Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 178 (2007) 51–59
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Jeliot 3 in a Demanding Educational Setting

Andrés Moreno[1]

Department of Computer Science
University of Joensuu
Joensuu, Finland

Mike S. Joy[2]

Department of Computer Science
University of Warwick
Coventry
UK

Abstract

We report the preliminary findings of a qualitative investigation into how students approach a program visualization tool, and whether the approach depends on how they are taught to use the tool. Volunteer students in an undergraduate programming course were divided into two groups. One group was taught programming concepts explicitly using the tool, and required to use it to solve weekly exercises and projects, whereas the other group only used the tool on a voluntary basis. We identify those aspects of using the tool which the students find beneficial, and discuss the limitations of the animations provided by Jeliot 3.

Keywords: Program Visualization, Jeliot 3, Experiment, Programming

1 Introduction

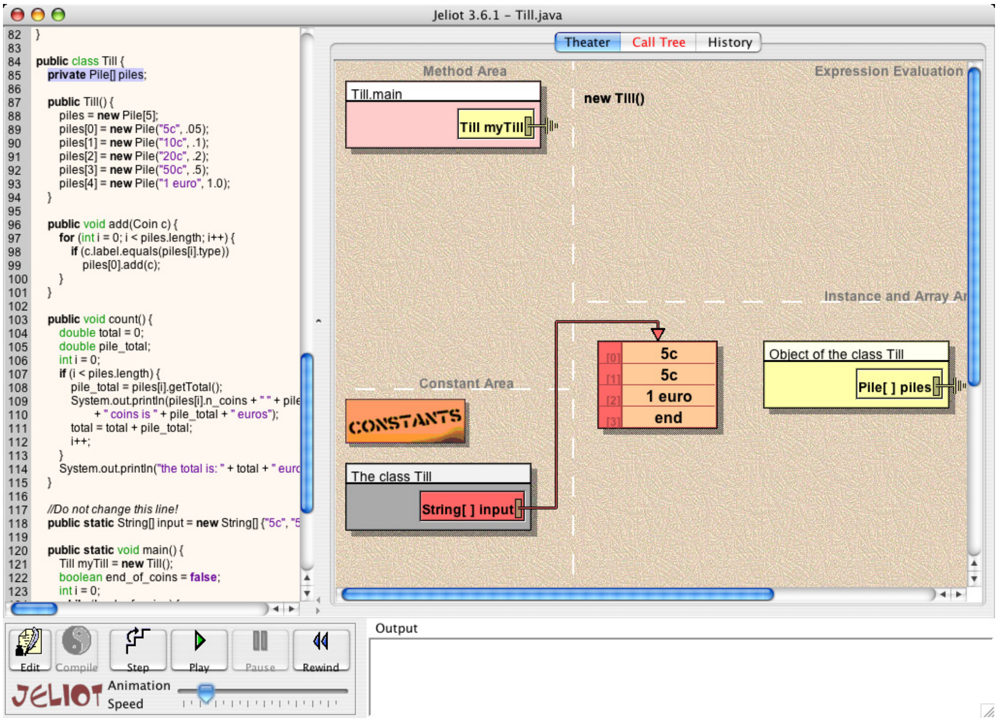
Software Visualization (SV) tools are intended to be used in the early stages of the learning path of a programmer, teaching them the basics of programming [7], algorithms [5], and the software development cycle [10]. SV tools convey important information by means of graphics or animations. Although it is believed that visual representations are useful, previous evaluations of Software Visualization tools suggest that such tools do not significantly improve the learning outcome [2]. Nonetheless, some experiments [4] have shown that they motivate the students when learning algorithms or basic programming skills.

