

ProGuide: A Dialogue-Based Tool To Support Initial Programming Learning

Cristiana Areias
Polytechnic Institute of Coimbra
Coimbra Superior Institute of Engineering
Dep. Informatics and Systems Engineering
3030-199 Coimbra - Portugal
+351 239 790 350
cris@isec.pt

António Mendes
University of Coimbra
Dep. Informatics Engineering
3030-290 Coimbra - Portugal
+351 239 790000
toze@dei.uc.pt

ABSTRACT

In this paper, we describe a new tool, ProGuide, designed to support some students in their initial programming learning. It is particularly suited to students that show deeper difficulties to create algorithms and programs, and need a strong and more detailed support to overcome their limitations.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education - *Computer science education*.

General Terms

Algorithms, Languages.

Keywords

Algorithmic learning, problem-solving techniques, programming learning.

1. INTRODUCTION

Computer programming learning is difficult and requires hard work from students. They need to develop several competences to be able to program correctly. Novice students must get acquainted with the syntax and semantics of programming constructs and, more difficult, use that constructs to create programs. This implies the development of competences in several areas like abstraction, generalization, transfer and critical thinking, among others [1]. This is difficult for many students and for a teacher it is common to face situations where a student can't even start a solution to a simple programming problem, even though they usually know and understand basic programming constructs. Like [2] we believe that the real problem is "putting all pieces together" composing and coordinating components of a program.

Animation based simulation has been proposed as a way to reduce student's difficulties. It can make concrete and visual program's dynamics and support practical work at the student own learning rhythm. It can be argued that animated views can help students in three central learning activities: Understand programs; Evaluate existing programs; Develop new programs [3]. This last activity is the most important and also the most difficult. Many students can understand programs previously developed by the teacher or other students, but they fail when they have to develop a program themselves to solve some problem, even if it is similar to the one they understood.

Three approaches have been proposed to make visualizations more helpful, engaging visualization, explanatory visualization and adaptive visualization [4]. 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.

We have been working mostly in engaging visualization, as we also 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 central 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. To support this type of activities we developed tools like SICAS [1] and OOP-Anim [5].

In our teaching experience we have found many novice programming students that after some initial teacher support can progress in their learning and develop autonomy in their work. However, others show deeper difficulties and need constant support to solve basic problems. This is almost impossible due to the large number of students in our courses and the limited number of available teaching staff. In that context, we developed and present in this paper a new environment, called ProGuide, which tries to support weaker novice students to acquire basic programming competences.

2. ProGuide ENVIRONMENT

ProGuide main objective is to help reduce the difficulty many students show to propose solutions, even incomplete or wrong, to basic programming problems. It has structures to store information about problems and uses that information to interact with a student when a particular problem is proposed to him/her. This interaction tries to help students to develop a correct solution to the problem.

In ProGuide students express their solutions on a flowchart based tool directly inspired in SICAS. It presents features that allow the student to create algorithms and simulate them to see if they work

as expected. It is also possible to automatically generate Java, C and pseudo code correspondent to the flowchart created by the student. The main problem with this approach is that weaker students don't take advantage of it, since they can't develop a first solution that can be simulated, corrected and improved. ProGuide new features try to support students during algorithm development, so that they can take advantage of the simulation tool.

ProGuide has a small collection of basic exercises to propose to students. Although we have plans to later include a specific tool to support teachers if they want to include new problems, currently the only possibility for the teacher is to specify the problem and associated information in XML. When a new problem is created its author must indicate its description, but also a solution plan, its goals and a dialogue plan. The solution plan includes the steps the student should follow to solve the problem (it is possible to have alternative plans). This information is necessary to validate student's actions. Goals are partial objectives that are included in the solution plan. They can be declared autonomously so that they can be reused in several problems. Finally, the dialogue plan includes the steps the dialogue should follow during the interaction with the students.

The architecture of ProGuide is presented in Figure 1. The environment has three main modules. The first manages natural language knowledge to be used in dialogues with students. The second module has information provided by the teacher about partial steps and strategies to develop a particular algorithm and the dialogue plan. The third module follows student's actions, so that the tool can decide when to initiate a dialogue with the student and the type of interaction that is more adequate depending on the resolution stage. All modules work together to create a useful dialogue with students, trying to encourage them and providing hints so that they can reach a correct solution to the problem.

In the figure the gray background represents ProGuide interface and the other components ProGuide engines. The arrows represent the data flow between the environment components.

