Algorithm visualization in programming education

Gábor Törley
Corvinus University of Budapest, Hungary, Faculty of Public Administration, Department of Information Technology
gabor.torley@uni-corvinus.hu

Abstract: This paper introduces the theory of algorithm visualization and its education-related results obtained so far, then an algorithm visualization tool is going to be presented as an example, which we will finally evaluate. This article illustrates furthermore how algorithm visualization tools can be used by teachers and students during the teaching and learning process of programming, and equally evaluates teaching and learning methods. Two tools will be introduced: Jeliot and TRAKLA2.

Keywords: programming, education, algorithm, visualization, multimedia

Among the aims of the Hungarian secondary education, the development of students' cognitive skills and the improvement of their thinking gets more emphasis. Everybody stepping in the "Labyrinth of Life" needs conscious thinking to find a quick and efficient answer for everyday problems. Acquiring the ability of *algorithmic thinking* provides also help to reach this goal. [1]

According to Szántó, [1] the most important purposes of algorithmic thinking are the followings:

- The elaboration of conscious, planning behaviour
- The elaboration of self-control
- Evaluation - consciousness

During the phase of planning, the student puts concrete ideas in order (thus elaborates an algorithm), then he/she takes time for reflection on these ideas, for classification of these ideas, and considers the strategy developed. It is important that he/she does not jump to conclusions too early. Finally, he/she evaluates the solution that he/she has found, and is able to see the initial problem in its integrity, as he/she has got to a solution through several efforts, often misdirected. He/she can afterwards recall his/her experiences, and use them during the resolution of future tasks. Like this, algorithms contribute to the development of the cognitive skills of students.

Programming education can take a decisive role in improving the students' cognitive skills by teaching programming theorems (basic algorithms), however, according to the experiences so far – abroad included –, it is rather complicated to learn and teach algorithms. Today's computer science education has an important task to develop the teaching methods in this field. [2, 21, 22]

Teaching of programming theorems is fundamental in teaching programming in secondary school. Through these theorems, the student understands how the specification, the algorithm and the program code work together and how he/she can create more complicated programs by combining these theorems. [21]

According to our experiences as a teacher and as a student, the hardest part of learning programming theorems is when we are searching for the answer to the question why that given algorithm will be suitable and why that given theorem will solve the problem. Obviously, we need to miss verification [23] as a method, because the students in secondary school do not have the required knowledge, furthermore, it would rather deter them. The other reason for skipping pure mathematical tools is that the goal of programming is improving the pupils' cognitive skills.

As practice, we taught programming theorems in a secondary school in Budapest. We demonstrated the mechanism of the given algorithm to the pupils by the help of an animation nested in a presentation, so the pupils saw the animation and heard the explanation at the same time. We could teach them much more than if we have studied only the text of the algorithm. This effect is called "Multimedia effect" by Mayer in Cognitive Theory of Multimedia Learning, which has five principles [3]:

- Multiple representation: It is better to present the explanation in words and pictures than only in words.
- Simultaneity: During explanation, we should present the corresponding text and picture together (at the same time) and not separately.
- Divided attention: Besides the pictorial explanation, we should give ours verbally and not in writing.
- Single differences: The principles above are more important for pupils with lower knowledge level than to those with higher knowledge level.
- Coherency: For example, at summary, let us use as few concepts and pictures as possible which does not belong to the topic.

The theory above, which emphasizes the visual and the pictorial elements, brought positive results in educational environment. [4] WE concluded from our experiences that if we used tools during teaching that help to *imagine* what is happening inside the algorithm, we could promote the process of understanding. Algorithm visualization (AV) tools support this purpose.

1. Algorithm visualization

Algorithm visualization (AV) is the subclass of software visualization and it handles the illustration of high level mechanisms of computer algorithms, usually in order to help the pupils who learn programming to understand better the function of the procedures of the algorithm. [5]

AV has been developed at the end of 1970's from batch-oriented softwares, which allowed instructors to create animation films [6], to today's systems with high-level interaction with which the pupils can explore, configure, change dynamically the animation of the algorithm according to their claim [7, 8] or they can create their own visualization [9, 10]. AV has been used for the following purposes: AV program has supported

- the instructor to illustrate the algorithm, [7]
- the pupils to understand the mechanism of basic algorithms, [11]
- the debugging process at consultation, [12]
- pupils to understand the function of operation of abstract data type. [13]

According to Hundhausen's and his colleagues' summarizing study [5], AV did not spread as a pedagogical tool in the teaching of informatics despite its attractive intuitive features. One of the reasons is, for example, that teachers do not find it efficient. It is true that the published results do not provide a uniform picture so far about the efficiency of AV; it was not proved in each case that the results are better with using AV. The paper analysed the results of several empirical investigations and it highlighted that the most published results were those which supported the cognitive constructivist theory; 71% of those gave significant difference for the group using AV, compared to the results of the control group. In the case of groups in which the task needs presumably more effort, there will be quite a significant difference compared to the control group. In general, the study found the AV technology efficient, but not in the sense that “a picture is worth 1000 words”. Rather the form of learning task is more important than the quality of the visualization. Of course, it does not mean that the quality does not matter. A well-designed visualization supports the successful learning activity. This is why the form of the activity is more important than the form of visualization.

