Συνέδρια της Ελληνικής Επιστημονικής Ένωσης Τεχνολογιών Πληροφορίας & Επικοινωνιών στην Εκπαίδευση

Τόμ. 1 (2008)

4ο Συνέδριο Διδακτική Πληροφορικής

The Effect Of The Jeliot Animation System On Learning Elementary Programming

Ben-Ari Mordechai (Moti)

Βιβλιογραφική αναφορά:

Mordechai (Moti), B.-A. (2023). The Effect Of The Jeliot Animation System On Learning Elementary Programming . *Συνέδρια της Ελληνικής Επιστημονικής Ένωσης Τεχνολογιών Πληροφορίας & Επικοινωνιών στην Εκπαίδευση*, *1*, 021–030. ανακτήθηκε από https://eproceedings.epublishing.ekt.gr/index.php/cetpe/article/view/5843

The Effect Of The Jeliot Animation System On Learning Elementary Programming

Mordechai (Moti) Ben-Ari

Department of Science Teaching
Weizmann Institute of Science
moti.ben-ari@weizmann.ac.il

Περίληψη

Στην εργασία αυτή παρουσιάζεται το Jeliot, το οποίο αποτελεί ένα σύστημα προσομοίωσης προγραμμάτων για τη διδασκαλία των εισαγωγικών μαθημάτων του προγραμματισμού. Το Jeliot μπορεί να δημιουργήσει αυτόματα οπτικοποιήσεις προγραμμάτων σε Java υποστηρίζοντας τη διάλεξη του διδάσκοντα και την εργαστηριακή εξάσκηση των μαθητών. Η παιδαγωγική έρευνα της εφαρμογής του Jeliot σε μαθητές δευτεροβάθμιας εκπαίδευσης στα πλαίσια μαθημάτων προγραμματισμού έχει δείξει ότι: (a) το Jeliot βελτιώνει τη μάθηση στον προγραμματισμό παρέχοντας στέρεες αναπαραστάσεις των δυναμικών πτυχών ενός προγράμματος, (β) το Jeliot βελτιώνει την προσοχή των μαθητών, (γ) πολλοί εκπαιδευτικοί ενσωματώνουν τη χρήση του Jeliot στο έργο τους ενώ άλλοι θεωρούν ότι έχουν περιορισμένο έλεγχο και κάνουν περιορισμένη χρήση ή απορρίπτουν τη χρήση του στην πράξη, (δ) σε ένα συνεργατικό πλαίσιο εφαρμογής τα εργαλεία οπτικοποίησης προγραμμάτων επηρεάζουν το επίπεδο εμπλοκής των μαθητών.

Keywords: *Program visualization, teaching introductory programming, pedagogical research*

Abstract

The Jeliot program animation system for teaching introductory programming automatically animates programs in Java so that the visualizations can be created during a lecture or laboratory. We have carried out pedagogical research on the use of Jeliot in high school computer science classrooms; the research has shown that: (a) Jeliot improves learning by providing a concrete representation of the dynamic aspects of a program; (b) attention is improved; (c) while many teachers internalize the use of Jeliot, others perceive a lack of control and may limit or reject its use; (d) in a collaborative setting, the use of a visualization tool affects the level of engagement.

Keywords: *Program visualization, teaching introductory programming, pedagogical research*

1. Introduction

One of the most important difficulties confronting novices learning programming is the difficulty of forming a mental model of the dynamic execution of a computer program (Ben-Ari, 2001). The reason is that a short piece of static source code gives

rise to a complex sequence of actions during the dynamic execution. For example, a simple for-loop:

```
for (int i = 0; i < 10; i++)
      sum = sum + a[i];
```

gives rise to the following actions:

Allocate memory for `i`.
Initialize `i` to 0.
Evaluate `i < 10`.
Decide whether to continue (result of evaluation is/is not zero).
Evaluate the expression `sum + a[i]`.
Assign the result to `sum`.
Increment `i`.
Jump to the evaluation.

In object-oriented programming, the execution of a single statement can be even more complex:

```
MyObject m = new MyObject(a, b);
```

Allocate memory for `m`.
Allocate memory for the new object.
Evaluate the arguments.
Call the constructor with arguments.
Allocate memory for the parameters and local variables.
Initialize the parameters and local variables.
Execute the statements of the constructor.
Return a reference to the object.
Assign the reference to `m`.

