

---

# PRODUCING AN EDUCATIONALLY EFFECTIVE AND USABLE TOOL FOR LEARNING, THE CASE OF JELIOT FAMILY

*Andrés Moreno and Niko Myller, University of Joensuu*

---

## Introduction

Jeliot Family is a group of program visualization tools that are designed to aid students to learn programming. Jeliot family's key feature is the fully or semi-automatic visualization of the programs' data and control flows. This means that the animation is produced from the original program code without any modifications. Thus, students and users can concentrate in programming correctly and not in how to implement the visualization. In addition to this in semi-automatic visualization the user can decide during the animation what kind of animation primitives (e.g. numbers, bounding boxes, vertical bars) he/she wants to use for the values in the program.

Development process of Jeliot has lasted for more than ten years and several versions of the concept have been developed, namely Eliot [5] (developed at University of Helsinki, Finland), Jeliot I [ref] (developed at University of Helsinki, Finland), Jeliot 2000 [2] (developed at Weizmann Institute, Israel) and the latest version Jeliot 3 (developed at University of Joensuu, Finland). The development has been research oriented and each of these versions has had their own research goals and empirical evaluations.

In this paper we present a new member of the Jeliot family, called Jeliot 3. We concentrate on the development and evaluation of the Jeliot family, giving more emphasis to the new version and its future development. Jeliot 3's main enhancement is the possibility to visualize objects. Moreover, the framework is open and more modular so new developments and improvements can easily be added to the system. It is published as open source software under the General Public License (GPL). Thus, we invite developers to enhance and extend the software. The University of Joensuu will provide tools and support to the community created around Jeliot 3.

## History of Jeliot Family

The development process of Jeliot has been research-oriented meaning that all the versions have had their own research problems rising from the previous versions' design and empirical evaluations. The development of the Jeliot family has taken more than ten years and several people have been involved during the process [1]. Different versions of Jeliot have been used in different parts of the world in courses of programming and data structures and algorithms. The feedback has been positive and it has given new directions to the development. Here we shortly describe the stage of development and the research related to them.

The development of the first version began when Erkki Sutinen and Jorma Tarhio were producing ready-made string matching algorithms' animations [11]. They found out that it took almost 100 hours to create a single animation with the tools available. To ease up the development of the process they decided to build an animation library that could be used to animate programs written in C-language. In this process the *self-animating data types* were created. Data type is self-animating when it visualizes its run-time behavior without extra code or annotations added to the program. The first version of Jeliot family, *Eliot* [6, 12], was a user interface to use the library of the self-animating data types and modify their appearance making the animation semi-automatic.

After Eliot was finished an empirical evaluation was carried out in the classroom environment and the results showed that the students were more motivated to study with this kind of environment [6, 12]. However, Eliot worked only in the X Windows environment and to port the software to other environments was laborious. This led to the development of *Jeliot I* [3, 13]. Jeliot I can be used in the Internet making Jeliot's use location independent (see <http://cs.joensuu.fi/jeliot/>).

Jeliot I used the same ideas of self-animating data types and semi-automatic animation as Eliot. It was developed in Java and it runs as an applet on the web page visualizing Java programs. It is a client-server application so that the arduous work is done on the server and the client is just a user interface to write the code and control the animation.

Jeliot I was evaluated in different kinds of situations and both quantitative and qualitative data was collected [7, 8]. The results were positive but they also gave new directions to the design. Especially, the qualitative data of Jeliot I's usage showed interestingly that the real novices had problems to handle the user interface. Furthermore, they would have needed more complete visualization of the program. Jeliot I concentrated mainly on the data visualization even though the novice students would have also required visualization of the control flow and objects structures.

From the research done with Jeliot I it was concluded that new version for novice students was needed. These results were used in design of the new version *Jeliot 2000* [2] which was especially developed for novice learners: the user interface was simplified to VCR-like buttons, animation generation was made fully automatic and animation showed both control and data flow of the program. However, it only supported a small subset of Java language.