What kind of features can characterise a useful demonstration tool? [2]

- Flexibility
    - Platform-independence: it is not needed to re-write the program for the visualization
    - There is an opportunity to apply different explanation strategies.
    - The users are able to visualize parts of the code
- Program structure, data structure, objects
    - Program structure should be shown
    - It should follow the execution of the program
    - Data structure and the changes of data should be shown
    - Objects and passed parameters should be shown

It is advantageous if the AV software provides full and continuous visualization, so every element (constant, variable, data structure and object) has its visual correspondence, and the AV software should show clearly the connection between the activities and procedures of the program. [14]

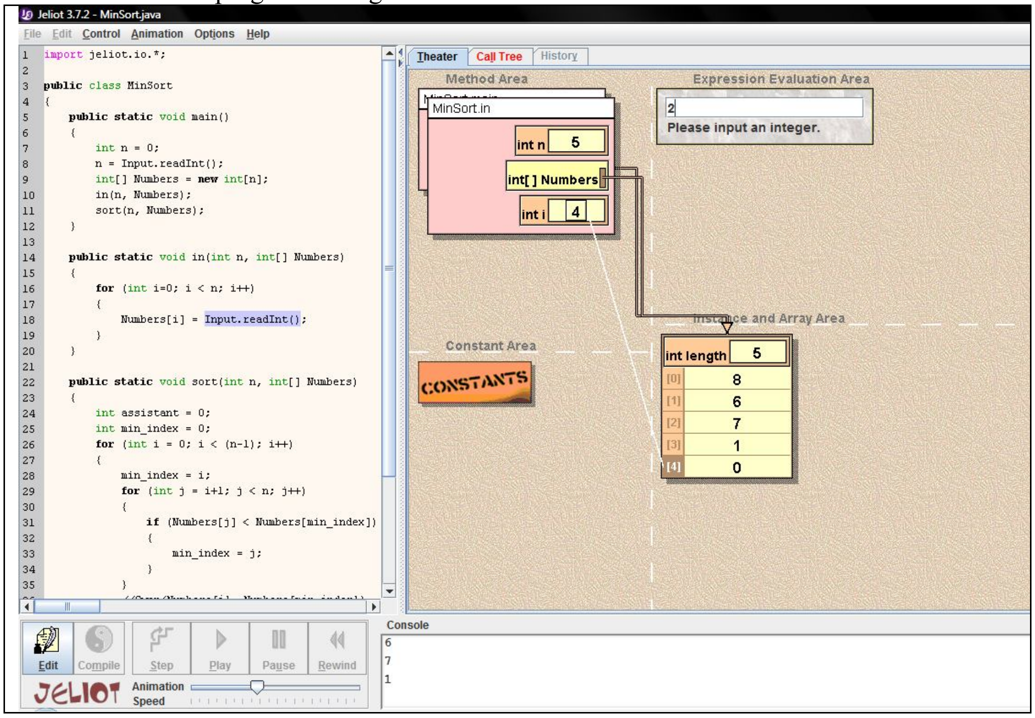
In Kehoe’s and his colleagues’ study [15], they inspected the pupils in homework-like, more real learning situation. This means that they took into account the whole period of the course, and did not judge only based on the result of the exam. Two groups learned on binomial heap; one of them had access to animation during learning. It was helpful in the creation of analogies and concepts that the pupils could watch not only the animation but the code as well at the same time. There were better results at the groups, which used animation, but the authors noted that they made the investigation with pupils with good skills; perhaps they would have had different results with pupils with weaker skills. One of the most striking differences was between the motivations of the two groups. The group that used animation participated whole-heartedly at the lessons; they were much calmer and more confident about their knowledge, they were more open for learning; and in average, they spent more time with the material than the other group. The algorithm animation seems to be the best for helping in teaching the operation of an algorithm step by step, so it can provide a visual representation of an otherwise abstract process.

Using AV tools is almost an unknown strategy in Hungary. There was no paper published in Hungarian or related to Hungarian authors in this topic so far, however, international studies showed many positive results in the past 40 years.

2. Jeliot 3

The Jeliot 3 is a free, Java-based AV system[1]. It has been developed by researchers of University of Joensuu, Finland since 1997. The current version was completed in 2004.

This program-visualization environment is made for beginners in programming. In this system, animations represent step by step execution of JAVA programs (see Figure 1.). Every step of the execution of the program can be seen and the animation simulates how the virtual machine interprets the program code. The place of the animation is the so-called "theatre", which is divided into four parts: the middle and the main part is the "Expression Evaluation" to which messages, method callings, values and references arrive from the other visualization areas. Pupils have the chance to program in Jeliot 3 and they can follow the mechanism of the program through the animation.



```
import jeliot.io.*;

public class MinSort
{
    public static void main()
    {
        int n = 0;
        n = Input.readInt();
        int[] Numbers = new int[n];
        in(n, Numbers);
        sort(n, Numbers);
    }

    public static void in(int n, int[] Numbers)
    {
        for (int i=0; i < n; i++)
        {
            Numbers[i] = Input.readInt();
        }
    }

    public static void sort(int n, int[] Numbers)
    {
        int assistant = 0;
        int min_index = 0;
        for (int i = 0; i < (n-1); i++)
        {
            min_index = i;
            for (int j = i+1; j < n; j++)
            {
                if (Numbers[j] < Numbers[min_index])
                {
                    min_index = j;
                }
            }
```

```
6
7
1
```

Figure 1. *Jeliot*

It is ingenious that the area where the animation happens is called "Theatre". The program (the algorithm) is the scenario; according to that the variables, data structures play their roles and the pupil is the director. So, the animation is part of the programming process and not only part of the learning process. A very favourable feature of the system is that the code and the animation can be seen simultaneously. The environment is able to be animated continuously and step by step, moreover, the pupil can mark the part which he/she wants to watch animated.

