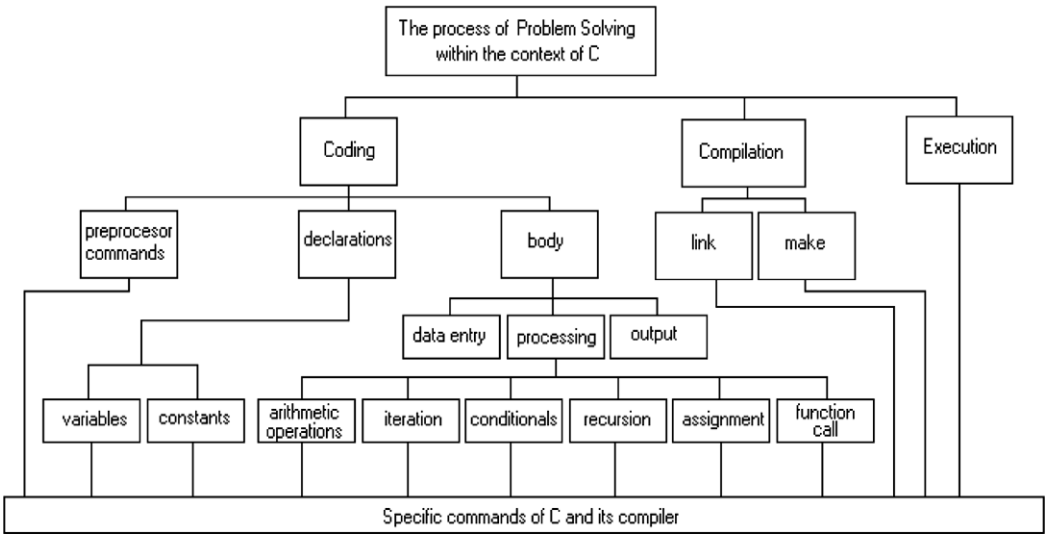

Fig. 1. The problem-solving model within the context of programming and C.

their knowledge so as to be aware of the specific constructs of the language and to see how they can be combined to produce effective solutions to problems. The organization of this information is further discussed in the next section of this paper (Section 3.1).

2.3. The learning model

The design of the proposed learning environment was based on modern social and constructivist theories of learning which require learning to be re-conceptualized as an *active*,[1] *subjective* and *constructive activity*, emphasizing the role of *computer tools* as cognitive tools in the learning process (Jonassen, 1999; Maureen, 2000; Vygotsky, 1978). To this end, some researchers propose that the solving of the problems associated with the learning of introductory programming by novices requires a radical change from traditional instructional methods to constructivist learning environments (Ben-Ari, 2004; Wulf, 2005). Constructivist design emphasizes the *fundamental and timeless concepts* of the learning subject in question and not its details (Nardi, 1996; Vygotsky, 1978). In the context of such design, the role of providing students with the ability to *express, represent and organize their own knowledge* is essential (Jonassen, 1996). To this end, the role of appropriately-designed computer tools is crucial.

Social learning theories emphasize the role of psychological tools and especially of computer tools as mediators in the development of students' higher mental functions (Noss & Hoyles, 1996). In fact, computer tools have been acknowledged as mind-tools that can engage and facilitate learners' cognitive processing and critical thinking (Jonassen, 1996). In the context of appropriately-designed computer environments, students can express their own knowledge, *explore the knowledge of others integrated in these environments* and receive appropriate *intrinsic and extrinsic feedback* as well as *scaffold their thinking and actions* in order to deepen understanding (Noss & Hoyles, 1996). Tools, and especially computer tools, can act as mediators of socio-cultural development to encourage students to enhance the Zone of Proximal Development (ZPD) that is essential in the development of their knowledge (Crooks, 1994; Vygotsky, 1978). Computer environments can also support both experiential and reflective thinking in students (Norman, 1993) by providing external support through the use of *various representation systems*. In fact, MRS can play an essential role in student learning (Ainsworth, Wood, & Bibby, 1996; Kaput, 1994).

Specifically, it is acknowledged that computer learning environments that provide *a variety of RS of different cognitive transparency* could encourage students to select the most appropriate RS to express their knowledge. These different RS can provide students with opportunities to express their inter- and intra-individual variety (Janvier, 1987). It is noteworthy that most learner difficulties are found in the gap between their intuitive knowledge and the knowledge they need to express in the RS proposed for use. For example, prepositional, symbolic and abstract RS prevent some learners (usually beginners) from expressing their knowledge, the same systems being intended for use by advanced learners. Contrariwise, metaphors of everyday life and visual RS are more suitable for beginners. To this end, *the use of various RS can support students to express their solutions in the most cognitive transparent RS and gradually translate these solutions in more cognitive opaque RS such as a program in a specific programming language*.

Hypermedia and multimedia can also play a crucial role in student learning as well as in the holistic organization of a learning content (CTGV, 1992). In particular, the selected learning activities can be hyperlinked with their multi-media applications, as well as *animations and simulations*, providing learners with opportunities to study their evolution (Davidovitch, Parush, & Shtub, 2006; Jonassen, Strobel, & Gottdenker, 2005). In addition, these activities can be hyperlinked with text-based information for an in-depth understanding of their surrounding theoretical context. The significant role of animations for the learning computer algorithms has been also acknowledged (Lahtinen & Ahoniemi, 2009; Naps et al., 2003).

In the context of these modern learning theories, holistic approaches to the organization of learning context are also acknowledged. In fact, *holistic, real life, problem-solving learning activities* are viewed as appropriate for use in various learning settings, as well as in the organization of the content presentation (Nardi, 1996) and especially for the learning of computer programming (Lahtinen & Ahoniemi, 2009). To this end, *activities stemming from the learners' worlds* can motivate them to become involved actively in their own learning. Consequently, emphasis is placed on the essential and necessary activities to be performed by students to effectively learn the subject matter. *Student difficulties* in the understanding of the concepts in focus also need to be taken into account. Furthermore, *multiple-solution activities* that can help learners express their inter-individual and intra-individual variety is of essential interest (Kordaki & Balomenou, 2006). Holistic learning approaches and multiple-solution based activities are also of great interest in the learning of concepts in CS and especially in programming (Ellis, 1998; Hadjerrouit, 2008; Kordaki, 2006).

The role of engaging learners in *meaningful and enjoyable learning activities* in terms of increasing student motivation and confidence, is also acknowledged as crucial for the learning of any subject (Jonassen, 1999; Nardi, 1996), let alone the learning of fundamental CS concepts and skills including computer programming (Bell, Witten, & Fellows, 2002; Teague, 2009). Specifically, researchers report that students involved in *artistic tasks* may exhibit higher academic achievement than their peers who are not involved in such tasks, possibly because of knowledge transfer but also due to increasing engagement and motivation in their learning (Burton, Horowitz, & Abeles, 2000; Csikszentmihalyi, 1997; Deasy, 2002). In fact, artistic tasks have been acknowledged as particularly important if students are to experience the joy of creating, developing attention to detail, and learning ways of expressing thoughts, knowledge, and feelings beyond words (Eisner, 2002; Greene, 1995). The benefits of the engagement in artistic tasks for the development and well-being of adults and children alike have also been widely reported (Gardner, 1983; Greene, 1995). Finally, artistic contexts were recommended to motivate students in being actively and passionately engaged in their learning of computer programming (Guzdial & Ericson, 2005).

Taking into account the specifications arrived at during the construction of the previously mentioned models (Sections 2.1 and 2.2) and the requirements of the constructivist learning model described in this section, an attempt has been made to interpret the aforesaid theoretical issues in terms of operational specifications necessary for the design of LECGO. It should be stressed that these interpretations are not unique, but express the designer's views on these issues. Therefore, it was decided to provide learners with opportunities to:

(i) play an *active role* in the learning process, by providing: (a) a highly interactive environment that allows them to do things rather than passively receive information, (b) diverse tools to use, to enable them to be actively involved in programming activities,

[1] Italics are used in this section, to highlight the key aspects considered as appropriate to be taken into account as operational specifications for the design and implementation of LECGO.

(c) motivational *holistic, real life, artistic, authentic, multiple-solution-based learning activities,* from the *familiar enjoyable and meaningful* context of drawing. Student difficulties in programming can be smoothly dealt with in the performance of such motivational activities.

(ii) express their *subjective* knowledge, by providing tools for expressing both their previous knowledge and other types of knowledge. To this end, the provision of: (a) tools for 'drawing-visual representations using hands-on experience' (a further description of these tools is presented in Section 3.2) has been viewed as essential for the construction and study of their own personal drawing strategies related to the given tasks and (b) *MRS of different cognitive transparency*, such as: drawing, free text, text in imperative, pseudo-code and code in C (a detailed description of these MRS is presented in Section 3.2) has been seen to be significant in allowing students to *express, represent and organize* the different kinds of knowledge they possess by selecting the most appropriate RS for their cognitive development and also constructing MR of their drawing strategies.