The empirical evaluation of Jeliot 2000 [2] was carried out with high school students and the results showed that it helped in the forming of the programming concepts such as loops and conditional statements. This proves that different kind of user population needs specialized tools for their use.

### **Jeliot 3**

#### *New Version and Its Capabilities*

Currently, the object-oriented paradigm is part of many introductory programming courses. Reports from the teachers indicate that object-oriented paradigm, while trying to be closer to human reasoning model, is still hard for students to understand. Especially, the primary concepts such as inheritance, generalization and encapsulation are hard to grasp.

Owing to the Jeliot 2000's design it proved to be difficult to add new features to the implementation of Jeliot 2000. The problem was that the animation engine and Java interpreter were strongly coupled and this meant that the system needed an own Java interpreter that was made for the purpose. We did not want to develop the hand-crafted Java interpreter further.

However, the visualization engine of Jeliot 2000 was easily reusable and that was taken as a visualization engine of the new version. Jeliot 2000's hand-crafted partial Java interpreter was replaced with an open source Java interpreter, *DynamicJava* [5]. It is almost fully compliant with Java language specifications and so it gave us a solid basis for the development.

The new interpreter let us form a tool with which we can also visualize object-oriented programs. However, the two separate systems need to be combined. The interface between the two separate systems was formed by using intermediate code that was extracted during the evaluation of the program from the interpreter (DynamicJava).

Jeliot 3 is the result of the development and this process still is continuing. With this kind of design we are able to visualize a large subset of the novice level programs in Java language. At the moment the

system supports only simple object structures with no inheritance but this is going to be added to the next version. Furthermore, it allows extensions that can use the intermediate code to form new dynamic visualizations of the executed program (e.g. method call tree or diagram of object relations at run-time). At the moment, the latest release is Jeliot 3.1 and it can visualize the following programming concepts:

- *All the primitive data types* (e.g. int, boolean, char) *and strings*. Both the values of the variables and the literal values are shown.
- *One dimensional arrays of primitive type*.
- *Expressions such as operations and assignments*. This includes all unary and binary operations except the *instanceof* operator.
- *Control structures*. The execution of the program through the different control structures is visualized by highlighting the source code and displaying the information resulted of the condition evaluation.
- *Input and Output*. There are separate classes for input (e.g. Input.readInt()) and output (e.g. Output.println())
- *Errors* are reported in a descriptive way. If available, the source of error is highlighted.
- The currently supported object-oriented features are *the creation of the objects* and *access to the variables and methods of the object*.
- Next release (Jeliot 3.2) will support *inheritance* to broaden possibilities for the object-oriented programming visualization.

Jeliot 3's graphical user interface was adopted from Jeliot 2000 as it was found usable by the novices [2]. Some new features to ease the use of Jeliot during the lectures were added. The picture of the Jeliot 3's user interface can be seen in the Figure 1 and its layout in the Figure 2. On the left hand is the *editor or code view* and on the right hand side the *visualization view* called theatre as the algorithm is acted out by the actors on the stage. The *menu bar and control buttons* are located on the top and in the bottom of the screen. The output from the program is printed in the *output console* located in the bottom of the user interface.