Program animation is based upon the claim that showing a concrete representation of the execution of a program can improved students' ability to understand the relationship between the static and dynamic aspects.

2. The Jeliot program animation system

Jeliot was originally developed by a team at the University of Helsinki under the direction of Jorma Tarhio and Erkki Sutinen. The current version, Jeliot 3, is a collaborative project of a team at the Weizmann Institute of Science under the direction of Moti Ben-Ari and a team at the University of Joensuu under the direction of Erkki Sutinen. Most of the work has been carried out by graduate students, primarily, Niko Myller and Andres Moreno (program development) and Ronit Ben-Bassat Levy (pedagogical design and research). For an overview of Jeliot, see Ben-Ari et al. (2002), or Moreno et al. (2004).

Jeliot is available from `http://cs.joesuu.fi/jeliot` under the GNU General Public License.

The animation is constructed automatically from the program text in Java; no additional effort is required from the teacher or student. Every dynamic feature of the program is visualized: memory allocation, loading and storing of variables, expression evaluation, control statements and method calls (Figure 1). The animation is continuous and there is a simple interface: a single window with two panels (source code and animation) and familiar VCR-like animation controls.

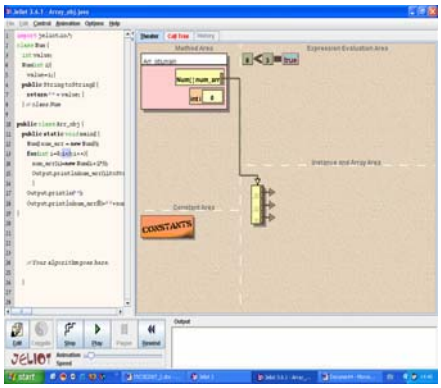

Figure 1: *Jeliot user interface*

3. Learning objects for visualization

A learning object is a small, self-contained resource for learning a specific topic (Boyle, 2003). Learning objects are usually built with multimedia tools like Flash. Since Jeliot can automatically create the *dynamic visual component* of a learning object, all that is needed in addition are source code to be animated and a guide for studying the animation. I have developed a set of 44 learning objects for introductory Java programming, which can be freely used and modified under the Creative Commons A-N-SA license. The website is:

http://stwww.weizmann.ac.il/g-cs/benari/lov/index.html

Each learning object consists of a Java source code file and written instructions consisting of the following paragraphs:

An overview of the concept to be learned.
A specification of the example program.
A guide for studying the animation.
One or more programming exercises so that the student can check his/her understanding.

4. Jeliot improves learning

In her MSc thesis, Ronit Ben-Bassat Levy showed that the use of Jeliot improves learning of programming (Ben-Bassat Levy, et al. 2003). She conducted an experiment on tenth-grade high school students studying an introductory course on algorithms and programming. Two classes were investigated: one where Jeliot was used and one that was a control group. The study was carried out for a *full year* ensuring that the results were the result of a long-term change in a real educational environment and did not result from a short-term change that would be characteristic of an hour-long experiment.

The course language was Pascal, but the syntax differences were not important for the introductory concepts that were studied. Each new concept was taught in the same way in each class, but an extra hour was added on: the animation group received demonstrations using Jeliot, while the control group received the same demonstrations using Turbo Pascal. Tests and assignments used questions of varying levels of understanding, and interviews were conducted during which the students were asked to solve a problem while verbalizing their thoughts. The interviews with the animation group also checked their attitudes towards Jeliot.

Since we could not control membership in the classes, the control group was always stronger, as reflected in higher average grades than the treatment group. But we did demonstrate greater *improvement* within the treatment group that used animation. The animation group used the representation of memory cells and execution paths in explanations, while the control group had difficulties in explaining how a cell gets its value. For example, in an interview students were asked to predict the outcome of x := x + 1. The students in the animation group imitated the graphics and gave a correct explanation, while those in the control group said that the statement is incorrect. Here is a transcript of two students from the animation group discussing the statement, where we have emphasized terms that refer to Jeliot, although Jeliot was not being used at that time:

> V: There is an error. It is a false statement.
> D: You are wrong. This is not mathematics, it is Pascal.
> V: So what; it is impossible that x will be equal to x+1
> D: But we saw in the lab that it takes what there is in the *rectangle* at the side and puts it on the *marble* [the students' term for the animation pane in Jeliot], and then adds one, and then puts the answer back into the *rectangle*. [Draws the Jeliot screen on the blackboard.]
> V: Wow, did I ever make a mistake! I forgot about it. [Turns to the teacher] I forgot. What am I going to do?!
> T[eacher]: What we saw in the lab was the execution of y := x + 3. You explained here the execution of x := x + 1. Are the statements similar or different?
> D: I think they are both similar and different.