(iii) play a *constructive role* in their learning process by providing: (a) tools to support the realization of drawing solutions using *hands-on experience* (a further description of these tools is presented in Section 3.2), (b) learning resources for programming using C, in the form of *hypermedia* and *multimedia,* as well as *animations, simulations and inter-linked visual RS* (an analytical description of these resources is presented in Section 3.1). By exploring these resources, students can *explore the knowledge of others* integrated within them, (c) such *scaffolding tools and materials* for student thinking and actions as: ready specific expressions in imperative, programming skeletons, authoring tools and templates, as well as primitive programming plans and constructs, to be used in the formulation of a program in pseudo-code and in C (a further description of these tools is presented in Section 3.2), (d) *scaffolding* materials and resources in terms of appropriate examples with solutions (see the resources described in Section 3.1), (e) *intrinsic visual feedback* to help students self-correct by comparing their drawing solutions with the visual output of their corresponding programs in C, and (f) *extrinsic feedback* provided by the compiler of Turbo C for the estimation of syntactical errors.

(iv) emphasize *fundamental and timeless concepts* of programming and C by stressing algorithmic logic and pseudo-code as described in the previous section (Section 2.2).

3. The general architecture and features of LECGO

The general architecture of LECGO is presented diagrammatically in Fig. 2 and also demonstrated on the LECGO home page. The aforementioned architecture is divided into two main parts: (a) that presenting an appropriate content for learning fundamentals in programming using C (Kordaki, 2006) and (b) that dedicated to the learning activities that have to be actively performed by students in order for them to learn fundamentals in programming using C. These parts are described in the next section of this paper.

3.1. The learning content integrated in LECGO

This content is comprised of holistic, activity-based, multi-media, multi-representational and multi-layered web-based information materials for the learning of programming using C, with emphasis on basic algorithmic structures. The role of presenting students with opportunities to study programming solutions to appropriate problems has been acknowledged as essential (Hadjerrouit, 2008). The proposed content is structured in a hierarchical order of five layers which have been further discussed in Kordaki (2006) and which are briefly presented below:

3.1.1. 1st layer: presentation of complex examples for the learning of programming using C

In these pages, a student is given the possibility to study examples for programming in C with holistic characteristics, e.g.: *select/draw a shape/pattern and write a program that uses it to fill your computer screen.* By studying this example, students have the chance to gain some knowledge about data input/output and the use of nested 'for . . . next' loops. However, one could face this problem by using nested 'do . . . -

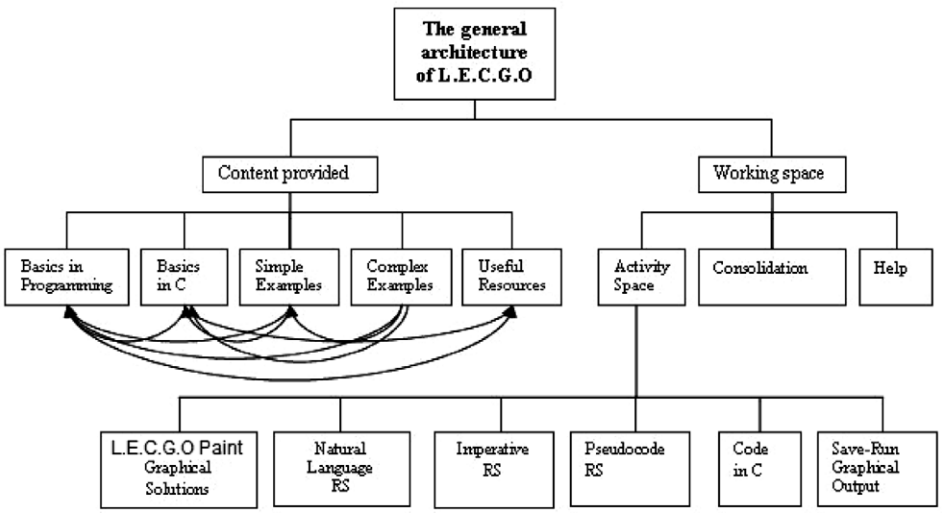

Fig. 2. The general architecture of LECGO.

while'. This example was selected to provide students with opportunities to study multiple-solution based problems that can help them to enhance their knowledge of the learning concepts in focus (Kordaki & Balomenou, 2006). Students also can explore the solutions provided for this task and subsequently invent their own solutions. For better understanding of this example, this is hyperlinked to other simple examples, namely the 1st and 2nd simple examples presented in the next section (2nd layer). In fact, the presentation of the solution of a problem starts with its graphic representation and moves gradually onto its representation in the programming language C, through the RS mentioned in the previous section (Section 2.1). This was chosen to help students move from intuitive graphic solutions to solutions using pseudo-code and, ultimately, to solutions using the commands of a programming language such as C.

3.1.2. 2nd layer: presentation of simple examples for the learning of programming using C

These examples are presented to assist students in their understanding of fundamentals in programming using C and their understanding of the complex examples presented in the 1st layer mentioned in the previous section. Some simple examples included in the 2nd layer of the proposed content are presented below:

3.1.2.1. Example 1. 'Write a program that draws 20 concentric circles. The centers of these circles are to be drawn on the point with coordinates (100, 200), the radius of the first circle being 10 pixels, that of the second circle 20 pixels, etc.' By studying this example, students have the opportunity to gain some knowledge of the 'for . . . next' statement (Fig. 3). The large number of concentric circles – 20 circles – was selected to provide students with the opportunity to understand the difference between two programs: one constructed using a sequential structure and another using an iteration structure. In particular, it should be emphasized that when using both of these type of programs the computer does the same work. However, when a program includes iteration structures it becomes more sophisticated, more convenient, easier to understand and the whole procedure is easier to correct.

3.1.2.2. Example 2. 'Fill the first line of your screen with a shape of your preference'. The aim of this example is to provide students with the chance to learn about the appropriate use of the iteration structure 'do . . . while' when the number of iterations is unknown beforehand.

3.1.3. 3rd layer: presentation of basic aspects and topics for the learning of programming in C

Through the pages included in this layer, students are provided with opportunities to access detailed information related to basic concepts and functions in C, namely: (a) description of the language C, (b) basic steps in forming an executable program in C, (c) the structure of a program in C, (d) variables in programming using C, (e) algorithmic structures in C, (f) data input/output in programming using C, (g) essential graphic characteristics of the programming environment using C, and (h) graphic functions in C. These web-pages feature an index that assists learners to access the specific information they require easily. The sub-set of the language C to be presented through these web-pages has been selected as that which is appropriate to support students in solving essential and fundamental problems in programming using C so that they may present drawings using basic algorithmic structures and simple geometrical objects.

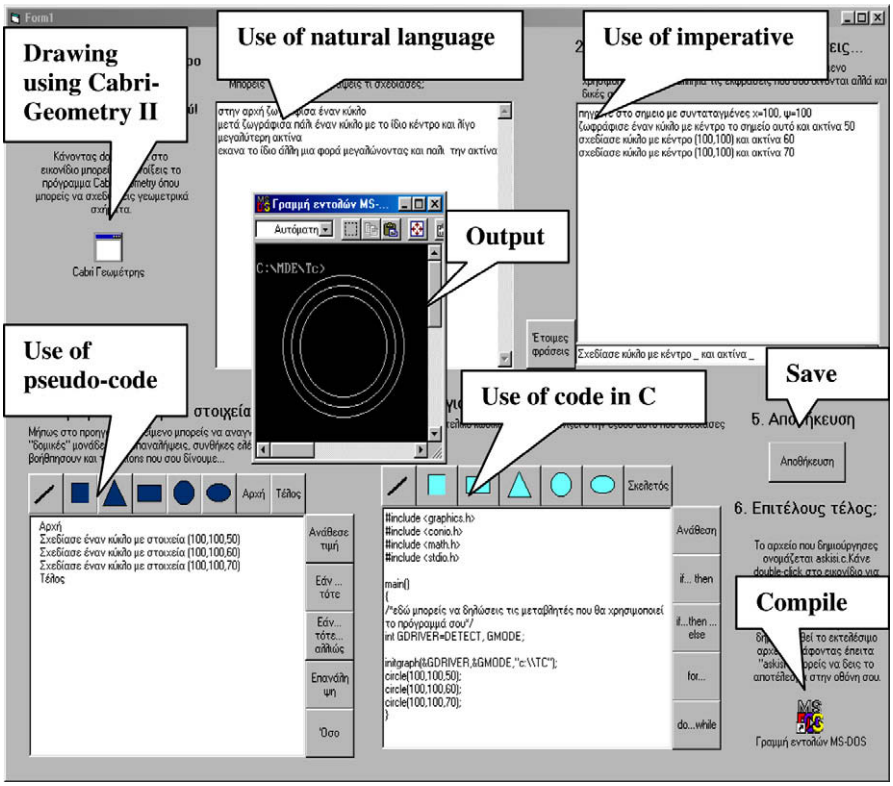

Fig. 3. The general interface of the working space integrated in LECGO.

3.1.4. 4th layer: presentation of fundamentals for the learning of programming

This layer was created to assist students to re-think basic aspects of programming and computer science and includes the necessary background for the learning of fundamentals in C. The information provided is organized in alphabetical order and presented through an index hyperlinked with the appropriate data.

3.1.5. 5th layer: presentation of broad information for the learning of programming using C at various locations on the WWW

A number of online resources – such as e-books, e-manuals, and URLs – are provided to assist learners to develop a broad view of programming using C. Some examples are presented below: (a) Books: 'The programming language C', By B. Kernighan, D. Ritchie, Klidarithmos Eds, http://www.klidarithmos.gr/, (b) e-manuals: http://www.strath.ac.uk/IT/Docs/Ccourse/, (c) URLs: http://www.programmersheaven.com/zone3/index.htm.