[1] http://cs.joensuu.fi/jeliot

For the sake of easier understanding, the obligatory parameter `String[] args` was left from the `main` procedure; this differs from the Java language. The environment accepts lots of elements of Java (one exception for this is the `Scanner` class).

The environment is very useful also for debugging, because every value of the actual variables can be traced and every value of the variables can be modified.

It is an advantage of the system that it is platform-independent; it can be used on Windows, Linux and Mac OS systems.

Jeliot 3 has many reassuring experimental results. The pupils' performance was measured more times during an entire course [14]. It turned out that the system supports frontal teaching, it helps to understand the teacher's explanation and the results of middle-skilled pupils improved the most, especially the way they defined and explained the ideas.

Attention is the first step in the process of learning [16]. Ebel and Ben-Ari did investigations based on this theory. [17] This investigation focused on pupils' attention, because the measurement of attention correlates well with the efficiency of learning. Pupils with behaviour and attention disorder were taught by the help of Jeliot 3; it was experienced that at the part of the lesson when the teacher explained the class material with Jeliot, there was not any behaviour and attention disorder shown by the pupils.

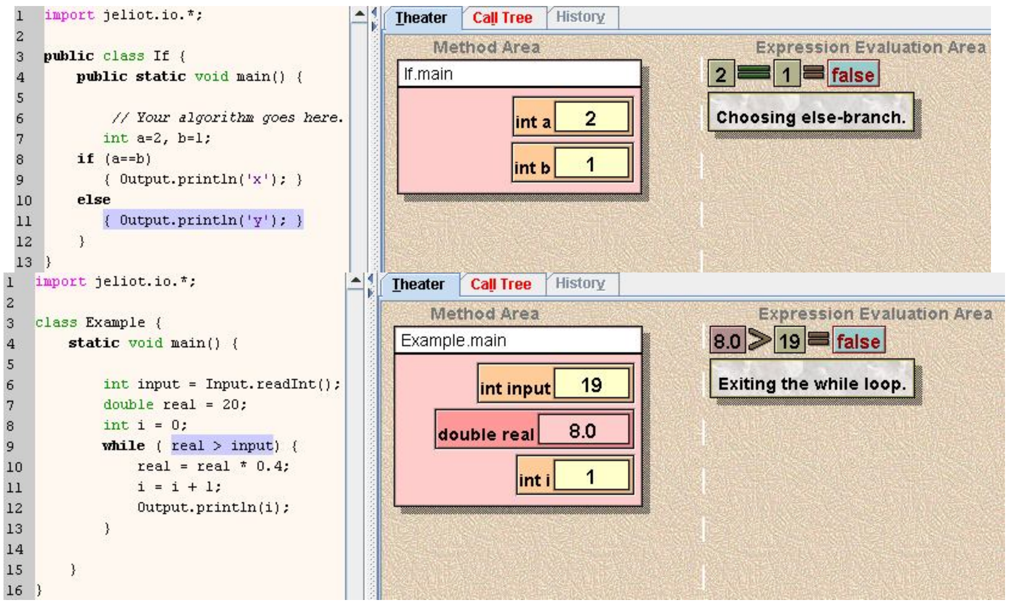
Moreno's and Joy's experiment [18] pointed out the limits of the system: Jeliot 3 is not enough flexible to support students with different knowledge level and different usage patterns. It is true generally that the researchers did not experience significant differences in the performance of pupils below and above the average.

3. Some examples of application

Jeliot 3 is able to represent control structures (branch, loop) and inner mechanisms of programming theorems (basic algorithms), so it supports the explanations of the teacher, as well as self-learning.

The environment's "language-dependence" is not a significant disadvantage; it is obvious, Java language-learners can make the best of the services of the system. The understanding of the mechanism of control structures and algorithms is not a language-dependent task.

The system always reasons why the program runs on the given branch and why we stay in the loop or leave it (see Figure 2.).



