Tools to support initial programming learning

António José Mendes and Maria José Marcelino

Abstract: *Basic programming learning is known as complex for many novice students at university level. We have been working for several years in the development of computer based tools that may help students in programming learning. From the utilization of those tools we concluded that while they proved useful for many students, for those with deeper difficulties they were not enough to promote learning to an acceptable level. Our current work tries to address this problem through the development of new tools and some studies that may lead to new directions in our future work. In this paper we describe our previous and current work in this field.*

Key words: *Programming learning, animation based simulation, problem solving.*

INTRODUCTION

Basic programming learning is complex for many novice students at university level. The most important problem for many is their low ability to develop an algorithm that can solve a particular problem. The application of basic concepts or the design of simple algorithms can be difficult obstacles. These difficulties are felt independently of the programming language or paradigm used. Some authors identified reasons for these difficulties [1], such as:

- The need for good competences on problem solving;
- Students are used to courses that depend mostly on theoretical knowledge and memorization, but basic programming learning needs a more practical approach, based mostly on problem solving activities;
- Traditional teaching, often based on lectures and specific programming language syntaxes, often fails to motivate students to get involved in meaningful programming activities;
- Programs have a dynamic nature, but most learning materials have a static format which makes difficult to analyse program's dynamic behaviour;
- Students' learning needs are often very different within the same course. Different learning styles and previous experience difficult a common approach to the whole group. Group sizes often difficult individualized support to students.

During the years researchers have been trying to find answers to the most commonly referred difficulties. A common approach has been the development and utilization of animation based simulation tools that try to take advantage of the potential of human visual system. Those systems are rooted in the conviction that programs can be better understood when represented graphically if compared with textual descriptions and representations. For example, BlueJ [2] is an integrated system including an object-oriented language and an object-oriented development environment. It uses UML- like class diagrams to present a graphical overview of a project structure. It allows the interactive creation of objects from any given class present in the project.

The high abstraction level of computer programs has also been mentioned as a source of difficulties. The use of micro worlds populated by representations of concrete entities (robots and turtles for example) has been proposed as a way to lower the abstraction level. The student can control the entities' movements and behaviour through programming instructions, developing his/her programming competences in a more familiar environment. Karel++ is an example of this type of environments [3].

Many other tools have been proposed, most of them having a limited scope, since they address only a particular type of algorithms and programs. They serve mainly demonstration purposes, as they can only animate a set of pre-defined programs, but not student's programs.

However, some studies have shown that the learning results obtained with the utilization of visualization tools are not as good as expected [4]. Three approaches have been proposed to make visualizations more helpful, engaging visualization, explanatory visualization and adaptive visualization [5]. Engaging visualization stresses the importance of student involvement in learning. This means that students must have an active role, instead of just seeing teacher prepared animations. Explanatory visualization proponents argue that many times students fail to understand what they are seeing. They defend that visual representations should be augmented with natural language explanations that can help student understanding. Adaptive visualization consists on adapting the level of detail of visual representations to the difficulties the underlying concepts pose to each student.

Our own work has followed the two first mentioned approaches. The tools we developed support engaging activities (they animate student developed programs) and have some explanatory features. Together with colleagues from Castilla-La Mancha University in Spain we have also been involved in the development of collaborative tools to support programming learning [6].

Currently we develop a new project, ProLearn. In this context we are developing some studies. The first tries to access the impact that mathematical and logical problem solving competences can have in programming learning. The second studies the connections between student's learning styles, the types of materials and activities used to promote learning with the student ability to learn programming. Both studies are still under development and final results will only be available after the end of current school year. ProLearn includes also the development of new tools and the improvement of previous ones. In the next sections we will describe briefly our group previous work and outline our current project (ProLearn).

ANIMATION BASED SIMULATION TOOLS

Animation based simulation has been proposed to reduce student's difficulties. It can make program's dynamics concrete and visual and support practical work at the student own learning rhythm. Animated views can help many students in three central learning activities: Understand programs; Evaluate existing programs; Develop new programs [7]. This last activity is the most important for learning, but also the most difficult for many students, as they often fail when asked to develop a new program, even if it is similar to one they just studied.

