A Simulation Tool To Help Learning Of Object Oriented Programming Basics

Micaela Esteves¹ and António José Mendes²

Abstract - In this paper we present the OOP-Anim learning environment. It was developed to help our students to learn the basic concepts of object oriented programming and to develop their programming capabilities using this paradigm. To achieve those goals students must practice intensively the development and debugging of programs. We believe this environment can help, since it uses animation to facilitate program understanding and error detection / correction. This debugging process has a lot of educational potential, as students can learn when correcting their own mistakes. When they reach a working solution, their experience and confidence normally improves, facilitating further learning. In the paper we describe the environment main features, some possible uses and the educational advantages associated with that utilization.

Index Terms – Active learning, Animation, Object Oriented Programming learning, Simulation.

INTRODUCTION

The option for object oriented languages in introductory programming courses is now common. Our own courses have migrated from C to Java in the last few years. However, the difficulties felt by many students in learning how to program correctly haven’t changed significantly. Many students continue to find difficult to understand the conceptual issues involved in programming and algorithmic design. Object-oriented programming (OOP) concepts like classes, objects, references and messages are no exception and often many students fail to understand them and, especially, to use them to solve problems.

These are some of the reasons that can explain the level of student failure common in introductory programming courses, such as [1]:

- Programs have a dynamic nature, but most learning materials have a static format (e.g. text books) which makes difficult to understand (and explain) program’s dynamic behavior.
- The abstract nature of programming makes many students fail to visualize how programming structures work and how problems can be solved using them.
- Programming learning needs a very practical approach when compared with many other courses the students are used, more dependent on theoretical knowledge and memorization.
- The large number of students that courses often have makes difficult for teachers to give individualized support to their students.
- Many times students with very different programming skills level coexist in the same course.
- Programming languages syntaxes are complex, especially for non English speaking students, and do not include visual representations for a better understanding.

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 [2]. 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.

In order to help our students, we developed OOP-Anim, an environment that can simulate and animate the execution of simple Java programs. It can support the three above mentioned learning activities. Students can use it to better understand and evaluate example programs given by the teacher and, most important, they can simulate and correct their own programs. We believe learning is more effective when students assume a more active role. We encourage them to try to solve problems we suggest, alone or in group work, and, if their solution doesn’t work as expected, to enter the Java code in the environment. There they can simulate their solution and see the corresponding animation. This allows them to compare how they thought the program would work with how the program really works. Ideally this process should lead to error detection, correction and, hence, learning. In our opinion, these activities are very important in programming learning because normally students reach a higher competence and confidence level after being able to correct all errors and have the program running correctly. This is extremely important, since after a first wrong attempt many students give up solving the problem and try to find a teacher or a colleague that shows them a solution. Even if they understand that

¹ Micaela Esteves, Assistant, Escola Superior de Tecnologia e Gestão de Leiria (ESTG), Portugal, micaela@estg.ipleiria.pt
² António José Mendes, Professor, Centro de Informática e Sistemas da Universidade de Coimbra, toze@dei.uc.pt

0-7803-8552-7/04/$20.00 © 2004 IEEE 34th ASEE/IEEE Frontiers in Education Conference

October 20 – 23, 2004, Savannah, GA
solution, learning is not as significant and many times they fail if a similar problem is proposed to them.

In programming, like in other areas, an error should not be understood as a bad thing, but as a learning opportunity. One of the important programming teacher’s tasks is to pass this message, because many students, if not all of them, are used to face an error only as a bad thing with no positive aspect. We believe that animation and simulation tools can contribute to help students take advantage of their own errors.

**RELATED WORK**

Several types of software tools have been proposed to support programming teaching and learning. Animation/visualization software systems have been used trying 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 [3] is an integrated system including an object-oriented language and an object-oriented development environment. It was developed specifically for teaching object-oriented programming to beginners. BlueJ 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.

Another known approach is to use micro worlds populated by representations of concrete entities (robots and turtles are common examples). The student can control those entities’ movements and behavior through programming instructions. OOP learning is particularly suited to this type of environments, since those concrete entities can represent well the object concept (they have characteristics and a defined set of behaviors they can present). In the literature we can find several examples of this type of tools, such as Mundo dos Actores [4] or Karel ++ [5].

Many other tools have been proposed, some of them having a more limited scope, since they animate only a particular type of algorithms and programs, such as linear data structures basic functions, binary tree primitives or sorting algorithms. Some other more generic tools have also been proposed. Most of them serve mainly demonstration purposes, as they can only animate a set of pre-defined programs, but not student’s programs. Others can animate the student’s algorithms or programs. SICAS is one of those tools [6].