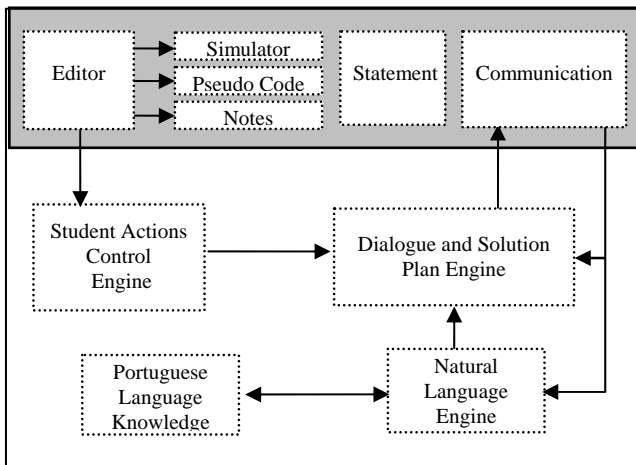


Figure 1 - ProGuide Architecture

ProGuide interface has three main areas, as shown in figure 2. When ProGuide is called the student must select a problem to solve. It is presented in the top right area. The tutoring (or communication) area (bottom right) starts presenting a global

question to establish the dialogue with the student. The interface left side is the solution area. Here the student can create, simulate and improve solutions to the problem. As mentioned before this is done using a flowchart approach.

Simplicity was a goal to us. It is important that students do not spend significant time struggling with the environment, but instead give attention and time to their programming learning tasks.

2.1 Editor Area

The editor area is based on SICAS, a previously developed environment that has functionalities that we consider useful in an early learning stage:

Algorithm design is supported by an iconic environment, where the student can build flowcharts to represent them. This option was based on studies that recommend this form of representation, as it is more appellative, facilitates understanding, and is simpler and probably less prone to errors than pseudo-code [6]. Also, research has shown that a graphical [7] or iconic interfaces [8] can be more suitable for program construction than textual interfaces.

Expressions use syntax similar to C and JAVA, because the environment potential users will probably program, at a later stage, is one of these languages. However, syntactic details are minimized, so that the students can concentrate completely in algorithm development.

As the environment objective is to support initial learning, the instructions that can be used in algorithm development are limited:

- *Input/Output* elements to read, write the value of a variable or expression
- *Repetition* element, creates a loop that repeats the execution of some action
- *Selection* element to choose between two sets of actions that may be executed
- *Variables* element to create, modify or delete variables.

Any of these components can be specified through the use of simple dialogue boxes where student must insert each instruction details (for example the condition in a selection instruction). This option minimizes the necessary syntactic knowledge, allowing students to concentrate in algorithm design.

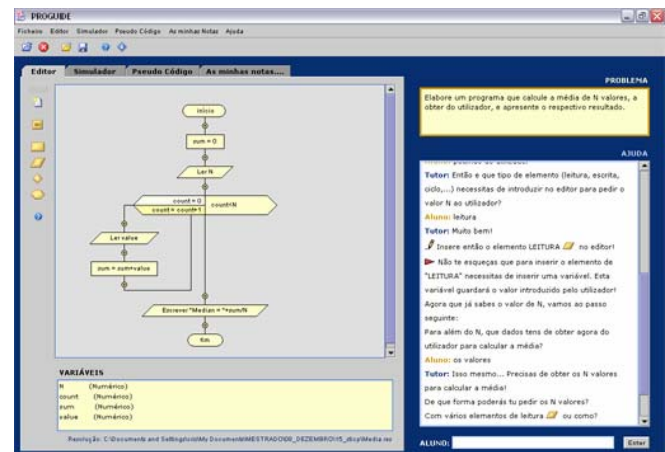


Figure 2 - ProGuide Main Interface

Students can simulate their solution through flowchart animation. They can see if it works as expected and, if necessary correct and improve it. Animation can be done in step-by-step mode or quickly, and can be stopped or paused whenever the student wants. During the simulation students can observe the algorithm output and the variable values.

2.2 Tutoring/Communication Area

Many educators would agree that the most effective form of teaching is through one-on-one interactions with students [9]. So, it is not surprising that an effective way to teach programming is to give students support in their reasoning and immediate feedback on the programs they create. That is also ProGuide main objective. We decided to use a subset of natural language to establish communication with the student, since we believe that other possible options would difficult communication at this early learning stage.

Natural language processing is complex and difficult. However, to reach our objectives we need only a small subset of the Portuguese language. There is no need to support very complicated statements.

ProGuide natural language engine was inspired by the A.L.I.C.E. project [10]. It uses AIML (Artificial Intelligence Markup Language) language, a XML-compliant. AIML supports the description of classes of objects and partly of program behavior. The objects are organized in topics and categories with relevant information. A category is not more than a standard answer.