In fact, students can access this last layer of information from each of the remaining layers using the navigation bar included on all pages of this content. However, it was decided not to hyperlink this layer to the others so as not to overload students with a plethora of hyperlinks and especially with those associated with a huge amount of information. 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 said programming language. All examples and topics integrated into this content can be accessed through a specially-designed index. In fact, each page provides possibilities to return to the home page and also displays an index for easy transition to the preferred example or topic. Learners can exploit the multiple media and various forms of the information integrated into LECGO to succeed in the tasks at hand.

3.2. The working space integrated in LECGO

This part includes tools to help students form algorithmic solutions to problems from the domain of drawing, express these solutions in MRS and also receive intrinsic feedback on their actions. In fact, the construction of the solution to a problem begins with its graphic representation and moves gradually onto its representation in the programming language C, through various RS. This was chosen to help students move from intuitive graphic solutions to solutions using pseudo-code and, ultimately, to solutions using the commands of a programming language such as C. These MRS are described below:

3.2.1. Drawing-visual representations using hands-on experience

Students can express their intuitive, drawn representations of the solutions to the given problems by using any of the tools provided by the well-known educational software Cabri-Geometry II (Laborde, 1990). Within Cabri-Geometry II, students can be helped to build various drawing constructions by acquiring hands-on experience. In fact, Cabri provides various tools for intuitive construction of basic geometrical figures such as; points, segments, lines, circle, polygons, rectangles, squares, etc. More sophisticated constructions can also be formulated by using the previously mentioned tools and specific macros can also be formed to illuminate these constructions automatically on the computer screen. These geometrical constructs can be described in the programming language C by using its functions. In addition, students can also trace the coordinates of these constructions and therefore use them as values of the related parameters in the corresponding functions in C. The modification of the constructions formed within Cabri can also be easily and intuitively performed by students using the 'drag mode' operation.

3.2.2. Free-text-based representations

Here, students have the opportunity to express their solution strategies using the familiar symbolic system of natural language. This step calls for reflection on their previous hands-on experience (Kordaki et al., 2008) in the context of Cabri-Geometry II. Here, it is worth noting that reflection is an essential process in the development of a student's critical thinking. The text produced is a 'transitional' representation that can, as with the previously mentioned graphic representation; act as a scaffolding element to help students to gradually develop the ability to express their problem-solving strategies in C.

3.2.3. Text-based representations using the imperative and specific expressions

Here, students have to express their free-text-based solution strategies in the imperative. This has been decided upon to help students to move from the 'I' or 'we' situation to one in which the student gives directions to the computer. The system also provides students with a number of specific expressions in the imperative, to select those most appropriate for the description of their solutions. These expressions are related to the specific available actions that can be performed within Cabri-Geometry II. These are not immediately visible but appear to help students if they are unable to take this step unaided.

3.2.4. Pseudo-code based representations

This is the step where students have to express their imperative-based representations in pseudo-code, using a set of buttons. Each of these buttons shows a basic algorithmic structure or geometrical figure (those that can be represented using a graphic function in C in the corresponding text-box.

3.2.5. Representations in C

Providing a number of authoring tools was seen as being useful to students. A number of buttons showing the basic component skeletons of a program in C, e.g. '*basic structure of a C program*', '*for*', '*while*' and '*if*" statements, as well as those showing basic graphic commands e.g. '*line*', '*circle*', '*rectangle*', *'ellipse'* and *'triangle*', are offered in this editor. By clicking on these buttons, students have the possibility to view the corresponding code. All these tools can help students to focus on the essential elements of the language, reducing the cognitive load arising from the need to recall the specific syntactic rules and commands of the language.

3.2.6. Graphic output of the written programs

After students have finished writing the code, they can compile the program and see the results of their programming attempts in another window on the computer screen. Students can verify their programming attempts by comparing this output with their drawing solutions performed using the tools of Cabri-Geometry II.

Fig. 3 shows the solution in the previously mentioned RS to the following simple drawing problem: *'develop a program in C to draw three circles with the same center and three different radii'*.

To realize LECGO, Microsoft Visual Basic 6.0. was used. The final program generated in LECGO is compiled by the "Turbo C 2.01" compiler. Every action of the student is also recorded in a log file, providing the researcher with a valuable bank of raw data.

Finally, before ending this section, it is considered appropriate to discuss the differences between the features provided by LECGO in relation to the features provided by the well-known LOGO programming language (Papert, 1980). Specifically, LOGO is an educational environment that supports constructivist learning through drawing. In the context of LOGO, students have the chance to instruct a turtle appropriately – by using a few commands – to draw (mainly geometrical) figures on the computer screen. In this way, students have the opportunity to think about their own thinking, by expressing themselves in their programs and then debugging these programs until they work, at the same time acquiring higher order thinking skills (Papert, 1980).

Contrarily, within LECGO students can firstly enjoy the drawing aspect, and then reflect on their drawing actions so as to be aware of them and transform them in the MRS mentioned above so as to progress smoothly in expressing themselves in C using a number of commands. By comparing their drawing attempts with the visual outputs of their programs, they can debug them until they work. During the whole process, students have the chance to acquire higher order thinking skills. However, at this point, it has been acknowledged (Guzdial, 2003), that each novice programming environment is attempting to answer the question: 'What makes programming hard?' Obviously, there are a great many answers to this question. For each answer, there are a great many potential environments that act upon that answer, and then there are a great many other potential environments that deal with multiple answers to that question. This is not surprising, since it is almost certainly true that there is no one correct answer to the question that applies to all people (Guzdial, 2003).

4. The pilot formative evaluation study of LECGO

4.1. The theoretical framework and the aims of the study

The aims of this evaluation study were: (a) to provide evidence about student programming constructions using the tools and RS provided by LECGO and (b) to compare these constructions to the ones realized by the same students while performing similar tasks in the p–p environment and in the typical programming environment of Turbo C. This was decided because in school practices students are very frequently asked to write programs using paper-and-pencil. The second aim was selected to clarify the specific contribution of LECGO to student learning about basics in programming using C in comparison to the aforesaid environments.

To meet the previously-mentioned aims, the following procedures were decided upon: (a) to compare the programming strategies constructed by students in these three learning environments, (b) to investigate student difficulties in performing the given tasks in each learning context, (c) to investigate the progress of student programming strategies for each task across the three learning contexts, (d) to investigate the progress of each individual student's programming strategies for each context across the four learning tasks, and (e) to establish the amount of time spent in each learning environment by the students and study how it is spent. In terms of methodology, this research is a qualitative work which can also be characterized as a case study (Cohen & Manion, 1989).

4.2. The learning experiment

The learning experiment took place in a technical secondary school located in Patras, Greece. In particular, nine 12th-grade students participated in this evaluation experiment. These students were attending specific classes dedicated to providing them with special knowledge of computing suitable for their employment in related jobs. These students did not have any knowledge of C; however, they had some knowledge about computer programming. In fact, the previous year, they had attended – at school – a specific course on the programming language Pascal. The nine students in this sample had demonstrated mixed learning achievement in terms of the previous year's school grades.

Students were set four types of learning tasks, in three learning contexts, namely: (a) the p–p classroom context, (b) the typical environment of Turbo C, and (c) the LECGO environment. Students worked individually within each of these learning contexts. To confront the tasks at hand, students were provided with paper-based information to be used in the p–p environment and in Turbo C. Students used the proposed multilayered, multimedia and activity-based content to deal with the set tasks in the context of LECGO. To perform the tasks in the contexts of Turbo C and LECGO, students worked in a computer laboratory equipped with nine computers. The researcher provided appropriate information, so student questions related to the use of the programming language C could be answered at the same time, taking care not to affect student solution strategies to the given tasks. The specific questions posed by the students and the answers they received by the researcher were recorded and are presented in the appropriate parts of Section 5 of this paper.

Students were asked to complete the first set of four tasks in the p–p environment; subsequently, 1 month later, they were asked to perform these tasks in the context of Turbo C, while the performance of tasks within LECGO was realized after a further month had passed. This strategy was chosen so as to prevent students from memorizing methods and solving strategies used in one learning context and transferring them to the other two contexts. The duration of student involvement in the three learning contexts was tailored to their needs. The time allowed per task/learning context differed according to student differences, and the use of time for each task/learning context will be further discussed in Section 5 of this paper.

The data resources included the field notes of the researcher recording student interactions performed in the three learning contexts of this experiment, student work sheets and their paper-drawings, the computer programs that students constructed during this study, the students' written work in the RS provided by LECGO as well as the log-files that automatically recorded student actions within LECGO.

4.3. The tasks

The aforementioned three sets of four types of tasks were holistic and open within the context of drawing, using simple geometrical objects. Each type of tasks posed across the three learning systems had the same aims and students needed the same kind of knowledge to deal with them successfully. Although the tasks of the same type posed across the three learning systems were similar, they were not identical, so as to minimize the possibilities for memorization of their solution strategies and transfer from the one learning setting to the other. Furthermore, the aims of each type of task/learning context were different. Specifically, the aim of the 1st type of tasks was to encourage student learning of 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 type of tasks was to provide students with opportunities to familiarize themselves with the concept of iteration structures and also with the concept of variables in programming. The 3rd type of tasks was more demanding than the rest, challenging students to learn the use of iteration structures in combination with control statements, and to learn the use of variables and arrays in C. Finally, the 4th type of tasks provided students with the opportunity to construct different solution strategies by using combinations of various algorithmic structures and appropriate use of variable arrays and iteration structures. Specifically, in the context of paper and pencil and Turbo C, students were asked to 'write a program that, when executed, will display on your computer screen the following drawings:

- a little house, as you like, and paint it as you prefer. To form it, use geometrical shapes of your own choice (*Task 1)*.
- a figure within a frame that must be constituted from a chain of squares *(Task 2*).
- a chessboard *(Task 3)*.
- 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 *(Task 4)*.