T: What do you mean?
D: They are similar because of the right-hand side, where it takes what there is in the *rectangle*, puts it on the *marble* and then adds one. They are different because of the left-hand side: in the lab y was written and here x is written.
T: What do you mean by "take"?
D: It copies what there is in the *rectangle* x. I remember now, in the lab you said "copies." I don't care if x or y is written on the left-hand side, because it is a computer and it does the same thing all the time. So instead of putting it in y he will put it in x.

Later during the school year, the contribution of Jeliot was even more obvious when learning a complex concept liked a nested if-statement. In the control group, the stronger students could answer the questions only after many attempts and had difficulty explaining them, while the other students could not answer the questions.

In the treatment group, the stronger students had difficulties answering this question, because they believed they could understand the material without Jeliot. The weaker students refused to work on the problem, claiming that nested if-statements are not legal, or that they did not understand it. The average students gave correct answers! They drew a Jeliot display and used it to hand simulate the execution of the program.

We found that animation does not improve performance of all students. Stronger students do not need it, while weaker students are overwhelmed by the tool. The concrete model offered by the animation can make the difference between success and failure for many students. Since the use of Jeliot did not harm the grades of either the stronger or the weaker students, there is no reason not to use it.

The students in the animation group used a different and better vocabulary of terms than did the students in the control group. Verbalization is the first, and most important, step to understanding a concept, thereby justifying the use of animation. In any case, animation must be a long-term part of a course, so that students can become familiar with the tool.

5. Jeliot improves attention

In his MSc thesis, Gil Ebel showed that the use of Jeliot improves the attention-directing characteristics of students (Ebel & Ben-Ari, 2003).

Definition: A student is *attentive* if the student is looking where he or she is supposed to be looking, where "supposed to'" is defined as the explicit or implicit intention of the teacher.

Attention is the first step in the learning process. We cannot understand, learn or remember that which we do not first attend to. The quality of attention correlates with learning effectiveness; the time students spend paying attention is a good predicator of their achievements.

The study involved ten high-school students studying a course *Foundations of Computer Science*. The students suffered from a variety of emotional difficulties and learning disabilities, including attention deficiency and hyperactive disorder, but they had normal cognitive capabilities. Four lessons were video-taped. We measured the rate at which unattentive behavior occurred in order to correlate it with the type of activity during which it occurred. Our definition of unattentive behavior was limited to easily recognized "acting-out" behaviors called *episodes of recognizable unattentive attitude (ERUA)*. A behavior is classified as an ERUA if it fulfils the following two criteria:

The student is performing an act that has nothing to do with the lesson;
The act performed is offensive toward the teacher or another student.
Here are some examples of ERUAs:
Disputes over objects: a student picks up a ruler from the next table, drops it, get cursed by the owner of the ruler, picks it up and refuses to return it.
Disrespect for the teacher: A student teases the teacher in a provocative way about a spelling error or the quality of his handwriting.
Deliberate interference with other students: When one student starts to ask a question, another student falls on the table and starts to snore loudly.
Calls attention to himself: A student claims that he needs to leave the class now or he will do "irrational" things.
The results showed that when Jeliot was used, there was a total absence of ERUAs, while when it was not used, there were an average of one ERUA every two minutes. This result was the same over the four lessons analyzed so that it cannot be attributed simply to the novelty of seeing the animation tool for the first time.

We believe that the behavior modification occurred *because* of the use of Jeliot. The improvement in the students' behavior in itself justifies using an animation system during classroom sessions. While we did not show directly that increased attention leads to learning, although this would seem to follow from the literature, as well as from common sense.

6. Acceptance of Jeliot by teachers

In her PhD thesis, Ronit Ben-Bassat Levy is studying the reasons why teachers do or do not use Jeliot in their classrooms (Ben-Bassat Levy & Ben-Ari, 2007; Ben-Bassat Levy & Ben-Ari, submitted). Teachers are the unavoidable link between visualization tools and the students, but they are not always effective agents for introducing the use of such tools, and frequently even oppose their use. Why?