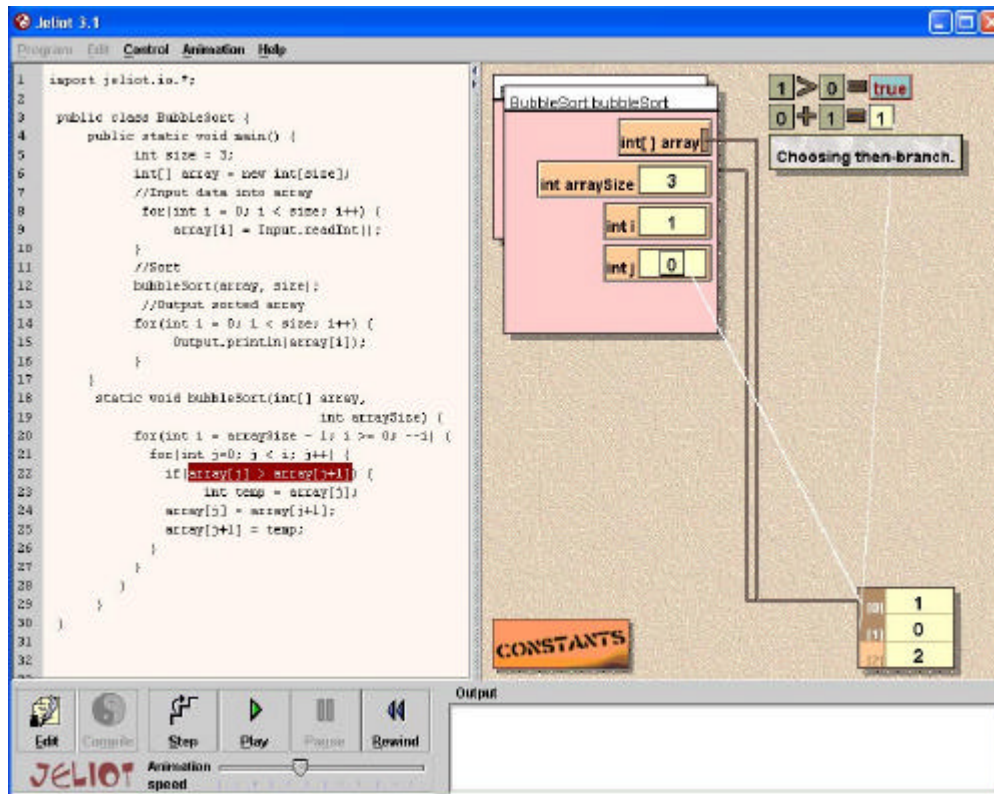
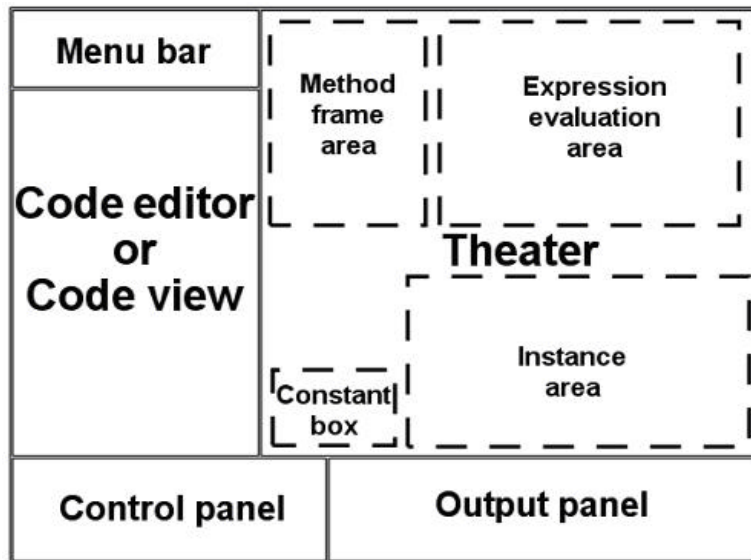


Figure 1: The user interface of Jeliot 3

There are three key issues that are stressed in the visualization of programs in Jeliot 3, namely *completeness*, *continuity* and *consistency*. Moreover, we have tried to reduce the simultaneous animated components. The reason to choose all these issues was to minimize the possibility that a student would get lost during the animation.