In the context of LECGO, students were asked to 'write a program that, when executed, will display on your computer screen the following drawings:

- a drawing you can hang on the wall of your room. Use geometrical shapes of your own choice (*Task 1)*.
- a train with as many carriages as you like *(Task 2*).
- a grid (7×7) with its diagonal cells painted black *(Task 3)*.
- 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 *(Task 4)*.

4.4. The process of analyzing the data

The various types of data were organized according to the four different tasks in each learning context. Each individual student's programming solution strategies were identified and reported for each task. These strategies were analyzed in terms of: (a) type of programs constructed (using/not using algorithmic structures), and their correctness, i.e.: (i) 'completeness/incompleteness' of the programs, which means full/partial description of the drawing solutions given by the students, (ii) correct/incorrect syntax, and (iii) use/non use of appropriate coordinates), (b) student difficulties, (c) questions posed by the students, (d) graphic functions used, (e) representation systems used, and (f) time spent per task/environment. Essential interventions performed by the researcher for each task/learning context, are also reported. The use of information material provided within each environment to the students was also investigated. In the next stage, the focus was on the entire group of students and a classification of strategies into categories was constructed taking into account the (a) term mentioned above.

5. Results

Only the data emerging from the first two tasks given during the course of this formative evaluation study are presented in this paper. This was decided due to space limitations and, most importantly, due to the fact that the results from the remaining two tasks were similar to the results of the first two'. These data are organized in terms of student programming problem-solving strategies for each task (see Sections 5.1 and 5.2) performed within: (i) the p–p environment, (ii) the typical Turbo C environment, and (iii) LECGO.

Table 1 features the general programming problem-solving strategies performed by the students in these three environments. These strategies were named as S1, S2, ..., S8 and are described in Table 1 in terms of correctness of the programs formed, that means: (a) program 'completeness/incompleteness', (b) correct/incorrect syntax, and (c) correct/incorrect coordinates used.

In the aforementioned sections (Sections 5.1 and 5.2), an analysis of the realization of student strategies in each task/environment is also reported in terms of: (a) type and correctness of the written programs, (b) student difficulties, (c) student questions, (d) graphic functions used, and (e) MRS used. In addition, an analysis of the time spent per task/environment is presented (Section 5.3) and finally, the basic points considering the use of information provided by each learning context are also reported (Section 5.4).

5.1. Task1: student sequential programming problem-solving strategies within; p–p, Turbo C and LECGO

Student programming problem-solving strategies employed when facing this task in the p–p environment in Turbo C and in LECGO are presented in Table 2.

In this Table, two columns are dedicated to describe student strategies within each environment in terms of: (a) the Graphical Functions and Algorithmic Structures (GF = Graphical Function, AS = Algorithmic Structure) used by the students, (b) the strategies used (Strat. Used) as described in Tables 2 and 3, and (c) the number of students (N/S), used each specific strategy. For example, in line four of Table 2, one can

Table 1
Student general programming strategies performed within p–p Turbo C and LECGO.

Strategies	Description
S1	Forming a complete program, using correct syntax and correct coordinates. The program runs and provides appropriate results in the sense that these match students' initial drawings within Cabri.
S2	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.
S3	Forming a complete program, using correct syntax but incorrect coordinates. Thus, the program runs but does not provide the appropriate results to match students' initial drawings within Cabri.
S4	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.
S5	Forming an incomplete program, using incorrect syntax and correct coordinates. Thus, the program does not run.
S6	Forming a complete program but using incorrect syntax and incorrect coordinates. Thus, the program does not run.
S7	Forming an incomplete program, using incorrect syntax and incorrect coordinates. Thus, the program does not run.
S8	Failing to form a program, at the same time using 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.

Table 2
Student sequential programming problem-solving strategies within p–p, Turbo C and LECGO.

P–P environment			Turbo C 2.01			LECGO		
GF&AS	Strategy used	N/S	GF&AS	Strategy used	N/S	GF&AS	Strategy used	N/S
Task 1: student sequential programming problem-solving strategies								
Line	S1	2	Line	S1	2	Line	A2: D1-S3	1
Rectangle line	S5, S7	1, 1	Line rectangle	S1	2	Drawpoly line	A1: D1-NL2-I4-PC1-S1	1
Drawpoly line	S7	1	Line drawpoly	S3	1	Line circle	A1: D1-NL2-I1-PC3-S1	1
							A1: D1-NL1-I1-PC1-S1	1
							A1: D1-NL2-I1-PC1-S1	2
Rectangle circle	S6	1	Rectangle	S3	1	Rectangle circle	A1: D1-NL2-I4-PC2-S1	1
None	S8	3	Drawpoly	S1,S4	1, 1	Circle	A1: D1-NL2-I1-PC1-S1	1
			Rectangle drawpoly	S1	1	Line rectangle circle drawpoly	A1: D1-NL2-I3-PC1-S1	1

Table 3
Student solution strategies across MRS provided by LECGO.

Abbreviation used	Student solution strategies across MRS provided by LECGO
A1	Path followed: Cabri ⇒ natural language ⇒ imperative ⇒ pseudo-code ⇒ C
A2	Path followed: Cabri ⇒ C
A3	Path followed: Cabri ⇒ pseudo-code ⇒ C
D1	Using Cabri to draw a figure that is a complete solution to the given problem.
D2	Using Cabri to draw a rough figure that is an incomplete solution to the given problem.
NL1	Use of natural language for a complete and precise description of the graphical solution to the given problem.
NL2	Use of natural language for a weak description of the graphical solution to the given problem.
I1	Complete description of the graphical solution to the given problem by using the ready specific expressions in the imperative provided by LECGO. Correct coordinates are also used with the assistance of Cabri.
I2	Complete description of the graphical solution to the given problem by using the ready specific expressions in the imperative provided by LECGO. Incorrect coordinates are used (the assistance of Cabri is not taken into account).
I3	Complete description of the graphical solution to the given problem in the imperative provided by LECGO. Correct coordinates are also used with the assistance of Cabri.
I4	Complete description of the graphical solution to the given problem in the imperative provided by LECGO. Incorrect coordinates are used (the assistance of Cabri is not taken into account).
PC1	Complete description of the graphical solution to the given problem in pseudo-code. Correct coordinates are also used with the assistance of Cabri.
PC2	Complete description of the graphical solution to the given problem in pseudo-code. Incorrect coordinates are used (the assistance of Cabri is not taken into account).
PC3	Incomplete description of the graphical solution to the given problem in pseudo-code. Correct coordinates are also used with the assistance of Cabri.
PC4	Incomplete description of the graphical solution to the given problem in pseudo-code. Incorrect coordinates are used (the assistance of Cabri is not taken into account).

see that during the first task in the paper and pencil environment, the Graphical Function '*Line*' was used by two students to perform strategy S1. In addition, the abbreviations presented in the 8th column of Table 2 used to describe the problem-solving strategies constructed by the students through the use of MRS provided by *LECGO* are explained in Table 3.

The abbreviations and the type of organization of data used in Table 2 are the same as those used in Table 4 (strategies used by the students to perform the 2nd task).

5.1.1. Paper and pencil and student involvement when facing the 1st task

5.1.1.1. Type and correctness of programs written by the students. All students attempted to construct serial programs to face this task; however, only two succeeded. Specifically, one student formed a 'complete' program using correct coordinates but was found lacking in the use

Table 4
Student programming problem-solving strategies demanding the use of iteration structures within p–p, Turbo C and LECGO.

Paper and pencil environment			Turbo C 2.01			LECGO		
GF&AS	Strategy used	N/S	GF&AS	Strategy used	N/S	GF&AS	Strategy used	N/S
Task 2: student programming problem-solving strategies demanding the use of iteration structures								
For-line circle	S1	1	For-line rectangle circle line	S1	1	For-rectangle for-line	A2:D1-S1	1
Line circle	S1	1	Line	S2	2	For-rectangle	A1:D1-NL2-I1-PC1-S1	1
						For-line circle	A1:D1-NL2-I1-PC3-S1	1
Rectangle	S4	1	Line circle	S1, S2	1, 1	For-line circle	A3:D2-PC3-S1	1
							A1:D1-NL2-I1-PC1-S1	1
Rectangle circle	S7	1	Rectangle circle	S2, S3, S4	1, 1, 2	For-line-circle–rectangle	A1:D1-NL2-I2-PC3-S3	1
Rectangle circle do . . . while	S7	1				Line rectangle circle	A1:D1-NL2-I4-PC4-S3	1
							A1:D1-NL2-I1-PC1-S1	1
Rectangle for	S7	1				Drawpoly line circle	A3:D1-PC1-S3	1
Drawpoly line	S7	1						
None	S8	2						

of correct program syntax. Another student formed a complete program but used both wrong program syntax and wrong coordinates, while two students formed incomplete programs using both wrong program syntax and wrong coordinates. Finally, the remainder (three students), did not manage to form any program but merely copied a skeleton of a program in C from the learning materials given.