[1] Email: `Andres.Moreno@cs.joensuu.fi`
[2] Email: `M.S.Joy@warwick.ac.uk`

doi:10.1016/j.entcs.2007.01.033

1.1 Jeliot 3

Jeliot 3 [7] is a program animation tool oriented towards novice programmers, in which animations represent the step by step execution of Java programs, see Fig. 1. Each step in the execution of a program is graphically made explicit, and the resulting animation is a simulation of how the virtual machine interprets the program code. The animation takes place in a "theater" that is divided in four separated areas. The central and main area is the "Expression Evaluation" one, to which messages, method calls, values and references are moved to and from the other visualization areas as the evaluation proceeds. Students can program in Jeliot 3, and later they can visualize their program and follow its execution path.


```
}

public class Till {
   private Pile[] piles;

   public Till() {
      piles = new Pile[5];
      piles[0] = new Pile("5c", .05);
      piles[1] = new Pile("10c", .1);
      piles[2] = new Pile("20c", .2);
      piles[3] = new Pile("50c", .5);
      piles[4] = new Pile("1 euro", 1.0);
   }

   public void add(Coin c) {
      for (int i = 0; i < piles.length; i++) {
         if (c.label.equals(piles[i].type))
            piles[0].add(c);
      }
   }

   public void count() {
      double total = 0;
      double pile_total;
      int i = 0;
      if (i < piles.length) {
         pile_total = piles[i].getTotal();
         System.out.println(piles[i].n_coins + " " + pile
               + " coins is " + pile_total + " euros");
         total = total + pile_total;
         i++;
      }
      System.out.println("the total is: " + total + " euro
   }

   //Do not change this line!
   public static String[] input = new String[] {"5c", "5

   public static void main() {
      Till myTill = new Till();
      boolean end_of_coins = false;
      int i = 0;
```

Fig. 1. Screenshot of Jeliot animating the creation of a new object.

2 Literature Review

Ben-Bassat Levy et al.[1] studied the pedagogic advantages of Jeliot 2000 (the predecessor to Jeliot 3) compared with the use of an Integrated Development Environment (IDE), TurboPascal, to trace programs. Their study divided high school students in an Introduction to Programming Course into two groups. Both of them attended the same lectures, but in their laboratory sessions one group used Jeliot 2000, and the other used TurboPascal.

The results identified an improvement in understanding amongst those who used Jeliot 2000. The largest improvement came for the mediocre students – Jeliot 2000 represented a viable working model of the execution of a program that they could understand and follow. However, the strongest and weakest students appeared not to gain much from the tool, since it was either too simple or too complex for them.

In another experiment, qualitative data were gathered from 35 students who were taking part in a second course on programming and were using Jeliot 3 [3]. Students were asked to reflect on their Jeliot 3 usage while solving programming and debugging tasks. The findings noted the problems that Jeliot 3 presented to those students with some experience in programming and to those without previous use of Jeliot 3. Again, novice students were more positive regarding the usage of the tool.

Kehoe et al. [4] argue that a different approach was needed to demonstrate the effectiveness of algorithm animations. They hypothesize that the pedagogic value of algorithm animation tools would be more apparent when used in an interactive setting, such a homework assignment. They claimed that the pedagogic value of such tools would increase if accompanying instruction is provided at the same time as the animation. Further, a meta-analysis notes the importance of how the animation is administered to the students, rather than what is actually visualized [2].

Petre [9] discusses the skills required to understand visualizations, and how novices and experts extract different information from them. The type of training given in using the visualization influences the development of the student's skill to notice secondary notation cues. The acquisition of this skill distinguishes novices from experts; hence, it is considered fundamental to understanding visualizations.

3 Purpose

The idea for our investigation was to extend the experiment done by Ben-Bassat Levy et al. [1]. However, rather than comparing two different tools, this investigation focuses on students' use of a single visualization tool in two different learning contexts, and compares their attitudes towards the tool and their usage of the tool. More importantly, we have sought to check whether the information conveyed by the animation is correctly understood by both groups, even if they have been using the tool differently. We may expect using the tool in different contexts should produce different effects on the students, both in the their performance using the tool [4] and in their understanding of the taught concepts [9]. We formulate the following hypotheses:

- *H1*: Jeliot 3's animations are comprehended better by those students who have been required previously to solve tasks with Jeliot 3. For example, they should be able to follow and reproduce what it is happening in the animation theater in greater detail than the students that have not been explicitly instructed.
- *H2*: Students' expectations from the tool should be different depending on how they have approached it, and we can distinguish two possible uses of the tool, either as a learning aid, or as a debugger.