We believe learning is more effective when students assume an active role. So, it is important that the students can see how their solutions work and compare with how they thought they would work. This process should lead to error detection, correction and, hence, learning. These activities are very important in programming learning, since students can reach a higher competence and confidence level after being able to have programs running correctly. This is very important, since after a first wrong attempt many students just give up or try to find a teacher or a colleague that shows them a solution.

Our team has been involved in the development of algorithm and program simulation tools for several years. In particular we developed three tools, SICAS, OOP-Anim and PESEN that share a common pedagogical foundation. They support a constructivist approach to learning, as each student assumes an active role, learning at his/her own pace and progressively constructing his/her own knowledge. Students are allowed to develop solutions to proposed problems, to simulate them through animation and to see if they work properly. In case there are errors students can correct their solutions and simulate them again. This process should continue until a correct solution is achieved.

SICAS

SICAS environment [8] was designed to support learning of basic procedural programming concepts, such as selection and repetition. It is language independent and

oriented to the design and implementation of algorithms. Using SICAS students are encouraged to develop their capacities through problem solving.

In SICAS, algorithm design is supported by an iconic environment where the student builds a flowchart that represents her/his solution. Algorithms developed with SICAS can include attributions, input/output, repetition and selection instructions. They can use numeric and string variables. Functions can also be defined and used. These elements can be introduced in a flowchart by clicking (in the toolbar icons) and pointing (in the design area). When that happens, a dialog box automatically opens asking the student to specify the element details (e.g. the condition in a selection). This option helps students to avoid common novice programmer syntax errors.

Any algorithm created with SICAS is automatically translated to pseudo-code, C and JAVA code. These alternatives show that a well designed algorithm can be easily translated into several programming languages and that the most important factor in algorithm design is its conception, not the programming language in which it will be coded.

OOP-ANIM

Object oriented programming (OOP) paradigm has grown significantly in the last years. Consequently many universities have adopted this paradigm in their basic programming courses. The development and popularization of OOP languages, like Java, has also contributed to this development. Although knowledge about basic concepts, such as selection and repetition, is still fundamental, it is also important to have tools that can support students when learning the basic concepts of the OOP paradigm.

OOP-Anim is a tool that can animate and simulate small Java programs [9]. Its objectives and pedagogical approach are the same presented for SICAS, but in this case the main focus are basic OOP concepts, like class, object or inheritance.

PESEN

The utilization of the above tools showed us that although they are useful to many students [10], for those with bigger difficulties, they are not enough. When asked to create an algorithm to solve a common programming problem some students do not know what to do, that is to say what instructions to choose (even if from a small set of available ones, as it is the case in SICAS) and how they should be organized. Usually students understand the problem and sometimes they can make some kind of high level description of a possible solution. However, they have difficulties creating a solution that can be expressed in instructions; no matter they use code, pseudo-code or flowcharts. The language seems irrelevant accordingly to students' own comments and observations. Typical examples are situations where they grasp that there must be a repetition in the solution, but they do not know how to articulate it with other instructions or simply how to control it.

PESEN (Programming Environment Simple Enough for Novices) main concept is to ask the student to program the movements of elementary geometrical shapes through simple commands [11]. The idea is to create exercises in a very simple environment, so that students have no doubts about what should be done and the necessary instructions. As the solutions involve decisions and repetitive tasks, we believe PESEN can help weaker students to learn the basic concepts of sequential, conditional and repetition execution usually present in computer programs. The main objective is to support students to improve their programming level, so that they can start using SICAS and, later, a programming language.

PROLEARN PROJECT

As mentioned before, our previous work consisted essentially in the development of animation based simulation tools. However, our experience shows that although these

tools are useful in many situations, in other cases they need to be complemented with other features and tools that can give a more complete support to students, especially those with deeper learning difficulties. Our current project includes our previous tools, but tries to address other aspects through the development of a new tool (ProGuide) and some studies that may help us to find better solutions. This ongoing work is described briefly in the next sections.

STUDENT GUIDANCE WITH ProGuide

One of the main problems with animation based simulation tools is that they depend on the capacity the student has to create a first solution to a given problem. Only after a first solution exists, even if wrong, students can use the tool to simulate and correct it. However, in early learning stages many students find difficult even to create a first solution attempt.