5.1.1.2. Student difficulties. Student difficulties in p–p concerned: (a) the structure of a program in C. Specifically, students did not know how and where to declare the variables needed (two students), nor where to put the specific statements of the program (one student). For example, they put some statements under the second bracket of the 'main' function and also failed to close the second bracket of this function (one student), (b) the syntax of graphical functions in C. Some students (three students) used fewer arguments than were appropriate, (c) the use of variables. Specifically, one student, instead of using variables, used their numerical values, while another used non defined and non initialized variables, and (d) the calculation of appropriate coordinates (three students). Some students were assisted in estimating the correct coordinates through their use of auxiliary figures.

5.1.1.3. Student questions. The questions asked by the students during their involvement in this task given in the p–p environment 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.

5.1.1.4. Graphic functions used. Students successfully used the '*line*' function to perform this task.

5.1.1.5. MRS used. Students used p–p drawing representations as well as code in C representations.

5.1.2. Turbo C and student involvement when facing the 1st task
5.1.2.1. Type and correctness of programs written by the students. Serial programs were constructed by all students participating in this experiment to confront this task, while more than half (five students) formed complete and correct programs.

5.1.2.2. Student difficulties. No syntactical errors were observed in students' programs in the context of Turbo C 2.01 in all tasks. This was due to the fact that these errors were automatically localized by the compiler and corrected by the students taking into account the 'book-like' notes provided. Here as well, all students faced difficulties in calculating the appropriate coordinates. More than half (five students) tried to solve this problem by trial and error but did not succeed. Others (four students) tried to draw auxiliary figures to estimate the coordinates needed in order to put them within the graphical functions. In fact, only three students succeeded in estimating the appropriate coordinates.

5.1.2.3. Student questions. These were the same as those in the paper and pencil environment.

5.1.2.4. Graphic functions used. The graphical functions '*rectangle*', '*line*', '*drawpoly*' and combinations of: '*rectangle-drawpoly*' and '*rectangle-line*' were successfully used by the students to perform the 1st task.

5.1.2.5. MRS used. Students used p–p drawing representations as well as code in C representations.

5.1.3. LECGO and student involvement when facing the 1st task
5.1.3.1. Type and correctness of programs written by the students. Within LECGO all students formed serial programs while the majority (eight students) formed complete and correct programs which also matched their graphical solutions performed within Cabri. It is worth noting that all students used Cabri with interest to express their drawing solutions to this task.

5.1.3.2. Student difficulties. Only one student formed a program that failed to display a figure matching the corresponding graphical solution constructed within Cabri. This was due to the fact that this student had estimated the needed coordinates by "eye" and had not exploited Cabri's ability to trace the coordinates of the figures he had drawn by using its tools. The majority (eight students) exploited this capability and incorporated the correct coordinates into their programs.

5.1.3.3. Student questions. During the performance of the 1st task within LECGO students asked questions regarding:

- How programs are executed by the computer (two students).
- The nature of a variable and especially its dynamic character (two students).
- How to use the repetition structure '*for...to*' (one student).

Students were helped to understand how programs are executed by the computer by participating in role-playing situations created by the researcher, who also provided the students with oral explanations and written examples. In fact, the researcher took on the role of a computer to 'execute' the specific commands written by the students. Before 'executing' each command, the researcher asked students to anticipate the execution performed by the computer. Next, the researcher realized the 'execution' as the computer would do and wrote down on a paper sheet the appropriate contents of the computer memory. However, students did not manage to understand the nature of both variables and the repetition structure '*for...to*', despite the fact that: (a) the researcher provided them with oral explanations and (b) they explored the written examples included in the content provided by LECGO.

5.1.3.4. Graphic functions used. All the graphical functions provided, namely; '*line*', '*rectangle*', '*circle*' and '*Drawpoly*', were successfully used in all possible combinations to perform the 1st task.

5.1.3.5. MRS used. All students expressed their interest in the tasks given and especially their pleasure in drawing through using the tools provided by Cabri. In addition, all those (eight students) who managed a complete and correct solution to this task also used all the RS provided by LECGO. Furthermore, all these students expressed their drawing solutions to the task at hand and next found a way to describe these solutions in natural language. Variations on student behaviour were observed regarding the use of both Imperative RS and pseudo-code RS. Specifically, some (four students) successfully used the ready specific expressions provided in both Imperative and pseudo-code, using correct coordinates, others (one student/two students, correspondingly) formed a complete solution to the said problem but used their own expressions in the imperative using/not using the correct coordinates, one of these translating this solution in pseudo-code without using both the specific expressions provided by the system or the needed coordinates. One other student also formed an incomplete solution in pseudo-code. Despite all these variations in student behaviour, all of them finally managed to form correct and complete programming solutions in C to the said task.

5.2. Task 2: student programming problem-solving strategies using iteration structures within: paper and pencil, Turbo C and LECGO

Student programming problem-solving strategies employed when facing the 2nd task within the three environments are presented in Table 4.

5.2.1. Paper and pencil and student involvement when facing the 2nd task

5.2.1.1. Type and correctness of the programs written by the students. Here as well, only one student correctly used a '*for-line*' structure, while two students incorrectly used the '*do...while*' and '*for-rectangle*' structures. The remainder (four students) used serial but incorrect programs. In fact, only two students formed an appropriate program to deal with this task. One student formed a complete program with correct syntax but incorrect coordinates while the rest formed totally inappropriate programs in terms of their incompleteness, incorrect syntax and incorrect coordinates used. Two students did not make any programming attempt at all.

5.2.1.2. Student difficulties. Student difficulties in facing this task concerned: (a) the syntax of the graphic functions '*rectangle*', '*circle*', '*line*', where the number of arguments was wrong (four students), (b) the syntax of the function '*drawpoly*' (two students), where the matrix of coordinates was incorrectly defined, (c) the calculation of appropriate coordinates (five students), (d) the use of repetition statements such as: (i) the '*for...to*' structure, where two students used a weak verbal description without any other specification and one student used this structure without its body and also using the wrong repetition condition and (ii) the '*do...while*' structure, where the control statement was not used (by one student) while the graphic function included inside this structure was used with constant arguments.

5.2.1.3. Student questions. These were identical to those in the first task.

5.2.1.4. Graphic functions used. Some students successfully used the statements '*for-line*' (one student), '*line*' and '*circle*' (one student) to perform this task.

5.2.1.5. MRS used. Students used p–p drawing representations as well as code in C representations.

5.2.2. Turbo C and student involvement in facing the 2nd task

5.2.2.1. Type and correctness of the programs written by the students. The majority of students formed serial programs to face this task although only one student successfully used the repetition structure '*for-line*' and only two students managed to form complete and correct programs. Some (four students) formed incomplete programs with correct syntax and coordinates while others (two students) formed incomplete programs, using correct syntax and incorrect coordinates. One student also managed to form a complete program using correct syntax but inappropriate coordinates.

5.2.2.2. Student difficulties. Here as well, students faced difficulties in estimating the appropriate coordinates and used the same procedures as in the previous task to deal with them. Specifically, six students used trial and error approaches while one student successfully used an auxiliary figure. Only two students succeeded in estimating the appropriate coordinates by trial and error. One student also confronted difficulties with the repetition structures in terms of: (a) estimation of the appropriate number of repetitions and (b) determination of the quantum of the increase of a variable used. This student overcame her difficulties by trial and error.

5.2.2.3. Student questions. These were the same as those in the first task.

5.2.2.4. Graphic functions used. To face this task the combination of '*for-line*' functions was effectively used (one student) as well as the functions '*line*', '*rectangle*' and '*circle*' (two students).

5.2.2.5. MRS used. Students used p–p drawing representations as well as code in C representations.

5.2.3. LECGO and student involvement in facing the 2nd task

5.2.3.1. Type and correctness of the programs written by the students. More than half (six students) successfully used repetition structures to face this task while the rest (three students) formed serial programs. The majority (six students), of students also formed correct and complete programs to face this task while the rest (three students) formed complete programs with correct syntax but inappropriate coordinates.

5.2.3.2. Student difficulties. During the performance of this task, students ran into difficulties related to: (a) the appropriate estimation of the coordinates needed (one student), (b) wrong initialization of the coordinates when used as variables participating in repetition structures (one student), (c) wrong formation of relationships between the coordinates used within the graphical function '*drawpoly*' when it was sequentially used (instead of using a repetition structure) within the body of a serial program (one student). These student difficulties were due to their failure both to use the appropriate features of Cabri to estimate the correct coordinates and to carefully reflect on their graphical solutions in order to find the appropriate relationships among coordinates.

5.2.3.3. Student questions. During the course of the 2nd task within LECGO, students asked questions about:

- The way programs are executed by the computer in relation to the 'behaviour' of variables (six students). Students seemed to understand these issues, by: (a) participating in a role-playing situation as described in the previous task, (b) exploring the content provided, especially the associated animations, and (c) by trial and error.
- The quantitative relationships between the coordinates of the shapes, so as to be appropriately informed of the associated variables. One student confronted this problem by grouping the graphical statements used, then estimating the variables included in these statements and next realizing the amount of increment of the quantum. Another student faced this problem by observing the variation in shape coordinates with the assistance of Cabri in combination with trial and error.
- The use of the repetition structure that is successfully encountered by studying the relevant examples and also observing the specific animations included in the content provided by LECGO (five students).
- The correct estimation of the number of repetitions within a repetition structure. Five students successfully confronted this problem by trial and error.
- The correct calculation of the coordinates needed. The majority (six students) managed to overcome this problem with the assistance of Cabri.