Natural language ProGuide engine starts filtering the input and converting it to a generic format. To do this it transforms the phrase, deleting some pronouns and finding synonymous to put it in a generic form that can be used in the next steps. Then, like A.L.I.C.E, we use XML files to validate the knowledge and categorize it for the dialogue. The knowledge base includes a lot of category elements. Figure 3 shows one of them:

```
<element category="LER_UTILIZADOR">
  <standard>OBTER UTILIZADOR _* </standard>
  <answer>
    <selement>LER<item/> </selement>
  </answer>
```

Figure 3 - ProGuide category element

Each *element* has a *standard* and an *answer* element. The standard is the phrase that can have some special elements, like *_** that represents the rest of the phrase. The answer can be null or can be another phrase that the engine must select recursively. In this example, it will select the element LER and the item refers to the rest of the phrase, that is, the same like *_**.

The category is then sent to the dialogue and to the solution plan engine that guides the next dialogue step. There is an especial category, designated as *_DATA* that informs the dialogue and solution plan engine that the next data doesn't need to go to the natural language engine. For example, this happens to get variable names.

When ProGuide can't understand a student input, it answers: "I'm sorry but I did not understand what you said." or "Can you repeat with others words?". When the dialogue and solution plan engine waits for some category and the student answers differently the reaction is to say "We are not in the same context" or similar. Some dialogue steps have associated timeouts, that may trigger one of the following ProGuide possible reactions:

- Repeat the question;
- Say one of the following expressions "Are you there?" or "If you don't know you can ask for help" or "You don't answer?", and so on;
- Or, present the answer and go to the next step.

2.3 Example

In this section we will present a ProGuide utilization example. We will use the problem "Compute and display the average of N values". The problem has an associated plan that should be followed. It includes five goals that the student must achieve:

- Understand how to compute an average;
- how to get the values
- how to do a loop
- how to do a sum
- how to print a value

Each goal can be simple or have its own internal plan. All goals must be reached during the dialogue and in parallel the algorithm solution must be constructed.

In the beginning of a session, a general question establishes the interaction between ProGuide and the student. Figure 4 presents a portion of a possible dialogue between ProGuide and a student. It was translated to English, since ProGuide currently works only in Portuguese.


```
Tutor: Do you know how to compute an average?
Student: Yes
Tutor: Ok! So, let's go to solve that problem.
Which data do you need to receive from the user to solve that problem?
Student: The values
Tutor: Very good! You need the values to compute the average.
How many?
Student: 5
Tutor: No, no... I suggest that you read the problem again.
Student: N
Tutor: And how can you do that?
Student: Ask to the user
Tutor: Good! And which element you need to put in the editor to do that? An input/output? A loop? Or?
(Timeout)
Tutor: Maybe an input element  ?
```

Figure 4 - ProGuide dialogue with the student

ProGuide allows animated algorithm simulation. The student can see how her/his solution works. Figure 5 shows the simulation of a solution to the “average” problem. The element in execution is highlighted and it is possible to view how the variables’ values change during execution.

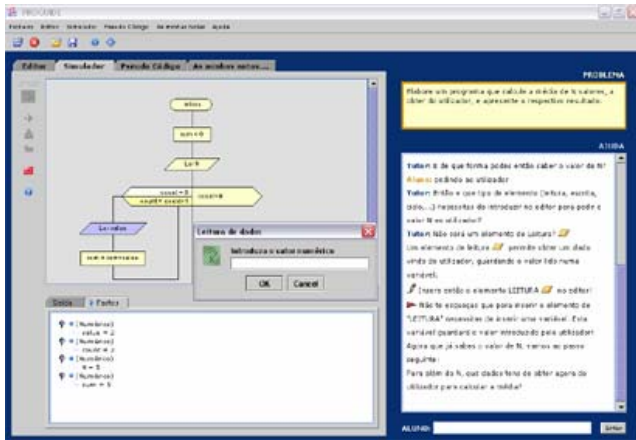


Figure 5 - Algorithm Simulation

During the dialogue the student can ask for more information about some concept that she/he wants to use, but doesn't know how. If included in the dialogue plan, ProGuide can also suggest that the student consults this extra information (for example if during the dialogue it is established that it is necessary some construct and the student says she/he doesn't know anything about it). Figure 6 shows a situation where an help about repetition structures had been asked.

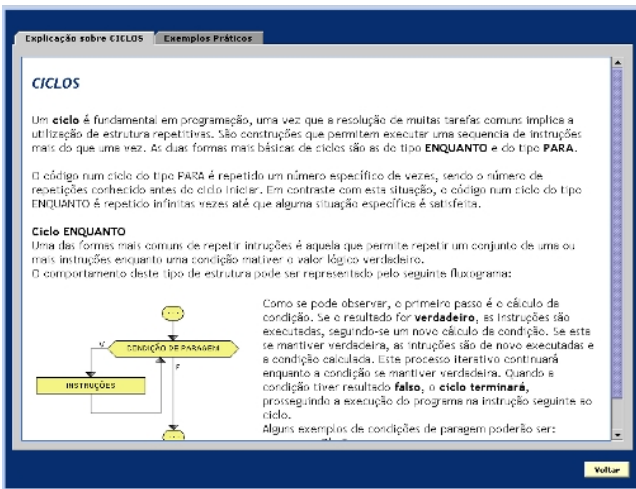


Figure 6 – Help about repetitions

If included in the dialogue plan, when the student shows many difficulties to solve the proposed problem, ProGuide can also suggest that the student studies a solution to a similar problem, so that she/he may try to transfer that knowledge to the current problem. This example solution is given as a commented flowchart, as can be seen in figure 7. In this case the example is about summing a set of numbers, which is a component of the “average” problem. The solution is commented on the left side.