We are trying to address this problem through the development of a new tool, ProGuide, that works together with SICAS, interacting with students during algorithm development, guiding them when necessary. It is a dialogue-based tool that helps novice students to solve problems using text based communication. When students are creating an algorithm, ProGuide monitors their actions (or lack of action) and interacts with them, providing some guidance whenever necessary (Figure 1).

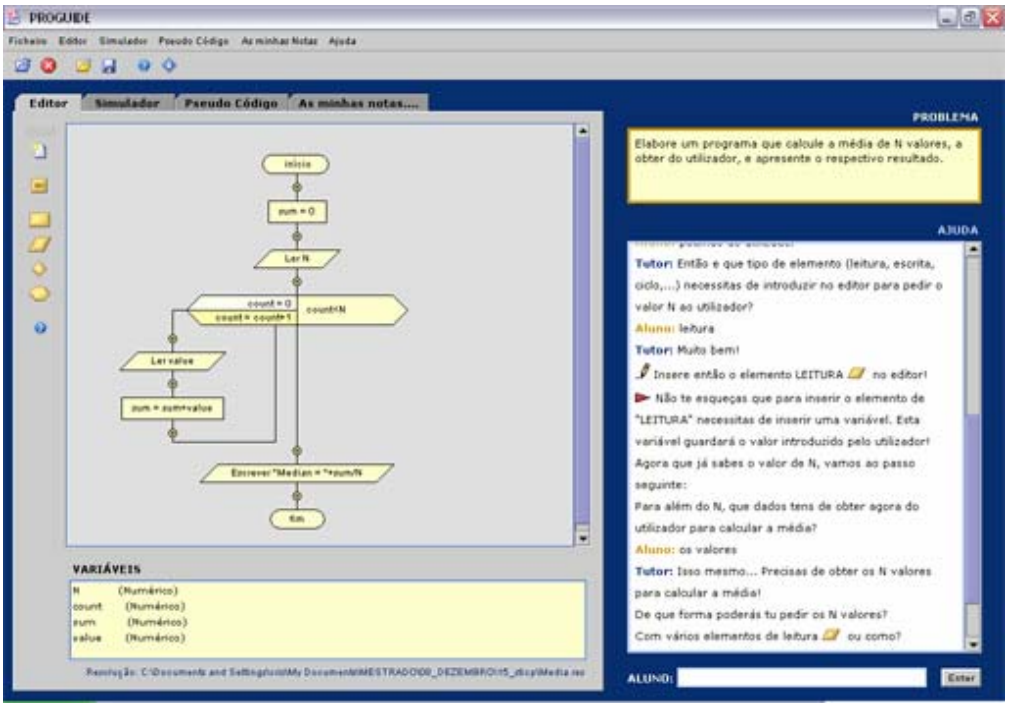

Figure 1 – ProGuide interface

ProGuide has three main modules. The first manages natural language knowledge to be used in dialogues with students. The second module has information about partial steps and strategies to develop a particular algorithm. The third module follows student's actions, so that the tool can decide when to initiate a dialogue and the type of interaction that is more adequate depending on the resolution stage. All these modules work together to create a useful dialogue with students, trying to encourage them and providing hints so that they can reach a good solution to their particular task.

Although we have plans to include a specific tool to support teachers if they want to include new problems in ProGuide, currently the only possibility is to specify the problem and associated information in XML. When a new problem is created the author must indicate its description, but also a solution plan, its goals and a dialogue plan. This

information is then used by ProGuide during interaction with the student. However, taking into consideration that the tool main objective is to support students with deeper difficulties in their initial learning, we can argue that we don't need a large number of problems to make the tool effective. To use the tool in other learning contexts where the number and variety of problems is important, it will be essential to have a tool to support problem specification.

RELATED STUDIES

Our group previous work has been based on the development and utilization of computer based tools. However, as teachers it is important to know the reasons to some student's difficulties, even if participating in the same classes and activities with students that don't show any significant difficulties. Two main questions have been raised as possible explanations: the level of previous mathematical and logical problem solving abilities and students different learning styles. As part of our work in this field we are developing two studies trying to get more information that can be used in our classes and in our tools to better help our students.