Finally, it is worth mentioning, that students expressed the need to receive informative feedback from the system, in the form of the correct interpretation of their drawing solutions in the previously mentioned MRS, so that they could be aware of the specific points of the translation process they failed in.

To this end, it is worth noting that, recently, the design and implementation of such a feedback system – the LECGO-Paint system – have both been realized (Kripotos and Kordaki, 2008). Specifically, LECGO-Paint is programming-oriented and has been designed in such a way as to meet the needs of learning programming in Turbo C. In the context of LECGO-Paint, learners are provided with tools that support drawing using simple geometrical objects, leading to their being actively involved in various meaningful and enjoyable drawing learning activities. Learners can interpret their drawing solutions themselves in the previously reported MRS (natural language, imperative, pseudo-code and code in the programming language C) and compare the visual outputs of their programs with their drawings formed within the LECGO-Paint system so as to implement self-correction. To this end, the LECGO-Paint system 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, ultimately, to make appropriate corrections and adjustments.

5.2.3.4. Graphic functions used. To face the 2nd task, students successfully used the combinations of '*for-rectangle*' (one student), '*for-line*' (one student), '*for-rectangle*', '*for-line*' and '*circle*' (two students), '*for-line*' and '*circle*' (two students), '*line*', '*rectangle*' and '*circle*' (one student).

5.2.3.5. MRS used. All students were interested in using Cabri to express their drawing solutions to the 2nd task. In addition, all but one student made an effort to express their drawing actions in natural language. It is essential to note that half (3 out of six students) of those who managed a complete and correct solution to this task within LECGO successfully used all the RS provided. One student from the remaining three appropriately used all these RS except pseudo-code, where he described part of the solution of the problem at hand. Another student formed a sketch of the solution within Cabri and then incompletely described his solution in pseudo-code, using appropriate coordinates, at the same time skipping the description of this solution in both natural language and the imperative. Only one student began drawing with Cabri and immediately attempted to describe their solution by writing code in C. Despite the variety of different behaviours described above, all six students finally managed to form correct and complete programming solutions to the said task in C. The rest (three students) formed correct but incomplete solutions to this task. Two of these students formed complete solutions in the imperative but without using the coordinates needed. Subsequently, they formed incomplete programs in pseudo-code using/not using (one student/one student correspondingly) appropriate coordinates, while another student expressed their drawing solution within Cabri and immediately tried to translate this in both pseudo-code and in C but failed to do so successfully.

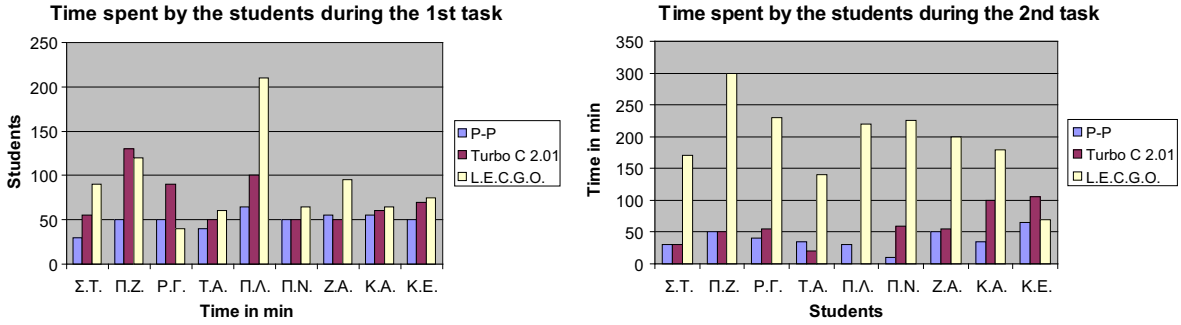

Fig. 4. Time spent by students on both tasks given within p–p, Turbo C and LECGO.

5.3. Time spent in p–p, Turbo C and LECGO

A diagrammatic presentation of time taken by each student to perform both of the given tasks in p–p, Turbo C and LECGO is demonstrated in Fig. 4.

As shown in Fig. 4, less time was spent in p–p than in both other environments as it did not provide students with aids and feedback to progress in their solutions. Specifically, in p–p, students took some time to be informed of syntactical issues from the 'book-like' content and to estimate the coordinates needed for their drawing solutions. In Turbo C, students took time to understand the compiler's messages about the syntactical errors of their programs and to estimate the appropriate coordinates by trial and error. However, the time spent by students within LECGO was more than that used in both other environments. Within LECGO, students firstly took time to familiarize themselves with both its features and Cabri so that they could form aesthetically pleasing drawing solutions for both tasks as well as trace the appropriate coordinates and their relationships when they were used as variables within iteration structures. Students also spent time navigating within the provided content to get information about programming using C as well as to be systematically engaged in both tasks across the MRS provided.

Students who successfully used iteration structures spent also a lot of time across RS. However, the time spent by the students was also related to the complexity of their drawing solutions to the given tasks. For example, the student Π.Λ. (1st task) who drew a ship in the sea and under the sun (more complex than the other students' drawings) spent double the time the other students did. Finally, it is worth mentioning that some students (excellent ones, such as the student K.E.) spent less time within LECGO (during the 2nd task) than in both other environments. This tells us that LECGO can also facilitate the more programming-experienced users to form correct programs fast and easily.

5.4. The use of learning content provided in p–p, Turbo C and LECGO

Data analysis revealed that all students visited all the types of examples integrated into the content of LECGO. All students visited the animations provided more than once in order to attempt the set tasks successfully. These animations were used to explain the solution of the specific examples by displaying the execution of the corresponding programs step by step. Students usually visited the proposed examples when they faced difficulties in understanding: (a) how a statement and an algorithmic structure works, (b) how to combine basic algorithmic structures to deal with a problem, (c) the type of iteration structures needed and the number of iterations required, and (d) the nature of the variables needed and their initialization.

Fewer students visited the pages that presented basic concepts of programming in C (3rd layer) and few students studied basic concepts of programming (4th layer). As the time of this evaluation study was limited, none of the students visited the various locations on the WWW suggested in the 5th layer of the proposed content. Students studied the proposed content in the manner described above in order to attempt the set tasks successfully.

Finally, the 'book-like' information taken in combination with the oral and written explanations of the researcher did not significantly help students to overcome their difficulties in both tasks in either of the, p–p and Turbo C environments.

6. Discussion

In this paper, the design, basic features and pilot evaluation of a computer environment for beginners' learning of computer programming using C has been presented. A number of decisions were made in terms of the design to interpret both basic aspects of constructivism and social theories of learning, as well as basic aspects of programming in C and student difficulties in the learning of programming that have emerged from the literature. As a result, a problem-solving environment was designed to facilitate students to express their problem-solving strategies in MRS. The focus when designing LECGO was on the learning of basic algorithmic structures in C, as a considerable amount of research has reported serious student difficulties in understanding the algorithmic logic in problem solving. LECGO provides students with opportunities to: (a) deal with a variety of familiar, authentic 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. 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 of graded difficulty and can also be solved without the extra cognitive load that might stem from the demand to perform complex geometrical constructions, (b) 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, (c) express their solution strategies in multiple representation systems (MRS); namely, drawing-visual representations, free-text-based representations, text-based representations using the imperative, pseudo-code based representations and representations in C. To this end, students are provided with the chance to start with intuitive 'anthropocentric' representations and non-necessary programming solutions and gradually move onto more computer-oriented programming solutions, (d) overcome the cognitive load of the syntactical rules of programming in C by using appropriately-designed computer-based authoring tools, (e) receive information about basic patterns/constructs of programming in general and of the programming language C, (f) study essential examples/models in C, and (g) receive various types of appropriate assistance to correct their mistakes. 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 tasks at hand, (ii) as ready structures and functions in pseudo-code, provided in the form of buttons and (iii) as ready structures and functions in C, also provided in the form of buttons, and (iv) as a hyperlinked, multimedia, multilayered and multi-representation system-based content emphasizing the role of holistic, real life, multiple-solution-based problem solving, based on the influence of activities from the students' own world on their learning. The design of LECGO could be modified for the learning of any programming language.

All the representation systems described in the previous section, with the exception of the last, are designed to act as 'transitional' representation systems to fill the gap between students' concrete graphic solutions and the symbolic ones written in C. These 'transitional' systems are designed to act as scaffolding elements for those students unable directly to express their solving strategies in C. Students can also be helped to understand how a computer works by being asked to explain their solutions by both giving a number of elementary directions to the computer and using the basic algorithmic structures provided. The intrinsic visual feedback can also encourage students to take control of their learning.

In comparing the design of the proposed learning environment to others reported in the literature, the following is stressed: (a) the context of drawing, which minimizes the cognitive load of the student, providing them with both a rich set of interesting, motivating problems and essential intrinsic visual feedback and (b) the existence of multiple representations in combination with multiple aids that could facilitate the student to cross the gap between the intuitive and the formal programming solution to the problems at hand.