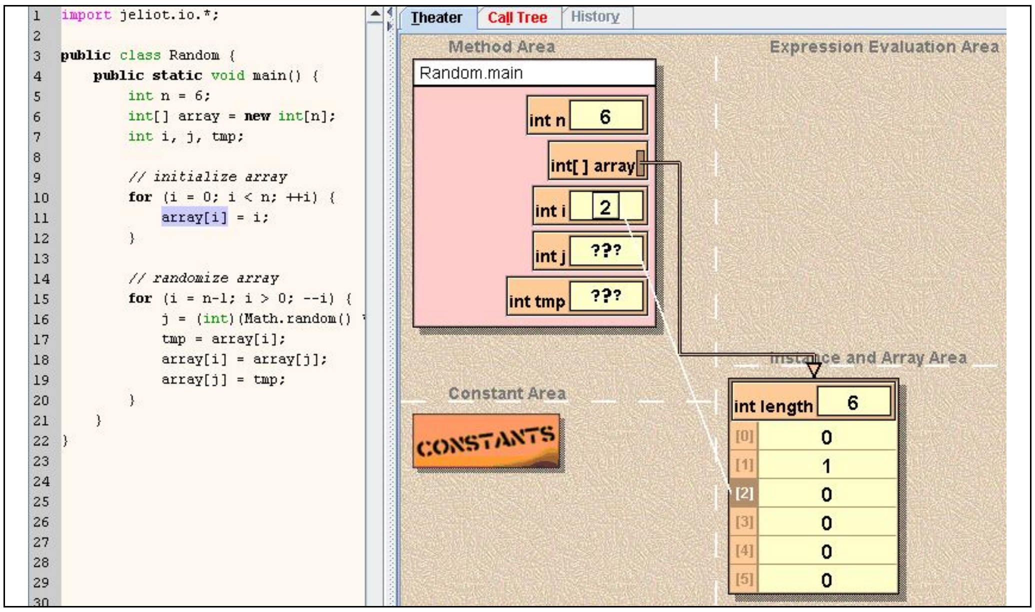
```
import jeliot.io.*;

public class If {
    public static void main() {

        // Your algorithm goes here.
        int a=2, b=1;
    if (a==b)
        { Output.println('x'); }
    else
        { Output.println('y'); }
    }
}
```

Choosing else-branch.

```
import jeliot.io.*;

class Example {
    static void main() {

        int input = Input.readInt();
        double real = 20;
        int i = 0;
        while ( real > input) {
            real = real * 0.4;
            i = i + 1;
            Output.println(i);
        }

    }
}
```

Exiting the while loop.

Figure 2. *Representation of the mechanism of control structures*



```
import jeliot.io.*;

public class Random {
    public static void main() {
        int n = 6;
        int[] array = new int[n];
        int i, j, tmp;

        // initialize array
        for (i = 0; i < n; ++i) {
            array[i] = i;
        }

        // randomize array
        for (i = n-1; i > 0; --i) {
            j = (int)(Math.random()
            tmp = array[i];
            array[i] = array[j];
            array[j] = tmp;
        }
    }
}
```

Figure 3. *Representation of the array*

Array is a frequently used data structure; the environment can represent its inner state (see Figure 3.).

Let us consider a simpler algorithm, linear search:

This specification follows conventions which are applied at Eötvös Loránd Univesrity of Budapest [19, 21]. (Its formal or informal feature is not important; the point is to provide exact information for the solution method.

Specification:

Input: $N \in \mathbf{N}$, $X \in \mathbf{S}^*$, Feature: $S \rightarrow \mathbf{B}$ [$\mathbf{B}$ = {true, false} – Set of Boolean values]

Output: Is_there?$\in \mathbf{B}$, Index$\in \mathbf{N}$

Precondition: Length(X)=$\mathbf{N}$

Postcondition: Is_there? $\equiv \exists i \in [1..N]$:Feature($X_i$) $\wedge$
Is_there?$\Rightarrow$Index$\in [1..N] \wedge$ Feature(X_{Index})

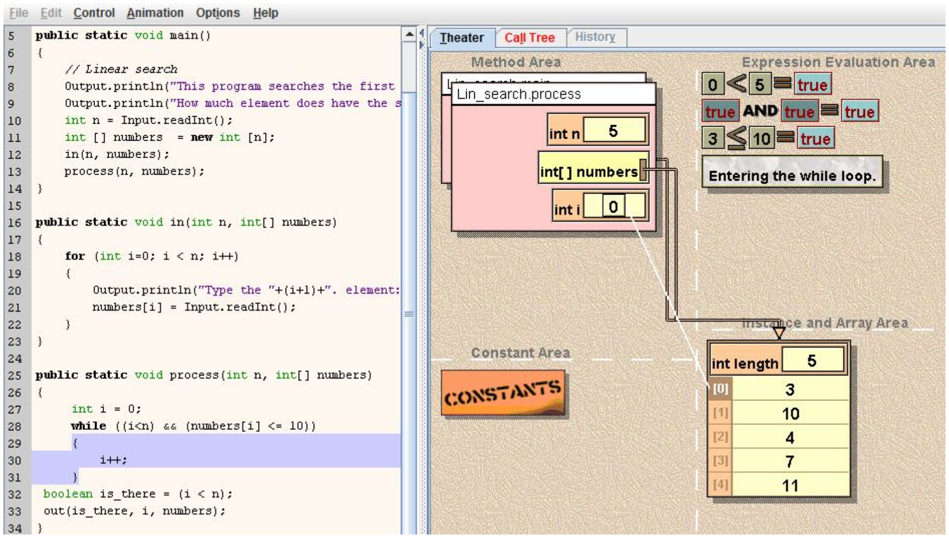
We only present and study the main part of the algorithm. We discuss the read and write procedures in connection with the programming language and environment, because they will be different mostly at that part.

Algorithm:

```
Procedure Linear_search(Constant N: Integer; X: tSs;
Variable Is_there?: Boolean):
    Variable
       I: Integer

    I:=1
    While I≤N and Not Feature(X[I])
       I:=I+1
    End While

    Is_there? = (I≤N)
    If Is_there? then Index:=I
 End Procedure
```

The pupil understood the algorithm if he/she can answer correctly the two following questions: Which condition should be true in order to exit from the loop? Why will the evaluation of the statement I≤N define whether we have found the element with the required attribute or not?