4 Methods

The participants in the study consisted of 6 Maths undergraduate students in an introductory course on programming, who voluntarily used Jeliot 3 as a programming environment for their weekly tasks and projects. They were divided in two groups, the "voluntary" group and the "normal" group. The former consisted of two students who attended an extra session each week during which where Jeliot was used exclusively. One of the students reported having prior programming experience of VisualBasic, and one other was highly proficient in the Microsoft Office suite, but none had previous Java programming experience.

All participants were taking part in a 10-week long "introduction to programming" module for Mathematics undergraduate students taught at the University of Warwick. The module lecturer collaborated with the experiment by encouraging students to use Jeliot 3 and by using it in the initial lectures and lab sessions.

The tool was deployed as a Java web-start application. A web-page[3] was set up to host the application, along with the on-line documentation.

4.1 Procedures

For half an hour each week, the "voluntary" group of students completed a set of extra mini-tasks, designed to explain one or more concepts using Jeliot 3. These taught concepts were closely related to the material taught during that week's lecture, and with what they were expected to use when solving the weekly assignments. Students completed these mini-tasks individually in a controlled laboratory environment, with the researcher (the first author) present to help them to understand the visualization and to make progress with the mini-tasks. Students' answers to these mini-tasks were collected, and the researcher checked their answers for programming and animation misconceptions, and gave further explanation to the students as to why their answers were incorrect or what the animation meant. This was the only difference between the two groups.

Both the "normal" and the "voluntary" group used Jeliot 3 at the weekly lab sessions (2 hours long each), and at home to complete their tasks. The only support given was that specifically requested during the lab sessions.

During the first four weeks, the researcher was present in the ordinary lab sessions as a teacher assistant, interacting with all the students taking part in those session (no more than 20 students), and writing down his own observations, and the attitudes and problems students found in Jeliot 3.

After the 4th week, when the students were supposed to move on to more complicated project work, semi-structured interviews were carried out with the six students from both groups.

[3] http://www.cs.joensuu.fi/jeliot/warwick/

4.2 Data Analysis

The qualitative data analyzed in this report consists of students' responses during the interviews. The core of the interview explored students' attitudes towards Jeliot 3, for example their knowledge about programming concepts, and whether they would continue to use Jeliot in future assignments. Assessing their knowledge was performed in two steps. They were asked firstly to explain which steps Java went through when creating an object, and secondly to describe the animation of an object being created as they watched it in Jeliot. In total, six interviews were audio recorded and transcribed.

A list of sentences was extracted from the transcripts and grouped according to how Jeliot was useful to the students, and how they used it. As expected, two main categories emerged: Jeliot as a learning aid, and as a debugger. Subsequently, the researcher's notes from the lab sessions were incorporated and used to further explain the reasonings given by the students and to illustrate the behavior of all the students taking part in the lab sessions, whether using Jeliot 3 or not.

5 Results

5.1 Jeliot as a learning aid

All of the students mentioned the existence of a "gap" in the course. At different points of the course, they realized that the requirements needed to follow the rhythm of the course and the practical assignments changed abruptly:

> "And it seems very simple, and the next week you do like, you jump, there is a big jump in the middle. And the lecture doesn't quite seem to follow the lab session enough."

As the student mentions, the gap appeared between theory and the application of that theory; however, a gap also appeared when students were not able to grasp a new concept. Arrays and objects, but also the basic syntax, seemed to be troublesome, and students asked for further explanations on those topics:

> "Maybe, if we had some sort of introduction more into the syntax of Java, and maybe a bit more of explanation, sort of why it is doing that."

Students' answers suggested that the animations involving arrays and objects were the most useful ones. Only one student, from the "normal" group, did not mention Jeliot to be of help for the gap. All the other students used Jeliot 3 when they tried to close a gap which had opened in the course, and they found that Jeliot 3 was useful then, no matter which group they were from.

> "It [Jeliot 3] is really useful because you can follow it, go through the code [...]. It is helping me to understand how I create objects. [...] It clarifies things, rather than just being a white screen."

Even if they claimed that Jeliot helped them to understand concepts, they failed to reproduce the execution path that follows an object creation. Only two of the students from the "normal" group produced an acceptable description of the allocation process. Both of them were aware that Jeliot had helped them.

> "[...] but as now we are learning new concepts and new objects, it [Jeliot 3] is useful."