LECGO has been evaluated in the field through a pilot study. In particular, LECGO was evaluated through a comparative qualitative study using nine 18-year old students in three learning settings where students faced similar tasks: (a) the p–p environment, where students were supported by information given in paper and pencil, organized in a traditional manner emphasizing presentation of definitions and theories before implementation of this theory in specific examples, (b) the Turbo C environment, providing the same information materials, and (c) the LECGO environment providing multiple tools and RS as well as multimedia, multi-layered animation and activity-based content.

Analysis of the quantitative data reveals that more students successfully attempted both of the given tasks in the context of LECGO (eight and six students for 1st and 2nd task correspondingly) than those who succeeded in both paper and pencil (one and two students for 1st and 2nd task correspondingly) and Turbo C (five and two students for 1st and 2nd task correspondingly) environments. In addition, no students gave up within LECGO while there were some students (three and two students, correspondingly) who gave up when facing both tasks within p–p. More students also successfully used repetition structures within LECGO (six students) than those who used such structures in p–p (one student) and Turbo C (one student). Moreover, all graphical functions were successfully used in all possible combinations within LECGO while fewer graphic functions were successfully used in p–p. In addition, students spent more time within LECGO than in the other environments. In this way, students had the opportunity to reflect on their drawing solutions and be aware of them as well as be thoughtful and careful in forming appropriate interpretations of these solutions in the MRS provided. Being involved with the interpretation of their solutions in these MRS, students were engaged more systematically with the tasks at hand. Students spent time consolidating their solutions in each RS and subsequently moved onto form their solutions in more demanding RS such as pseudo-code and code in C. Finally, students asked more complex questions when interacting within LECGO than those they asked while acting in the paper and pencil or Turbo C environments. This is probably due to the fact that – within LECGO students were freed from thinking about both syntax of the programs and coordinates needed, and therefore they could consider other essential issues in programming in C.

The variations of student behaviour in the said three environments are due to their different character. In fact, the paper and pencil environment in combination with the 'book-like' content seemed to be inappropriate for the learning of programming using C as these are inert media which do not provide the students with appropriate aids or interactive feedback about their programming problem-solving strategies; nor do they provide assistance with the program syntax. The environment of Turbo C in combination with the 'book-like' content also appeared to be an inappropriate environment for the learning of programming using C by beginners because it provides feedback exclusively about syntactical errors that is appropriate for programmers but which is apparently insufficient for beginners.

As regards student performance within LECGO, first of all, it is essential to note that all students found drawing activities using Cabri to be attractive. The drawing context of the learning activities in combination with the possibility for drawing in the context of the computer acted as a strong motivation for students to be involved in the tasks given. Students also began to build self confidence in their ability to face this type of task because the expression of their intuitive knowledge (the formation of their drawing solutions to the given tasks) was part of the demands of the tasks at hand.

In addition, by trying to interpret their drawing attempts in natural language and in the imperative, students were provided with opportunities to reflect on their drawing actions and to be aware of them. During this interpretation procedure, students felt more confident and were encouraged to progress in their solutions to the tasks at hand, as this procedure was not only part of the demands of their fulfillment but also easier than the formation of the appropriate pseudo-code and code in C. The formation of the solution of the tasks in the imperative also provided support for the students in their attempts to form pseudo-code, especially when the use of iteration structures was necessary. In fact, students had the chance to form the solution to a problem sequentially in the imperative and subsequently condense this solution into pseudo-code using both iteration statements and appropriate variables. In addition, the demand of tasks within LECGO to use pseudo-code before coding in C provided students with opportunities to connect their informal descriptions of their drawing solutions in the imperative with semi-formal solutions in pseudo-code.

Students were also helped to transform their solutions easily in pseudo-code by using the ready templates provided in the form of buttons, and to realize basic algorithmic structures and graphical functions in pseudo-code. The ready templates in C also helped students to form programs with correct syntax, at the same time incorporating basic algorithmic structures and graphical functions. As a result, most of the programs written by the students within LECGO incorporated various algorithmic structures and graphical functions without

syntactical errors. Cabri also helped students to incorporate the appropriate coordinates in their programs. Finally, it is worth noting that the majority of students who successfully managed both tasks used all the RS provided by LECGO. Contrarily, in p–p and Turbo C, few students succeeded while some gave up entirely, as there was a big gap between the formal knowledge they needed in using the RS of coding in C and both their intuitive knowledge and natural language not encouraged to be expressed in these environments.

The analysis of the data collected during this experiment also showed that all students were interested in the examples provided and visited them more than once to deal with the set tasks successfully. Students also seemed to prefer to study the information presented in the form of examples (complex and simple examples) while they seemed not to be attracted by the presentation of information in terms of definitions and descriptions (included in the 3rd and 4th layers). However, despite the fact that, all students visited the examples provided, fewer students visited the other types of information provided. Students also seemed to understand the information better when the execution of the program was presented dynamically step by step than when it was represented statically, as in the paper and pencil environment.

7. Conclusions

This paper considered the design and the pilot evaluation phase of a drawing, multi-representational and multi-aided, web-based learning environment that emphasizes the performance of drawing tasks for the learning of fundamentals in programming using C, named LECGO. Multimedia, multi-representational and multilayered, activity-based hyperlinked content was integrated into the proposed environment. A modeling methodology was used for the design of LECGO that was also inspired by modern social and constructivist theories of learning. The data analysis from the LECGO field-piloted comparative evaluation study, revealed that the learning of programming seemed to be a complex task for beginners to grasp, especially when they are left unaided or receive little assistance, as in the paper and pencil and Turbo C environments correspondingly. In a comparison of the effects of the proposed learning environment, the data analysis revealed very different results: more students successfully performed in LECGO than in both the paper and pencil and Turbo C environments, while performing similar tasks. This was due to the fact that LECGO generally provided different capabilities from those of the static paper and pencil environment and the typical compiler of Turbo C. In fact, within p–p and Turbo C, students are forced to use the commands of the programming language C directly. Contrariwise, within LECGO, students are provided with opportunities to express different types of knowledge, such as intuitive knowledge in the form of drawing solutions to the given tasks, knowledge in the form of descriptions of their solutions using natural language, and knowledge in the form of giving commands to the computer in both the imperative and pseudo-code to perform the tasks at hand. By using the previously-mentioned features of LECGO, students were helped to be: (a) pleasurably involved with the given tasks by actively forming their drawing solutions, (b) aware of their drawing solutions by trying to describe them in natural language, (c) able to transform these solutions into the form of commands to the computer in the imperative, (d) able to translate the said solutions in pseudo-code by using algorithmic structures and graphic functions in the form of buttons, and (e) able to form appropriate programs in C easily by using algorithmic structures and graphic functions in the form of buttons. Analysis of the data also shows that students found the 'book-like' information provided within both p–p and Turbo C to be boring. However, students exploited the opportunities to explore the dynamically-formed content by using holistic examples as well as specific program plans and language constructs provided by LECGO to accomplish the set tasks successfully. On the whole, this study argues that multiple aids and features in combination with attractive tasks – such as drawing tasks – are necessary for beginners' successful learning of basics in programming using C.

Finally, it should be noted that, due to certain limitations of the LECGO, pilot evaluation study it is not possible to generalize its results. Specifically, the size of the group of students participating in this pilot study was relatively small, and the number of tasks they attempted during this experiment was also limited. In addition, intermediate or experienced users may prefer to use a regular IDE such as Turbo C, since they already understand basic programming constructs. However, the positive results of this field study can be used as a basis for further investigation of the impact of LECGO on teaching and learning of programming using C by beginners in secondary level education. Our future plans include the extension of LECGO to support the learning of other programming languages, as well as its introduction into a variety of real classroom settings, also using diverse types of tasks for further investigation of student learning while interacting within this computer environment.

Acknowledgement

Many thanks to Konstadina Zikouli, not only for her ideas but also for her contribution in the collection of the data used in this paper.

References

Ainsworth, S. E., Wood, D. J., & Bibby, P. A. (1996). Analysing the costs and benefits of multi-representational learning environments. In M. van Someren, H. P. A. Boshuizen, T. de Jong, & P. Reimann (Eds.), *Learning with multiple representations* (pp. 120–134). Oxford: Elsevier Science.

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

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

Anderson, B., & Soloway, E. (1985). The role of domain experience in software design. *IEEE Transactions on Software Engineering, SE-11*(1), 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. <http://www.acm.org/education/curric_vols/k12final1022.pdf?searchterm=K-12> Retrieved 25.03.08.

Bell, T., Witten, I., & Fellows, M. (2002). Computer science unplugged. <http://www.unplugged.canterbury.az.nz>.

Ben-Ari, M. (2004). Situated learning in computer science education. *Computer Science Education, 14*(2), 85–100.

Berglund, A., Daniels, M., & 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.

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.

Burton, J. M., Horowitz, R., & Abeles, H. (2000). Learning in and through the arts: The question of transfer. *Studies in Art Education, 41*(3), 228–257.

Calloni, B., & Bagert, D. (1994). *Iconic programming in BACCII vs. textual programming: Which is a better learning environment?* (pp. 188–192). Phoenix, AZ: ACM (SIGCSE '94 3/94).