Jeliot 3 helps giving the correct answers to the questions above in two ways:



```
public static void main()
{
    // Linear search
    Output.println("This program searches the first
    Output.println("How much element does have the s
    int n = Input.readInt();
    int [] numbers  = new int [n];
    in(n, numbers);
    process(n, numbers);
}

public static void in(int n, int[] numbers)
{
    for (int i=0; i < n; i++)
    {
        Output.println("Type the "+(i+1)+". element:
        numbers[i] = Input.readInt();
    }
}

public static void process(int n, int[] numbers)
{
     int i = 0;
     while ((i<n) && (numbers[i] <= 10))
     {
         i++;
     }
 boolean is_there = (i < n);
 out(is_there, i, numbers);
}
```

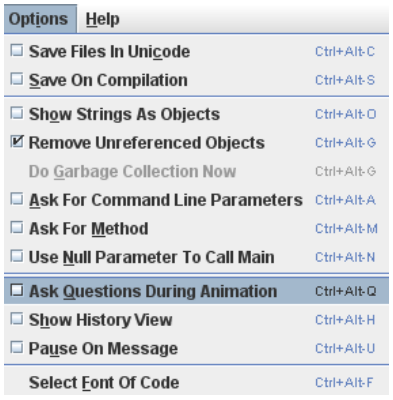
Figure 4. Evaluation of the loop's condition

In the figure above, we can see the solution of the following task, where Linear search should be used: „A sequence of integers is given. Search and write the first number, which is greater than 10. If there is not so, inform the user."

Jeliot 3 evaluates the condition at every branch and loop, and this way it reasons why the program runs the way it runs. Using the step above (see Figure 4), the teacher can point out the role of the loop conditions and he/she can take the thread towards the cases when we will exit from the loop and explain why; and he/she can also explain what it means from the point of view of the task that which condition is false when exiting from the loop.

What does the pupil see? He/she sees the program code and the inner state of the program, so he/she has both textual and pictorial information, in this way the Multimedia-effect is realized. The code can be hidden, if it causes problems, for example if the students do not use Java but a different programming language. In this case, he/she can read his/her own code besides the animation. The inner state of the program tells everything: which array item we are investigating, what are the values of the local variables and in which state the evaluation of the entering condition of the loop is.

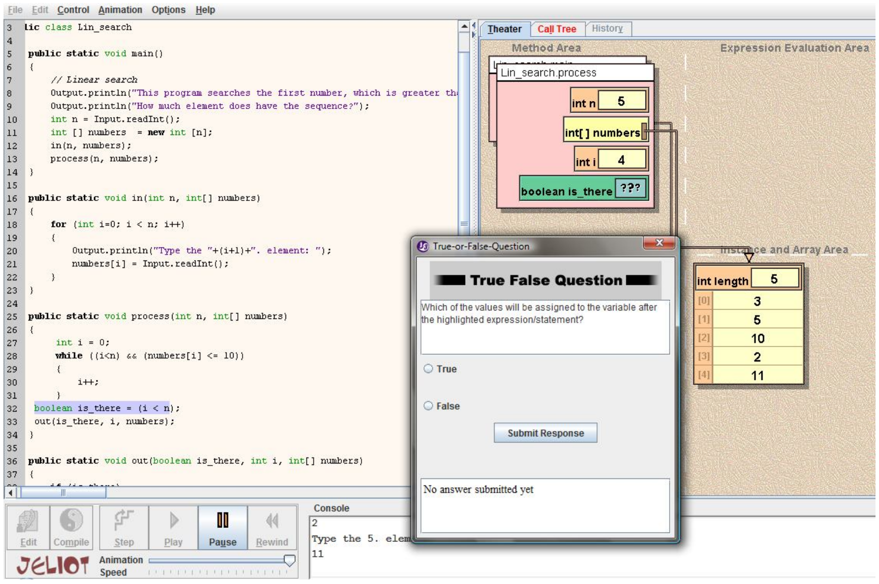
We can set Jeliot 3 to ask questions about the values of variables. You can see this in Figure 5.



- ☐ Save Files In Unicode Ctrl+Alt-C
- ☐ Save On Compilation Ctrl+Alt-S
- ☐ Show Strings As Objects Ctrl+Alt-O
- ☑ Remove Unreferenced Objects Ctrl+Alt-G
- Do Garbage Collection Now Ctrl+Alt-G
- ☐ Ask For Command Line Parameters Ctrl+Alt-A
- ☐ Ask For Method Ctrl+Alt-M
- ☐ Use Null Parameter To Call Main Ctrl+Alt-N
- ☐ Ask Questions During Animation Ctrl+Alt-Q
- ☐ Show History View Ctrl+Alt-H
- ☐ Pause On Message Ctrl+Alt-U
- Select Font Of Code Ctrl+Alt-F