Several studies have been published trying to evaluate how animation could help programming learning and which type of animations and accompanying materials are more effective [7]-[8]. Although it is possible to find studies with contradictory conclusions, it is also possible to find interesting suggestions and comments. For example, Kehoe et al. [9] concluded that when students were allowed to freely use a program animation and other study materials, they tended to use the animation together with the source code and/or a written explanation. Stasko et al. [10] used a system that animated data structures, without showing the source code. The students who used this system said they would prefer to have also a written explanation of what the algorithm is doing in each moment, and why. This study also concluded that the students wanted to have control over the animation, for example running the program step by step instead of continuously, or going back to previous steps. Miyadera et al. [11] reached similar results. They concluded that most students think it is very important to have written explanations in addition to animations, allowing them to observe the code, its data structures representation and a brief explanation of what the program is doing at each step. These results influenced some of the options we made when developing OOP-Anim, as we describe in the next section.

**OOP-ANIM ENVIRONMENT**

The environment was designed taking into consideration the above mentioned studies and also our own ideas and long experience in programming teaching at University level.

The main initial decision was to define which student activities the environment should support. Programming is essentially a problem solving activity and we believe that the most important learning activity for the students is to actively engage in solving programming problems. Of course demonstrations and explanations are important, especially in early learning stages, but trying to solve problems, correct errors and improve the solution is, in our opinion, the most important learning activity in our field. Consequently, our decision was to create a flexible environment that could support students in those activities. It doesn’t have any theoretical contents for the students to read, but it allows the students to enter their solution to a problem, simulate and animate it to visualize how it works, correct it if necessary, simulate again until a working solution is reached.

As we use Java in our introductory programming courses, we decided that the environment should accept solutions expressed as Java programs. This meant that we had to develop an environment that could analyze, interpret and simulate Java programs. A common technical approach in program animation systems, requiring that the code is annotated with special commands that are interpreted by the animation engine, could not be used, since we don’t want to add more complexity to the student’s task, as they would have to annotate their programs in order to simulate them. In consequence, we opted for a more complex development work to achieve a simpler environment to the students.

Simplicity to use was another main goal to our development. It is important that students do not need to spend significant time struggling with the environment, but instead concentrate on their programming learning tasks.

OOP-Anim main interface is divided in four areas, as shown in Figure 1.
The top left area shows the program code, allowing the system to highlight which instruction is being executed at any stage. This facilitates program understanding and error detection. The top right area shows the program animation. It has sub areas to display the program classes, objects and references. The bottom left area is used to display the program output and also the program inputs (if any) being used in a particular program simulation. The bottom right area includes control commands that students can use to control several animation aspects, such as speed, step by step or continuous, and starting, pausing and stopping the animation. This area also includes a bar that displays the program status (running, stopped, or waiting for input), and a brief explanation of what is happening in every animation step.

The possibility given to the student to go back in the animation is important to allow the repetition of any less understood part of the animation. To pause the animation allows a deeper analysis of available data is necessary (e.g. the values of some instance variables) and/or a discussion with the teacher or other learners.

In the animation area the classes are represented by a rectangle with the class name on it. The objects created during program execution are represented by a rectangle that shows the object’s class name, its instance variables (with current values) and the methods defined in it (inherited methods and own class methods). The references are represented by a black rectangle with a black ball in the middle. During the simulation the instruction currently in execution is highlighted with a different color, so that the student can easily locate what is happening.

The environment was developed essentially to support basic OOP learning. However, in order to give also some support to more advanced students it includes the possibility to animate any method belonging to any of the program classes. This allows the students to use OOP-Anim when creating larger and more complex solutions. They can focus on any method independently, so that they can concentrate on the most critical parts of their solutions or in any method not working as expected.

In the next sub sections we will describe how OOP-Anim supports student understanding and utilization of the most important object oriented programming concepts.

**Classes**

The notion of class as the entity where object characteristics are defined is stressed in OPP-Anim. It also shows that a class alone does not have operational value. It needs to be instantiated in an object to be put to work.

In the beginning of an animation, OOP-Anim analyzes the program code, identifies the classes defined in it and creates their representation in the animation area. At this stage the header of the main method is highlighted signaling the beginning of the program and only the classes (four in this example) are represented in the respective area (top right), as can be seen in Figure 2. The objects and references areas are empty, since none has been created yet.

Pressing the mouse right button over the class representation the student can access class details, namely its instance variables, methods and source code.

**Objects**

Novice students often have difficulties understanding the concept of object as an instance of a class. Many times they fail to understand how objects are created and how objects that are instances of the same class are independent entities with separated data. OOP-Anim tries to help to clarify these questions through the animation of object creation process and allowing the inspection of the objects instance variables, so that students can confirm their current values in any moment.

Figure 3 shows the animation of the object creation process. In this example a new object of the class *Triangulo* is being created. To show that fact, the class representation is highlighted with a red border and a square moves to the objects area where the new object is then created. The object...
representation is connected with its class representation and with its reference.

FIGURE 3
OBJECT CREATION ANIMATION

The object creation animation and the type of representation used can also clarify the distinction between and object and a reference to that object. That is often a difficult distinction for novice students. For example, they tend to think that an attribution between references (an instruction like ref1 = ref2;) will make equal the value of the instance variables of the objects referenced by ref1 and ref2. When such an instruction is simulated in OOP-Anim the students can see clearly that the referenced objects are not affected. In this example only the value of ref1 changes, as it will reference the same object ref2 references.