Calloni, B., & Bagert, D. (1997). *Iconic programming proves effective for teaching the first year programming sequence* (pp. 262–266). CA, USA: ACM (SIGCSE '97).
Christiaen, H. (1998). Novice programming errors: Misconceptions or mispresentations? *ACM SIGCSE Bulletin, 20*(3), 5–7.
Cohen, L., & Manion, L. (1989). *Research methods in education*. London: Routledge.
Cohen, V. B. (1985). A reexamination of feedback in computer-based instruction: Implications for instructional design. *Educational Technology*, 33–57.
Crooks, C. (1994). *Computers and the collaborative experience of learning*. London: Routledge.
Csikszentmihalyi, M. (1997). Assessing aesthetic education. *Grantmakers in the Arts, 8*(1), 22–26.
CTGV (1992). Technology and the design of generative learning environments. In D. H. Jonassen & T. M. Duffy (Eds.). *Constructivism and the technology of instruction* (pp. 77–89). New Jersey: Laurence Erlbaum.
Dagdilelis, V., Satratzemi, M., & Evangelidis, G. (2004). Introducing secondary education to algorithms and programming. *Education and Information Technologies, 9*(2), 159–173.
Davidovitch, L., Parush, A., & Shtub, A. (2006). Simulation-based learning in engineering education: Performance and transfer in learning project management. *Journal of Engineering Education, 95*(4), 289–300.
Deasy, R. J. (2002). *Critical links: Learning in the arts and student academic and social development*. Washington, DC: Arts Education Partnership. <http://www.aep-arts.org>.
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.
Eisner, E. W. (2002). The state of the arts and the improvement of education. *Art Education Journal, 1*(1), 2–6.
Eckerdal, A. (2009). *Novice programming students' learning of concepts and practise*. Dissertation presented at mathematics and computer science, department of information technology, Upsalla University, Sweeden, March 6, 2009. <http://uu.diva-portal.org/smash/record.jsf?pid=diva2:173221>.
Ellis, A. (1998). Development and use of multimedia and internet resources for a problem based environment. In *Proceedings of the 3rd conference on integrating technology into computer science education and on 6th annual conference on the teaching of computing* (pp. 269). Ireland.
Freund, S. N., & Roberts, E. S. (1996). *THETIS: An ANSI C programming environment designed for introductory use*. Philadelphia, PA, USA: ACM. pp. 300–304 (SIGCSE '96 2/96).
Gardner, H. (1983). *Frames of mind: The theory of multiple intelligences*. New York: Basic Books.
Garner, S., Haden, P., & Robins, A. (2005). My program is correct but it doesn't run: A preliminary investigation of novice programmers' problems. In *Proceedings of Australasian computing education conference 2005* (pp. 173–180). Newcastle, Australia.
Garner, S. (2007). A program design tool to help novices learn programming. In *ICT: Providing choices for learners and learning*. In *Proceedings ascilite Singapore 2007*. <http://www.ascilite.org.au/singapore07/pros/garner.pdf> Retrieved 25.03.08.
Greene, M. (1995). *Releasing the imagination: Essays on education, the arts, and social change*. New York: Teachers College Press.
Guzdial, M. (2003). *Programming environments for novices*. <http://coweb.cc.gatech.edu/mediaComp-plan/uploads/37/novice-envs2.pdf> Retrieved 14.05.09.
Guzdial, M., & Ericson, B. (2005). *Introduction to computing and programming in Java: A multimedia approach*. New Jersey: Prentice Hall.
Janvier, C. (1987). Representation and understanding: The notion of function as an example. In C. Janvier (Ed.), *Problems of representation in teaching and learning of mathematics* (pp. 67–72). London: Lawrence Erlbaum.
Jonassen, D. H. (1996). *Computers in the classroom: Mind tools for critical thinking*. Columbus, OH: Merrill/Prentice-Hall.
Jonassen, D. H. (1999). Designing constructivist learning environments. *Instructional Design Theories and Models, 2*, 215–239.
Jonassen, D. H., Strobel, J., & Gottdenker, J. (2005). Model building for conceptual change. *Interactive Learning Environments, 13*(1–2), 15–37.
Hadjerrouit, S. (1998). A constructivist framework for integrating the Java paradigm into the undergraduate curriculum. In: *Proceedings of the 3rd on integrating technology into computer science education and on 6th annual conference on the teaching of computing, Ireland* (pp. 105–107).
Hadjerrouit, S. (2008). Towards a blended learning model for teaching and learning computer programming: A case study. *Informatics in Education, 7*(2), 181–210.
Hu, G. (2006). It's mathematical, after all-the nature of learning computer programming. *Education and Information Technologies, 11*, 83–92.
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), 1–16.
Kaput, J. J. (1994). The representational roles of technology in connecting mathematics with authentic experience. In R. Biehler, R. W. Scholz, R. Strasser, & B. Winkelman (Eds.), *Didactics of mathematics as a scientific discipline: The state of the art* (pp. 379–397). Kluwer Academic Publishers.
Kolikant, Y. B.-D., & Pollackm, S. (2004). Establishing computer science norms among high school students. *Computer Science Education, 14*(1), 21–35.
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., & Balomenou, A. (2006). Challenging students to view the concept of area in triangles in a broader context: Exploiting the tools of Cabri II. *International Journal of Computers for Mathematical Learning, 11*(1), 99–135.
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, 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., & Kapsampelis, G. (2008). A computer environment for the learning of sorting algorithms: Design and pilot evaluation. *Computers and Education, 51*(2), 708–723.
Laborde, J.-M. (1990). *Cabri-geometry [software]*. France: Universite de Grenoble.
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* (pp. 14–18). Portugal: ACM Press.
Lahtinen, E., & Ahoniemi, T. (2009). Kick-start activation to novice programming – A visualization-based approach. *Electronic Notes in Theoretical Computer Science, 224*, 125–132.
Lemone, K. A., & Ching, W. (1996). Easing into C: Experiences with RoBOTL. *ACM SIGCSE Bulletin, 28*(4), 45–49.
Maureen, T. (2000). Constructivism, instructional design, and technology: Implications for transforming distance learning. *Educational Technology and Society, 3*(2), 50–60.
Mellar, H., & Bliss, J. (1994). Introduction: Modeling and education. In H. Mellar, J. Bliss, R. Booham, J. Ogborn, & C. Tompsett (Eds.), *Learning with artificial words: Computer based modeling in the curriculum* (pp. 1–8). London: The Falmer Press.
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.
Naps, T., Rossling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., et al. (2003). Exploring the role of visualization and engagement in computer science education. *SIGCSE Bulletin, 35*, 131–152.
National Research Council Committee on Information Technology Literacy, (1999). *Being fluent with information technology*. Washington, DC: National Academy Press (May 1999). <http://www.nap.edu/catalog/6482.html> .
Norman, D. A. (1993). *Things that make us smart: Defending human attributes in the age of the machine*. Reading, MA: Addison-Wesley.
Noss, R., & Hoyles, C. (1996). *Windows on mathematical meanings: Learning cultures and computers*. Dordrecht: Kluwer Academic Publishers.
Pacheco, A., Gomes, A., Henriques, J., de Almeida, A.-M., & Mendes, A.-J. (2008). Mathematics and programming: Some studies. In *International Conference on Computer Systems and Technologies – CompSysTech'08*, V15-1, V15-6.
Papert, S. (1980). *Mindstorms: Children computers and powerful ideas*. New York: Basic Books.
Pattis, R. E., Roberts, J., & Stehlic, M. (1995). *Karel the robot a gentle introduction to the art of programming* (2nd ed.). New York: Wiley.
Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137–172.
Samurcay, R. (1989). The concept of variable in programming: Its meaning and use in problem solving by novice programmers. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 161–178). Hillside, NJ: Erlbaum.
Sangwan, R. S., Korsh, J. F., & LaFollete, P. S. (1998). *A system for programming visualization in the classroom*. Atlanta, GA, USA: ACM. pp. 272–276 (SIGCSE '98).
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', Phodes, Greece* (pp. 289–298).
Soloway, E., & Spohrer, J. C. (1989). *Studying the novice programmer*. Hillside, NJ: Erlbaum.
Teague, D. (2009). A people-first approach to programming. In *Eleventh Australasian computing education conference (ACE2009), Wellington, New Zealand, January 2009*.
Vygotsky, L. S. (1978). *Mind and society: The development of higher mental processes*. Cambridge, MA: Harvard University Press.
Wallingford, E. (2001). *The elementary patterns home page*. <http://www.cs.uni.edu/~wallingf/patterns/elementary/> Retrieved 25/03/08.
Winslow, L. E. (1996). Programming pedagogy. *SIGCSE Bulletin, 28*(3), 17–22.
Wulf, T. (2005). Constructivist approaches for teaching computer programming. In *SIGITE'05, October 20–22, Newark, New Jersey, USA* (pp. 245–248).

Zikouli, K., Kordaki, M., & Houstis, E. (2003). A multi-representational environment for learning programming and C. In *3rd IEEE international conference on advanced learning technologies, July 9–11, Athens, Greece* (pp. 459).

Maria Kordaki is an advisor for secondary education mathematics teachers and adjunct assistant professor in the Department of Computer Engineering and Informatics, Patras University, Greece, where she trains students in Educational Technology and in Didactics of Computer Science. Her research interests include Educational Technology and Didactics of both Mathematics and Computer Science. Her correspondence address is: Maria Kordaki, Department of Computer Engineering and Informatics, Patras University, 26500, Rion Patras, Greece. Tel.: +2610 996932; e-mail: kordaki@cti.gr.