Figure 5. *If we check the checkbox, we get questions during animation*

The questions are too "simple"; unfortunately, they inquire only about the values of variables'/expressions', and about their changes. However, considering our algorithm, the system asks one of the questions at a very good place (see Figure 6), this way, and the environment is useful for home- learning, self-learning and debugging.



```
lic class Lin_search

 public static void main()
 {
     // Linear search
     Output.println("This program searches the first number, which is greater th
     Output.println("How much element does have the sequence?");
     int n = Input.readInt();
     int [] numbers  = new int [n];
     in(n, numbers);
     process(n, numbers);
 }

 public static void in(int n, int[] numbers)
 {
     for (int i=0; i < n; i++)
     {
         Output.println("Type the "+(i+1)+". element: ");
         numbers[i] = Input.readInt();
     }
 }

 public static void process(int n, int[] numbers)
 {
      int i = 0;
      while ((i<n) && (numbers[i] <= 10))
      {
          i++;
      }
  boolean is_there = (i < n);
  out(is_there, i, numbers);
 }

 public static void out(boolean is_there, int i, int[] numbers)
 {
```

Which of the values will be assigned to the variable after the highlighted expression/statement?

No answer submitted yet

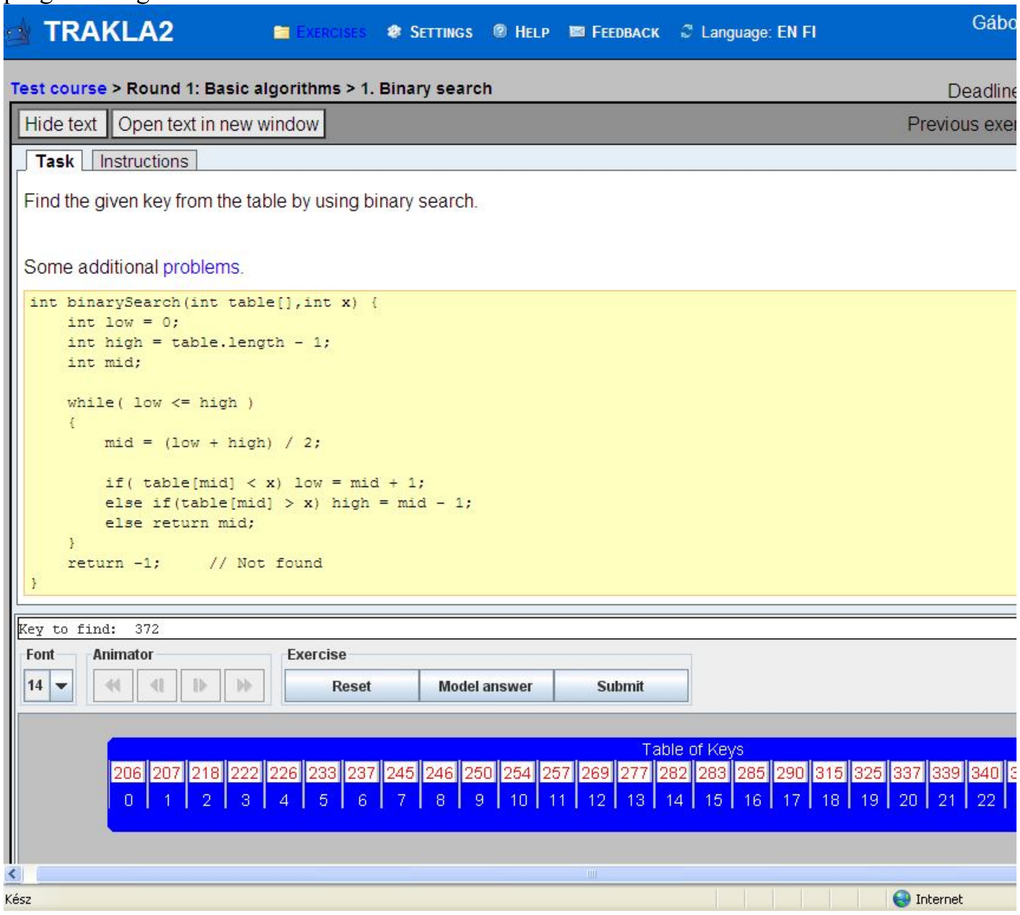
```
2
Type the 5. elem
11
```

Figure 6. *If we understand the inner state correctly, we can have the right answer*

It can be interesting to ask how disturbing is the fact that, obviously, the environment has been planned for teaching with Java programming language. It may be disturbing a bit. At former inquiries [20], pupils used Turbo Pascal, and according to this study, at basic concepts, for example at assigning values, the syntactic differences were not high. The teacher could always help to solve these kinds of problems.

4. TRAKLA2

TRAKLA2 algorithm visualization (AV) system follows a completely different strategy and method. It was developed in order to support first grade university students in their programming studies.



Find the given key from the table by using binary search.