The object representation in the animation allows the student to see its instance variables with current values and the methods defined in it. Figure 4 shows an example of an object of class Circulo.

This type of representation emphasizes another important OOP concept: Data encapsulation. The object is represented as a closed box that keeps its data inside accessible only through the object’s defined methods.

FIGURE 4
OBJECT REPRESENTATION

Messages

The process of message passing between objects is also animated. When the instruction being simulated is a message the environment starts by highlighting the reference to the object receiving the message and then it changes the color of the method being invoked. If the method execution results in the alteration of any instance variable value that change is also shown in the object representation.

This functionality also allows the student to understand and use method overloading (methods with the same name, but different parameter lists). This concept can be confusing to many students especially those with previous experience of other programming paradigms where methods names must be unique.

Inheritance

Inheritance is a basic and powerful object oriented concept. OOP-Anim supports it showing the hierarchical relationships between classes and showing the inherited instance variables and methods in the sub classes based objects. Figure 5 shows an example where two classes (Dicionario and LJava) descend from the same super class (Livro).

FIGURE 5
HIERARCHY OF CLASSES

Figure 6 shows a situation where two objects were created. One is an instance of the super class (Livro) and the other is an instance of the sub class (Dicionario). The later
representation has a dotted line separating inherited methods (top) from own methods (bottom), so that students can easily see that objects from sub classes inherit its super class methods (and instance variables).

These representations allow the student to understand and use another important and often confusing concept: Method overriding. This happens when a sub class redefines a method already defined in its super class. In Figure 6 example the method void descreve(); defined in class Livro is redefined in class Dicionario. Many students find difficult to know which of the methods is called when a message with its signature is sent. This is more complex for the students as the same reference can be used to reference any object related by inheritance (in this example the reference dict could also reference an object of class Livro). Hence, the same instruction can call different methods, depending on the class of the object used.

OOP-Anim animations can help the student to understand this often confusing concept, since it is possible to observe which object is referenced and, consequently, which method is called in each situation.

PEDAGOGICAL USES

An environment like OOP-Anim can be used in several pedagogical activities, with various objectives and in different learning contexts.

As we mentioned in the previous section, the current version of the environment does not have any theoretical introduction to object oriented programming. Consequently, to use the environment autonomously the student must have some previous basic knowledge of OOP and Java syntax.

OOP-Anim was developed essentially for student use. However, this tool can also be useful to support teacher’s introduction to OOP. As mentioned before, explaining dynamic concepts based on static materials is a difficult and often ineffective task. Presentations, blackboard diagrams and daily life analogies are often used by teachers, but we think that small animations can be a better option to illustrate teachers’ explanations and to clarify doubts that students may have. It can also be used to create a more interactive environment in programming lectures, often found difficult to follow by many students, as it facilitates student participation in classes. For example, the teacher can prepare some examples with mistakes and ask the students what is wrong or can create two solutions for the same problem and discuss with them which is better and why.

The environment can also be used in programming labs to support individual or group student work. The teacher can use it to raise discussion within groups of students and to stimulate weaker students to be persistent when trying to find solutions for the proposed problems.

In our view the most important utilization for OOP-Anim environment is to support student autonomous work, individually or in group. As we mentioned before, we believe this work is fundamental for students learning. Often novice students working only with integrated development environments feel a lot of difficulties to identify the causes of their programs malfunctions. A pedagogically oriented environment can help them in such situations.

Student autonomous work can be even more supported if the teacher gives them problems to solve, together with sets of test data (inputs and correspondent expected outputs of the program) for the problem being solved. This allows the student to test her/his own program with the same data, hence validating (or not) it. This is particularly useful if the teacher specifies special cases in the input data (e.g. invalid data or data that will likely produce error situations), since many students tend to get satisfied if their solution works for an average case in the possible input data, without caring to see if it still works in other cases.

CONCLUSION

OOP-Anim is an educational environment that intends to help student to understand and master the basic concepts of object oriented programming. The student can create programs to solve programming problems possibly proposed by a teacher. The system uses animation to simulate program behavior, allowing the student to closely follow execution.

This environment can be used by students alone or in group, in classes or in self study. This can allow weaker students, many times with difficulties to cope with class rhythm, to improve their knowledge and experience.

The first prototype version of OOP-Anim is currently being evaluated by a group of experienced programming teachers and also with small groups of students. We hope that a good number of improvement suggestions will result from this work. A revised version will be produced including those suggestions and some own ideas we couldn’t develop yet (for example allow direct edition of the Java code in the environment). After this revised version is complete a larger scale evaluation will be realized involving students of our institutions and also of a Spanish University that collaborates with us in other projects.

OOP-Anim will also be used as a part in a larger project that aims to explore social aspects in programming learning.
The idea is to create an environment that facilitates student collaboration and teacher supervision in programming activities. A first description of this project can be found in [12].

REFERENCES