The other students showed some misunderstandings when describing the animation of an object allocation. The `this` reference, and argument passing from the constructor call to the constructor frame, caused most of the problems. For example, some of them expected parameters to be passed "by value" straight to the object. Nonetheless, the animations corresponding to basic statements, e.g., assignments or variable declarations, were correctly described by all subjects but one from the "normal" group.

5.2 Jeliot as a debugger

Only one of the participants, who belonged to the "voluntary" group, did not used Jeliot 3 to complete the assigned tasks during the four observed lab sessions. The others used Jeliot 3 intensively during the lab sessions.

Most of them followed an iterative approach when completing the tasks: (i) read instruction, (ii) write method, and (iii) create and run short test. The first iterations of this cycle can be understood as part of the learning process. The following iterations had a clearer debugging aim. They used Jeliot 3 to visualize and identify bugs:

> "[...] with Jeliot, you played through it, [...] a number goes to somewhere it shouldn't do, and, oh!, that bit is wrong."

When dealing with project work which consisted of several files, they had to temporarily abandon Jeliot 3 because it does not support classes a set of classes stored in different files. However, they came back to use Jeliot 3 when they did not understand why their programs were not working:

> "If a particular thing I was having trouble understanding, I pasted it into Jeliot and play around with it, so we had values that actually existed."

It was while using Jeliot 3 as a debugger that students found out the current limitations of the tool. Most of them suggested the possibility of being able to "jump through the animation" in order to visualize only what they were interested in:

> "Possibly to be able to skip the animation, and then animate from a certain state onwards. [...] just to test a tiny bit of the code."

None of the students mentioned the usage of two Jeliot 3 features that could be activated from different menus of the tool— the ability to insert a breakpoint, and the ability to call a method different than the main method.

5.3 Lab session observations

From observations during the lab session, we noted that the complete novices all exhibited a common behavior. Students appeared to have two different goals when attending the course. Their main aim is to pragmatically complete the course by solving the assigned tasks, and their secondary goal is to understand how their solution is achieved. It was noted that neither the animation (if they were using Jeliot 3) nor the console output (if they were using the system Java implementation rather than Jeliot 3) were used to understand the basics of programming, unless explicitly taught.

Students found it difficult to understand compound assignments, e.g., `a+=3;`. Some of the students using Jeliot 3 could not decipher the steps taken in its animation. Only after the animation was explained to them, could they make sense of it.

Students also grew tired of the length of the animations, and would run them at full speed. However, at the third lab session, one student from the "voluntary group" identified non-syntactic bugs in his program while visualizing the animation at full speed.

6 Discussion

According to our hypothesis *H1*, students from the "voluntary" group should show a greater vocabulary and understanding than the "normal" students. The two "voluntary" students showed a similar level of detail when describing the animation, but they also showed several misconceptions about the object creation animation, which suggests that the voluntary mini-tasks might not have been of great help. One possible explanation could be that the half hour the "voluntary" students additionally spent working with objects was not enough.

Hypothesis *H2* would be supported if we observed the "voluntary" group exhibiting different behaviour to the "normal" group. In this case, students from both groups were able to use Jeliot to try to understand new concepts and to resolve bugs in their code, and we do not have evidence to support *H2*.

In this small investigation we have not been able to identify any clear distinction between students who received extra education on using Jeliot as a tool to assist in learning programming, and those who used it to learn programming but without any special help. For example, it did not matter if the "voluntary" students had been taught explicitly how Jeliot 3 creates an object, or if the "normal" students had seen the object creation animation many times before. It appears that they did not contrast the knowledge they have, with the information that the animation keeps repeating.

The number of repetitions of the same animation may desensitize students to the importance of the animation itself, and reduce it to a "movie of moving boxes". They were able to follow the boxes, and discover when they have been misplaced, but whilst this is useful to identify bugs, the underlying meaning of the animation may not have been assimilated.

7 Conclusions and Future Work

This investigation reflects on how students have used an educational tool in the context of solving programming assignments in a programming course for undergraduates. Two groups of students were observed using the tool. The first group were given explicit instruction in its use, and required to use it to solve mini-tasks related to the weekly assignment. The other group did not have any help apart from the one they requested during the lab sessions.