Some additional problems.

```
int binarySearch(int table[],int x) {
    int low = 0;
    int high = table.length - 1;
    int mid;

    while( low <= high )
    {
        mid = (low + high) / 2;

        if( table[mid] < x) low = mid + 1;
        else if(table[mid] > x) high = mid - 1;
        else return mid;
    }
    return -1;     // Not found
}
```

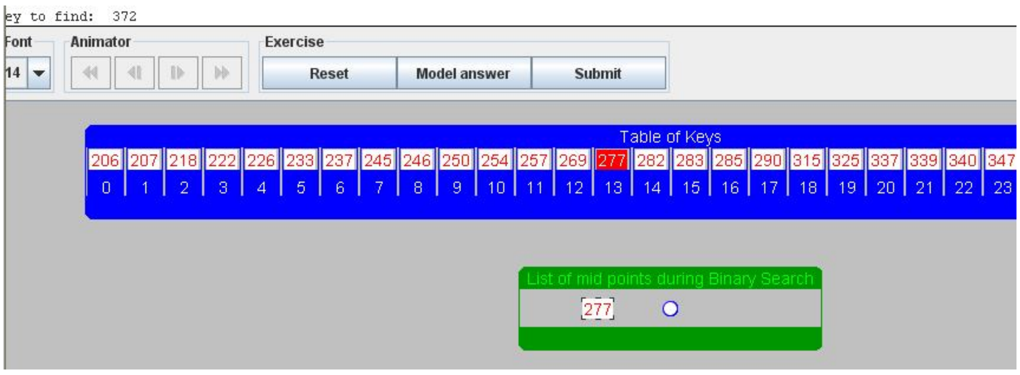
Key to find: 372

206	207	218	222	226	233	237	245	246	250	254	257	269	277	282	283	285	290	315	325	337	339	340
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Figure 7. TRAKLA2

This system does not use animations; or, more precisely, it uses animations as the student himself/herself should "invent" the animation based on the algorithm which the system evaluates. As an example here, we will use the binary or logarithmic search.

Like Jeliot 3, this system also supports both the teacher's lesson and he student's self-learning. In practice, the teacher and the student both play the algorithm by putting the middle elements in the green "box" (see Figure 8).



```
ey to find:  372
```

Table of Keys

206	207	218	222	226	233	237	245	246	250	254	257	269	277	282	283	285	290	315	325	337	339	340	347
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

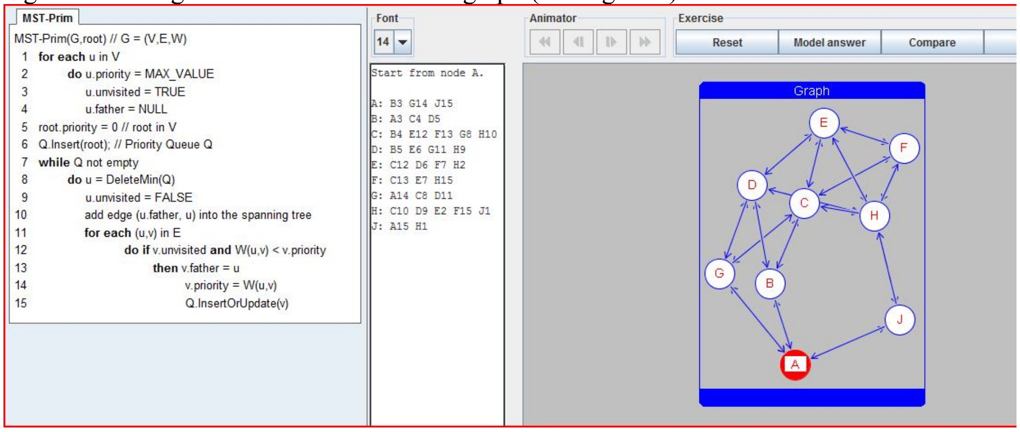
Figure 8. During the completion of the task

After we have given the answers, the system shows us the result, and it gives opportunity for us to review the steps of our solution. It can show us the correct solution step by step, if we have not found it yet.

This system is useful for teachers for giving homeworks, and thus, the teacher can check the results, so he/she is able to follow the students' performance.

TRAKLA2 supports self-practising, self-learning at home in a good way, because the student can "play" the mechanism of the algorithm through more than one input, and after completing the task, he/she is immediately informed about the efficiency. A great advantage of the environment is that the student practises the pure algorithm (and its mechanism) itself and thus, his/her knowledge will not depend on the programming language. This statement is true despite the fact that the algorithmic language is very similar to Java programming language. Since TRAKLA2 is freeware and it is able to be developed freely, after the Hungarian localization, it can be a useful tool in the higher education in Hungary, for example at the class named "Algorithms and Data Structures"; and expanded by basic algorithms; it can be a useful tool for secondary education as well.

The following figures show well that the environment indicates not only the mechanism of the algorithm but the representation of the data structure as well (in case of binary search: array). This way, for example in case of a graph-algorithm, the student/teacher can "play" the algorithm through the data structure of the graph (see Figure 9).



```
MST-Prim(G,root) // G = (V,E,W)
for each u in V
    do u.priority = MAX_VALUE
       u.unvisited = TRUE
       u.father = NULL
root.priority = 0 // root in V
Q.Insert(root); // Priority Queue Q
while Q not empty
    do u = DeleteMin(Q)
       u.unvisited = FALSE
       add edge (u.father, u) into the spanning tree
       for each (u,v) in E
           do if v.unvisited and W(u,v) < v.priority
                then v.father = u
                     v.priority = W(u,v)
                     Q.InsertOrUpdate(v)
```

```
Start from node A.

A: B3 G14 J15
B: A3 C4 D5
C: B4 E12 F13 G8 H10
D: B5 E6 G11 H9
E: C12 D6 F7 H2
F: C13 E7 H15
G: A14 C8 D11
H: C10 D9 E2 F15 J1
J: A15 H1
```

Figure 9. *An example for graph-algorithm*

5. Summary

The Jeliot 3 AV program has the features which define an efficiently-using program-visualization system.