We used the research framework called *phenomenography* developed by Ference Marton at Göteborg University, Sweden. It has been applied to CS education by Shirley Booth and Anders Berglund. Phenomenography focuses on the qualitatively

different ways that people experience, understand, perceive or conceptualize a phenomenon. The number of qualitatively different *ways of experiencing* a phenomenon is limited and they form a hierarchy.

Teachers and student teachers from various courses both on Jeliot itself and on Java programming were studied using interviews, exercises, reflections, and questionnaires. The statements made by teachers were taken from their original context (decontextualised), and gathered again under a category (recontextualisation) that was common to all those statements.

Table 1 shows the results of the phenomenographic analysis, namely, the different ways of experiencing Jeliot and the typical actions associated with each category. Here are some typically quotes associated with each category:

Internalization: I am using Jeliot as a tool that can explain all the previous information needed, and I created a library that contains all the necessary information to the subject that is being discussed in class.
By-the-book: I use Jeliot only when I teach a new subject or when I feel the need to attract my students.
Rejection: I do not see why I have to use Jeliot, my students always understand me. (We call this the *centrality of the teacher*.)
Dissonant: I will have to use Jeliot since I do not want to be different from other teachers who use it.

Table 1: *How teachers experienced Jeliot*

Ways of experiencing	
Internalization	Jeliot is experienced as a useful tool consistent with the teacher's pedagogical style
By-the-book	Jeliot is experienced as a possibly useful tool, but it may not fit with the teacher's pedagogical style
Rejection	Jeliot is experienced as an externally imposed tool of limited usefulness for teaching
Dissonant	Jeliot is experienced in a conflicting manner: on the one hand with enthusiasm and on the other with a reluctance to actually use it
Typical actions associated	
Internalization	Frequent use of the animation; use of novel methods; use for inquiry; integration of the animation system into routine classroom activities
By-the-book	Work by the book; use the animation system on special occasions such as teaching a new subject
Rejection	Do not use the system at all; confident that the animation system is not a useful pedagogical tool
Dissonant	Use the system only when required such as when someone comes to observe; use under compulsion

As a result of the phenomenographic analysis, it became clear that the dissonant teachers were the problem. Why did they tend not to use Jeliot even though they did not explicitly reject it? To answer this question we looked at attitudes that determine behavior using Izak Ajzen's *theory of planned behavior*. The most important aspect of the teachers' attitudes turned out to be their *perceived behavior control (PBC)*: The extent to which a person feels that he/she can control the behavior. How much control a person believes he/she has on the behavior (control beliefs). How confident a person feels about his/her ability to behave in a certain way (the influence of the control beliefs on the person).

We were able to develop predictors that characterized each of the categories from the phenomenographic analysis:

Internalization: positive attitudes, high PBC.
By-the-book: positive attitudes, low PBC.
Rejection: negative attitudes, high PBC.
Dissonat: positive attitudes, high direct PBC, low indirect PBC.
The difference between a direct measurement and an indirect measurement is whether you ask the subject directly what his/her attitude is or you infer it from indirect questions. A surprising result was a relatively low level of indirect PBC in all subjects! That is, even if they directly expressed confidence in their ability to control the software tool and its use in the class, they nevertheless experience some apprehension. We are currently working on suggestions for overcoming this problem.

7. How does visualization affect collaboration?

In his PhD thesis, Niko Myller is studying how the use of visualization tools (Jeliot and BlueJ) affect collaboration (Myller et al., submitted). Collaborative learning and pair-programming are widely used in CS classrooms, so visualizations are used in collaborative contexts. The question arises: How are visualizations actually used in collaborative contexts and how do they affect the collaboration?

The *Engagement Taxonomy (ET)* was develop to describe interactions that visualizations promote (Naps et al., 2002). It is claimed that a higher-level of engagement between the learner and the visualization results in better learning outcomes. We extended the ET to an *Extended Engagement Taxonomy (EET)* to take into account interactions characteristic of tools like Jeliot and BlueJ that visualize aspects of program automatically from the source code, as opposed to visualizations that are created directly. (In the following list, (*) denotes categories from the original ET.)

No viewing (*): There is no visualization to be viewed but only material in textual format. For example, the students are reviewing the source code without modifying it or they are looking at the learning materials.