From these initial results, it appears that Jeliot 3 animations are hard for novice students to understand. The transfer of knowledge from the tool to the student is not successful. Even students who have been explained explicitly the meaning of the animations have problems understanding or applying this later. Despite this issue, both groups of students found that the help they could get from the animation was useful to debug their programs. Moreover, all the students found Jeliot 3 easy to use, and most the students who started using Jeliot 3 continued to use it.

Results from the investigation are inconclusive, but they spread some more light in why Software Visualization tools are not as widely used in education as expected. In the case of Jeliot 3, we believe that adding verbal explanations to the animation [6] and "stop and think" questions as in JHAVÉ [8] might solve some of the issues found in this investigation.

Mayer's multimedia theories [6] claim that learning can be improved when two different channels are used to convey information. In multimedia, this means that the simultaneous combination of audio and graphics will optimize the cognitive load of the learner. An alternative, and simpler, approach is to combine graphics with textual captions. Such extended animations would provide novices with important information to help them comprehend the secondary notation used in an animation [9], and to build a correct mental model of a program computation.

"Stop and think" questions are those which are woven into the animation. They evaluate student comprehension by asking specific questions about the running algorithm (e.g., which is the next value of the variable stepper?). Such questions have been found to help retain students' attention during an animation and to encourage them to process recently acquired data into meaningful information [8].

Finally, we believe that students get benefits from using Jeliot 3. However, Jeliot 3 is not flexible enough to support the different levels of users' knowledge and cognitive skills (e.g. some students grasp the concepts faster than others), and the different usage patterns (e.g. comprehending or debugging). These three factors would imply the need to model the student, so that content (a combination of animation questions and explanations) could be adapted to the student skills and the task they are performing with Jeliot. Moreover, explanations could help weaker students by providing them with extra and valuable learning material.

Acknowledgments

The authors thank the participants and the staff at the University of Warwick, specially the students at MA117 and their teacher Petr Plechac. We also thank the rest of the Jeliot team and Justus Randolph, who helped us in designing this experiment.

References

[1] Ben-Bassat Levy, R., M. Ben-Ari and P. A. Uronen, *The Jeliot 2000 program animation system*, Computers & Education **40** (2002), pp. 1–15.

[2] Hundhausen, C. D., S. A. Douglas and J. T. Stasko, *A meta-study of algorithm visualization effectiveness*, Journal of Visual Languages and Computing **13** (2002), pp. 259–290.

[3] Kannusmäki, O., A. Moreno, N. Myller and E. Sutinen, *What a novice wants: Students using program visualization in distance programming course*, in: A. Korhonen, editor, *Proceedings of the Third Program Visualization Workshop*, The University of Warwick, UK, 2004, pp. 126–133.

[4] Kehoe, C. M., J. T. Stasko and A. Talor, *Rethinking the evaluation of algorithm animations as learning aids: an observational study*, International Journal of Human Computer Studies **54** (2001), pp. 265–284.
URL citeseer.ist.psu.edu/kehoe99rethinking.html

[5] Korhonen, A., L. Malmi and R. Saikkonen, *Matrix - concept animation and algorithm simulation system*, in: *ITiCSE '01: Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education* (2001), p. 180.

[6] Mayer, R. E., "Multimedia Learning," Cambridge University Press, 2001.

[7] Moreno, A., N. Myller, E. Sutinen and M. Ben-Ari, *Visualizing programs with Jeliot 3.*, in: *Proceedings of the 8th International Working Conference on Advanced Visual Interfaces*, 2004, pp. 373–376.

[8] Naps, T. L., *JHAVÉ – Addressing the need to support algorithm visualization with tools for active engagement*, IEEE Computer Graphics and Applications **25** (2005), pp. 49–55.

[9] Petre, M., *Why looking isn't always seeing: readership skills and graphical programming*, Commun. ACM **38** (1995), pp. 33–44.

[10] Walker, R. J., G. C. Murphy, B. Freeman-Benson, D. Wright, D. Swanson and J. Isaak, *Visualizing dynamic software system information through high-level models*, in: *OOPSLA '98: Proceedings of the 13th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (1998), pp. 271–283.