The first study tries to establish a link between mathematical and logical problem solving abilities with programming learning. This study involves students that in this year first semester failed to be approved in the first programming course. It can be assumed that they have significant difficulties in programming learning. During this second semester part of those students follows a free course on mathematical and logical problem solving where no programming is used. The other part is only following the normal second semester courses. After the classes finish all students will be submitted to a programming test and we will use the results to establish if the free course improved programming competences.

The second study is connected with the previous one and uses the same group of students. Its objective is to verify if we can establish a link between student learning styles and programming learning difficulties. In the beginning all students were tested and their predominant learning style was determined using a commonly used test. Students are now following the free course mentioned above and each two weeks they are submitted to a mathematical problem solving test. Those tests are evaluated and for each student the most common errors are determined. We will then see if it is possible to establish a connection between learning styles and common errors. If this is possible we can then look for answers that explain why students with a particular learning style have some specific difficulty while others do not show it.

The conclusions of these studies may lead to a change in our approach, both in what concerns class methodologies and computer based tool support. We believe that our animation based tools will continue to be useful in many cases, but for some students they are not enough, which means that new tools and approaches must be identified and implemented.

CONCLUSIONS AND FUTURE WORK

In this paper we briefly described some of our past work in the field of computer based programming learning support. The utilization of those tools has shown they are useful for many students, but for some they are not enough. Our current work tries to address those students with more learning difficulties. We described the main options and objectives of a new tool currently under development and mentioned the reasoning behind two studies currently under development. This work will continue in the near future and will hopefully lead to the development of a powerful learning environment that will integrate existing tools, but also new ones that may result from our current project.

REFERENCES

[1] Jenkins, T., On the Difficulty of Learning to Program. In Proceedings of 3rd Annual LTSN-ICS Conference, pp 53-58, Loughborough University, UK, 2002.

[2] Kölling, M., Quig, B., Patterson, A. & Rosenberg, J., The BlueJ system and its pedagogy, Journal of Computer Science Education, 13(4), Dec 2003.

[3] Bergin, J., Stehlik, M., Roberts, J. & Pattis, R., Karel ++: A Gentle Introduction to the art of Object-Oriented Programming. John Wiley & Sons, 1997.

[4] Byrne, M., Catambarone, R & Stasko, J., Evaluating Animations as Student Aids in Learning Computer Algorithms. Computers & Education, 33(5), 1999.

[5] Brusilovsy, P. & Spring, M., Adaptive, Engaging and Explanatory Visualization in a C Programming Course . In Proceedings of ED-MEDIA - World Conference on Educational Multimedia, Hypermedia & Telecommunications, Lugano, June 2004.

[6] Mendes, A. J., Esteves, M., Gomes, A., Marcelino, M., Bravo, C. and Redondo, M., Using simulation and collaboration in CS1 and CS2. In Proceedings of The Tenth Annual Conference on Innovation and Technology in Computer Science Education – ITICSE05", Lisbon, June, 2005.

[7] Stasko, J., Tango: A Framework and System for Algorithm Animation. IEEE Computer, 23(9), pp 27-39, 1990.

[8] Gomes, A. and Mendes, A., SICAS: Interactive system for algorithm development and simulation. In Ortega, M. and Bravo, J. (eds.), Computers and Education in an Interconnected Society, Kluwer Academic Publishers, 2001, pp. 159-166.

[9] Esteves, M. and Mendes, A., A Simulation Tool to Help Learning of Object Oriented Programming Basics. In Proceedings of the 34th Frontiers in Education Conference, Savannah, USA, October 2004.

[10] Rebelo, B., Marcelino, M. and Mendes, A., Evaluation of a System to Support Algorithm Teaching and Learning. In Proceedings of CATE2005 - Computers and Advanced Technology in Education Conference, Aruba, August, 2005.

[11] Mendes, A., Jordanova, N. and Marcelino, M., PESEN - A Visual Programming Environment to Support Initial Programming Learning. In Proceedings of CompSysTech05 – International Conference on Computer Systems and Technologies, Varna, Bulgaria, June 2005.

ABOUT THE AUTHORS

Assist. Prof. António Mendes, PhD, Department of Informatics Engineering, University of Coimbra, Phone: +351 239 790000, E-mail: toze@dei.uc.pt.

Assist. Prof. Maria Marcelino, PhD, Department of Informatics Engineering, University of Coimbra, Phone: +351 239 790000, E-mail: zemar@dei.uc.pt.