The completeness and continuity come from the previous version of Jeliot (Jeliot 2000) and the empirical evaluation of it [2] and partially from the theory of multimedia learning [9]. Mayer and Moreno [9] have carried out general research about multimedia learning. They argue that there are five different aspects that should be taken into consideration when creating multimedia for learning. We have used two of them in our design: *Use narration and animation rather than narration alone* and *present corresponding narration and animation simultaneously rather than successively*. In Jeliot 3 the completeness means that all the aspects of the program are visualized and the visualization components do not just appear in random places but they have their own dedicated places. Furthermore, the subexpressions of the expressions are also animated and descriptive messages are shown to the user to make all the intermediate steps visible and more understandable. The visualization is continuous so that in between the animation steps there are no discreet actions that could not be seen and be misleading for a student. Also continuity in this context means all the connections between the previous and next phase of the animation. Nothing is supposed to disappear before the next phase of the program is evaluated and so user can form connections between the concepts. For instance, the method call is not removed from the visualization before the corresponding method frame is shown and the actual parameters have flown from the method call into their places in the method frame.

As stated by Petre [10] the novices need to learn to read the graphics displays. This means that the visualizations have to be as consistent as possible to reduce the cognitive load of the novice student when he/she learns to read the graphical displays. In Jeliot 3 consistency means that all the visualized components have their own area on the screen and they always appear in that area. See the areas (dashed boxes) in the Figure 2. Moreover, the visualizations are formed as close to the Java Language Specification as possible whenever it has been pedagogically sound and sensible. In this way student can map the source code and the visualization to each other more easily.



**Figure 2: The layout of the user interface and the areas of different components in the visualization frame (Theatre)**

### *Using Jeliot 3*

Jeliot 3 can be installed to any computer with the Java Runtime Environment. It does not have any other requirement, and, as being developed in Java, it should work in any kind of computer system with Java support. After installing Jeliot 3, many possibilities emerge to learn with Jeliot:

- The lecturer can use Jeliot 3 as a part of the lecture material. He can explain different concepts and then show their corresponding animations with Jeliot. This way, students can create the correct relationships among the animation, new concept and previously learned concepts, and apply them later with reduced possibility of misunderstanding [2].
- Students may use Jeliot 3 by themselves after the lectures and do the assignments with it.
- Jeliot 3 can also be used as a tool in interactive laboratory sessions. The students can use Jeliot 3 to utilize their recently acquired knowledge by writing new programs and debugging them with Jeliot. The animations of their programs help the students understand more easily whether the programs behavior is correct or not. The teacher should be available to students, as students may require external help to understand their errors.
- Finally, it can support virtual courses. Jeliot 3 provides a tool that can aid in courses when external help is not available. After learning how to use Jeliot 3, students may continue its use remotely and visualize proposed example programs and modify them. Its visualization paradigm creates a reference model that can be used to explain problems and thus it eases the communication between student and teachers when difficulties come up.

### *Licensing and Further Development*

Jeliot 3 is a free piece of software published under General Public License (GPL). This means that the program can be downloaded and used freely. Moreover, the future platforms can be developed in collaboration by the user from different universities and other institutions. These kinds of networked teams present the idea of learning communities. In these communities the distinction between a teacher, a learner and a developer disappears thus learner can develop the tools he or she needs with the other members of the community. We want to support this kind of development and form tools that facilitate the communication between the community members.

Web-based forum is a one way to support the development. We want to form tools on the web that can help the current development, the evaluation of Jeliot and design of the future development. A

message-board, where members can discuss and keep track of ideas, is the basis for an online community. Members can raise questions, proposals, through this way. A mailing list can be used to inform about the user from the new releases and important issues of the development. The web-based forum will also contain questionnaires for students and teachers using the tool. In this way we can get more information about the usage of Jeliot and new ideas for the development.

Another possibility to support the discourse in the development community is to use a collaborative authoring tool for the code of Jeliot and for the visualizations of the programs on the Internet. The new ideas and implementations can form a continuum in which the development history and new stages of development can be seen. This could be achieved by integration of Jeliot with a collaborative authoring tool called *Woven Stories* [3].

## Conclusion

The development of the learning tool is not a simple task and requires large resources to be successful. As seen in the case of Jeliot the development cycle should always contain an empirical evaluation to address the issues where the next version of the development should focus on. This means the evaluation should not be just the end of the development but it should be also regarded as the beginning of the new development cycle.