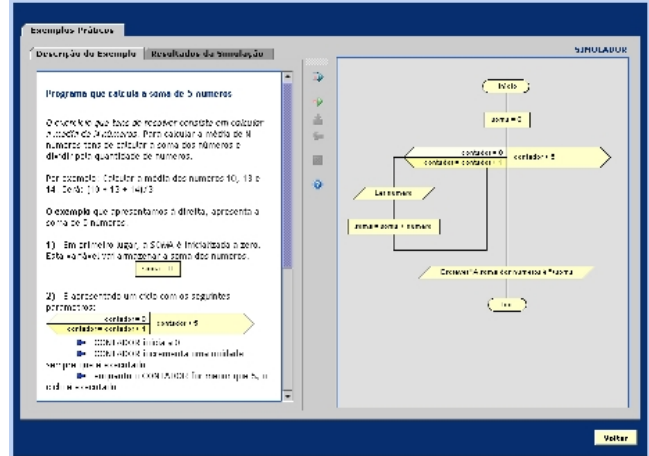


Figure 6 – Solution to an example problem

3. CONCLUSION

Programming is essentially a problem solving activity and ProGuide is an educational environment that allows novice programming students to focus on the problem solving aspects. The student can solve problems using the editor and simulation features included. When necessary the environment establishes a dialogue with the student, trying to help her/him to solve the proposed problem. This dialogue may also include information about basic programming constructs and/or similar problems solved (totally or partially) and commented.

We believe ProGuide can help novice programming students, especially those with more difficulties. The main idea is to support them in their autonomous work, avoiding the common situation where they can't even start a solution.

ProGuide wasn't yet fully evaluated with students, but we believe that it will facilitate student's learning and motivation. That was the opinion of some programming teachers that made a preliminary test of ProGuide. This preliminary evaluation resulted in some suggestions that will be included in ProGuide next version. These include improvements in the dialogue engine, a software tool to facilitate problem specification, and so on. ProGuide will be evaluated by Computer Science freshman in our institutions in the beginning of the next school year.

4. REFERENCES

- [1] Marcelino, M., Gomes, A., Dimitrov, N. and Mendes, A. Using a computer-based interactive system for the development of basic algorithmic and programming skills, *In Proceedings of CompSysTech04 - 5th International Conference on Computer Systems and Technologies*, (Rousse, Bulgaria, June, 2004).
- [2] Soloway, E. Learning to Program = Learning to Construct Mechanisms and Explanations, *Communications of the ACM*, 29(9), Sep 1986, 850-858.
- [3] Stasko, J. Tango: A Framework and System for Algorithm Animation. *IEEE Computer*, 23(9), 1990, 27-39.
- [4] Brusilovsky, P. and 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, Switzerland, June, 2004).

- [5] Esteves, M. and Mendes, A. A Simulation Tool To Help Learning Of Object Oriented Programming Basics. *In Proceedings of the 34th ASEE/IEEE Frontiers in Education Conference*, (Savannah, United States, October, 2004).
- [6] Scanlan, D. Structured Flowcharts Outperform Pseudocode: An Experimental Comparison, *IEEE Software*, 6(5), 1989, 28-36.
- [7] Reiser, J., Friedmann, P., Gevins, J., Kimberg, D., Ranney, M. and Romero, A. A Graphical Programming Language Interface for an Intelligent Lisp Tutor. *In Proceeding of CHI'88: Conference on Human Factor in Computing Systems*, (Washington, United States, June, 1988), 39-44.
- [8] Calloni, A. and Bagert J., Iconic Programming Proves Effective for Teaching the First Year Programming Sequence. *In Proceeding of the ACM SIGSE 1997 Conference* (San Jose, California, United States, February, 1997), 262-266.
- [9] Hash, E. and Zachary, J. Automated Feedback on Programs Means Students Need Less Help From Teachers. *In Proceedings of the ACM SIGSE'01 Conference*, (Charlotte, United States, February, 2001), 55-60.
- [10] A.L.I.C.E. Artificial Intelligence Foundation, "A.L.I.C.E Home Page" <http://www.alicebot.org/>, accessed on 22 June 2006.