Viewing (*): The visualization is viewed with no interaction. For example, the students are looking at the visualization or the program output.
Controlled viewing: The visualization is viewed and the students control the visualization, for example by selecting objects to inspect or by changing the speed of the animation.
Entering input: The student enters input to a program or parameters to a method before or during their execution.
Responding (*): The visualization is accompanied by questions which are related to its content.
Changing (*): Changing of the visualization is allowed during the visualization, for example, by direct manipulation.
Modifying: Modification of the visualization is carried out before it is viewed, for example, by changing source code or an input set.
Constructing (*): The visualization is created interactively by the student by construction from components such as text and geometric shapes.
Presenting (*): Visualizations are presented and explained to others for feedback and discussion.
Reviewing: Visualizations are viewed for the purpose of providing comments, suggestions and feedback on the visualization itself or on the program or algorithm.
Student groups were video-taped as they solved exercises in elementary programming. They used two visualization tools: Jeliot and BlueJ. Samples from the video tapes materials were coded for: (a) the categories of the EET, (b) the activities that the students engaged in, (c) the number of participants in each activity, (d) the contents of the discussion and (e) transactive reasoning, which is talking about one's thinking process or one's understanding of the partners' thinking processes.

Comprehensive results are given in Myller et al. (submitted). Here is an example of one Table 2. The columns show various activities, while the rows show the category of the EET that the students were engaged in.

Table 2: *Students' engagement with visualization tools*

EET/ Activity	Conversing	Listening to instructor	Silent	Count
No viewing	**47.5%**	27.5%	25.0%	160
Viewing	36.2%	15.9%	47.9%	340
Entering input	**57.1%**	12.4%	30.4%	161
Overall	44.0%	17.9%	38.1%	661

We can see, for example, that when the students were viewing the visualization, they engaged in *less* conversation than when they weren't viewing! Furthermore, the discussion contents were more related to the program code on the no viewing level than on the others. This is somewhat counter-intuitive, because one would expect

students to talk about the visualization that they are viewing. We did find, however, that at higher levels of the EET, there was more transactive reasoning. The conclusion is that visualization by itself does not facilitate collaboration; this will only occur if the system encourages high levels of interaction.

8. Conclusions

There are dozens of pedagogical software tools for computer science, but they are not as widely used one would expect. Part of the explanation is that tools are helpful only in certain contexts or only for certain students or only when used in certain ways. The development of such tools should be accompanied by pedagogical research that can guide instructors in the use of these tools.

Our research has shown that the Jeliot program animation system is effective in improving learning and has discovered the conditions that are favorable for its use.

References

Ben-Ari, M. (2001). Constructivism in computer science education. J*ournal of Computers in Mathematics and Science Teaching*, 20(1), 45–73.

Ben-Ari, M., Myller, N., Sutinen, E. & Tarhio, J. (2002). Perspectives on program animation with Jeliot. In S. Diehl, editor, *Software Visualization, Lecture Notes in Computer Science*, 2269, 31–45.

Ben-Bassat Levy, R., Ben-Ari, M. & Uronen, P.A. (2003) The Jeliot 2000 program animation system. *Computers & Education*, 40(1), 15–21.

Ben-Bassat Levy, R. & Ben-Ari, M. 2007. We work so hard and they don't use it: Acceptance of software tools by teachers. *Twelfth SIGCSE Conference on Innovation and Technology in Computer Science Education*. Dundee, UK, 246–250.

Ben-Bassat Levy, R. & Ben-Ari, M. (submitted). *Perceived behavior control and its influence on the adoption of software tools*.

Boyle, T. (2003). Design principles for authoring dynamic, reusable learning objects. *Australian Journal of Educational Technology* 19(1), 46–58.

Ebel, G. & Ben-Ari, M. (2006). Affective effects of program visualization. *Second International Computing Education Research Workshop*, Canterbury, UK, 1–5.

Moreno, A., Myller, N., Sutinen, E. & Ben-Ari, M. (2004). Visualizing programs with Jeliot 3. *Conference on Advanced Visual Interfaces*, Gallipoli, Italy, 373–376.

Myller, N., Bednarik, R., Ben-Ari, M., & Sutinen, E. (submitted) *Extending the engagement taxonomy: Software visualization and collaborative learning*.

Naps, et al. (2002). Exploring the Role of Visualization and Engagement in Computer Science Education, *Working Group Reports from the 7th Conference on Innovation and Technology in Computer Science Education*, Aarhus, Denmark, 131–152.