The lesson learned with Jeliot family is that the program visualization software as well as other learning tools should aid a learner to reach his or her individual learning goals adaptively. However, these tools should also honor the individual differences in the learning styles and cognitive abilities. This can also mean that tools should be focused on certain user population and address the issues that are important to that user population like in the case of Jeliot 2000 and Jeliot 3.

In the future this can be done more efficiently and educationally effectively when students and teachers can develop the program visualization tools in together in the formed community from the basis of Jeliot 3. We are interested to see how this development evolves into new directions not yet seen.

## Acknowledgements

We want to show our gratitude to Erkki Sutinen and Mordechai Ben-Ari for their support and helpful guidance during the design and development of Jeliot 3.

## References:

1. MORDECHAI BEN-ARI, NIKO MYLLER, ERKKI SUTINEN, JORMA TARHIO (2002) *Perspectives on Program Animation with Jeliot*, (ed. Stephan Diehl) Software Visualization, Lecture Notes in Computer Science 2269, 31–45.
2. RONIT BEN-BASSAT LEVY, MORDECHAI BEN-ARI, PEKKA A. URONEN (2003) *The Jeliot 2000 program animation system*, Computers & Education 40 (1), 15–21.
3. PETRI GERDT, PIET KOMMERS, CHEE-KIT LOOI, ERKKI SUTINEN (2001) *Woven Stories as a Cognitive Tool*, Cognitive Technology, Lecture Notes in Artificial Intelligence 2117, 233–247.
4. JYRKI HAAJANEN, MIKAEL PESONIUS, ERKKI SUTINEN, JORMA TARHIO, TOMMI TERÄSVIRTA, PEKKA VANNINEN (1997) *Animation of User Algorithms on the Web*, IEEE Symposium on Visual Languages (VL'97), 360–367.
5. KOALA PROJECT (2002) *DynamicJava*, <http://koala.ilog.fr/djava/>.
6. SIMO-PEKKA LAHTINEN, ERKKI SUTINEN, JORMA TARHIO (1998) *Automated Animation of Algorithms with Eliot*, Journal of Visual Languages and Computing 9 (3), 337–349.

7. MATTI LATTU, VEIJO MEISALO, JORMA TARHIO (2003) *A visualization tool as a demonstration aid*, *Computers & Education* 41(2), 133–148.
8. MATTI LATTU, JORMA TARHIO, VEIJO MEISALO (2000) *How a Visualization Tool Can Be Used - Evaluating a Tool in a Research & Development Project*, 12th Workshop of the Psychology of Programming Interest Group, 19–32, <http://www.ppig.org/papers/12th-lattu.pdf>.
9. RICHARD E. MAYER, ROXANA MORENO (2002) *Aids to computer-based multimedia learning*, *Learning and Instruction* 12, 107–119.
10. MARIAN PETRE (1995) *Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming*, *Communication of the ACM* 38 (6), 55–70.
11. ERKKI SUTINEN, JORMA TARHIO (1993) *String matching animator SALSA*, (ed. Mati Tombak) *Third Symposium on Programming Languages and Software Tools*, 120–129.
12. ERKKI SUTINEN, JORMA TARHIO, SIMO-PEKKA LAHTINEN, ANTTI-PEKKA TUOVINEN, ERKKI RAUTAMA, VEIJO MEISALO (1997) *Eliot — an Algorithm Animation Environment*, Department of Computer Science, University of Helsinki, Report, A-1997-4, <http://www.cs.helsinki.fi/TR/A-1997/4/A-1997-4.ps.gz>.
13. ERKKI SUTINEN, JORMA TARHIO, TOMMI TERÄSVIRTA (2003) *Easy Algorithm Animation on the Web*, *Multimedia Tools and Applications* 19 (2), 179–184.

**Authors:**

Andrés Moreno and Niko Myller  
University of Joensuu, Department of Computer Science  
P.O. Box 111  
FIN-80101 Joensuu  
FINLAND  
{amoreno, nmyller}@cs.joensuu.fi