This program supports frontal teaching when it completes the teacher's oral explanations with pictorial information; and at individual learning, it helps to understand the program code with visual elements so it fulfills well the Mayer's theory. [4] The system has a function that asks questions about the value of variables when it is used during the animation. Unfortunately, it does not ask other types of questions, but these questions can test how the pupil understands the mechanism of the program and they support debugging.

It is possible to insert *one* breakpoint in *one time* in the code from which the animation runs so the pupil can animate only that part what he/she has chosen. The system represents well the inner mechanism of data structures and the procedures, and their local variables and parameters.

Great advantage of both programs is that they support the teacher in theoretical and practical education as well, and that the pupil can use them as a helpful tool when he/she solves the task by himself/herself.

TRAKLA2 supports active learning, especially because the student is not only a passive beholder but he/she "plays" the animation himself/herself, this way, the system affirms the final correct mechanism of the algorithm through incorrect solutions.

There is not any result in Hungary or any paper in Hungarian about the usage of AV in education. We need further inquiry to answer the question: what methods and principles are needed in order to introduce AV into the Hungarian secondary education. The international results are mixed, but there are promising ones which show that with time, a tool can be provided to teachers and students which will improve the efficiency of learning and the motivation of the students.

References

[1] Sándor Számtó: *Improving algorithmic thinking in elementary school.* New Pedagogical Review, 2002/05, in Hungarian

[2] Matti Lattu, Veijo Meisalo, Jorma Tarhio, *A visualisation tool as a demonstration aid*, Computers & Education, v.41 n.2, p.133-148, September 2003

[3] Mayer, R. E. (1997). *Multimedia learning: Are we asking the right questions.* Educational Psychologist, 32, 1-19.

[4] Mayer, R. E. & Moreno, R. (1998, April). *A Cognitive Theory of Multimedia Learning: Implications for Design Principles*. Paper presented at the annual meeting of the ACM SIGCHI Conference on Human Factors in Computing Systems, Los Angeles, CA.

[5] C. Hundhausen, S. A. Douglas, and J. T. Stasko. *A meta-study of algorithm visualization effectiveness.* Journal of Visual Languages and Computing, 2002.

[6] R. Baecker (1975) T*wo systems which produce animated representations of the execution of computer programs.* SIGCSE Bulletin 7, 158-167.

[7] M. H. Brown (1988) *Algorithm Animation*. The MIT Press, Cambridge, MA.

[8] J. T. Stasko (1990) *TANGO: a framework and system for algorithm animation*. IEEE Computer 23, 27-39.

[9] J.T. Stasko (1997) *Using student-built animations as learning aids*. In: Proceedings of the ACM Technical Symposiumon Computer Science Education. ACM Press, New York, pp. 25^29.

[10]C. D. Hundhausen (1999) *Toward effective algorithm visualization artifacts: designing for participation and communication in an undergraduate algorithms course*. Unpublished Ph.D. dissertation, Department of Computer and Information Science, University of Oregon.

[11]P. Gloor (1998) *Animated algorithms*. In: SoftwareVisualization: Programming as a Multimedia Experience (M. Brown, J. Domingue, B. Price&J. Stasko, eds) TheMIT Press, Cambridge, MA, pp. 409-416.

[12]J. S.Gurka,&W. Citrin (1996) *Testing effectiveness of algorithm animation*. In: Proceedings of the 1996 IEEE Symposium onVisual Languages. IEEE Computer SocietyP ress, Los Alamitos, CA, pp. 182-189.

[13]T. Naps (1990) *Algorithm visualization in computer science laboratories*. In: Proceedings ofthe 21st SIGCSE Technical Symposium on Computer Science Education. ACM Press, New York, pp. 105-110.

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

[15]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.

[16]J. D. Kindlon. *The measurement of attention*. Child Psychology & Psychiatry Review, 3(2):72–78, 1998.

[17]G. Ebel, M. Ben-Ari. *The affective effects of program visualization*, ICER'06, September 9–10, 2006, Canterbury, United Kingdom

[18]Moreno, A. and M. Joy, *Jeliot 3 in a Demanding Educational Setting*, in: Proceedings of the Fourth International Program Visualization Workshop, Florence, Italy, 2006, pp. 48–53.

[19]Péter Szlávi, László Zsakó: *Methodical programming: Programming Theorems.* Eötvös Loránd University, Faculty of Science, Department group of Informatics, Budapest, Hungary (1996., in Hungarian)

[20]Ronit Ben-Bassat Levy, M. Ben-Ari, Pekka A. Uronen: *An Extended Experiment with Jeliot 2000.* First International Program Visualization Workshop, Porvoo, Finland, 2000.

[21]Péter Szlávi: *Didactical questions of creating programs.* (Ph. D. thesis) Eötvös Loránd University, Budapest, Hungary (2004., in Hungarian)

[22]Péter Szlávi: *Creating programs and thinking.* Informatics in higher education 2008., Debrecen, Hungary (Conference publication in Hungarian)

[23]Péter Szlávi: *Programs, program specifications*, In: Informatics in higher education'99, pp. 576-582, Debrecen (1999., in Hungarian)