ACTA DIDACTICA NAPOCENSIA

Volume 7, Number 1, 2014

ALGORITHM VISUALIZATION IN TEACHING PRACTICE

Gábor Törley

Abstract: This paper presents the history of algorithm visualization (AV), highlighting teaching-methodology aspects. A combined, two-group pedagogical experiment will be presented as well, which measured the efficiency and the impact on the abstract thinking of AV. According to the results, students, who learned with AV, performed better in the experiment.

Key words: programming, education, algorithm visualization, multimedia, algorithm, algorithmic thinking, abstraction

1. Introduction

Among the aims of the Hungarian secondary education, the development of students' cognitive skills and the improvement of their thinking attracts more and more emphasis. Stepping into the "Labyrinth of Life" needs conscious thinking in order to find a quick and efficient answer for everyday problems. Acquiring the ability of *algorithmic thinking* also provides help in reaching this goal. [23]

According to Szántó, [23] the most important purposes of algorithmic thinking are the following:

- The elaboration of the conscious and planning behaviour
- The elaboration of self-control
- Evaluation - consciousness

During the planning phase, the student puts concrete ideas in order (thus elaborates an algorithm), then they take time to reflect on these ideas, to classify these ideas, and to consider the strategy developed. It is important that they do not jump to conclusions too early. Finally, they evaluate the solution that they have found, and are able to see the initial problem in its entirety, as they have gotten to a solution via numerous, often misdirected efforts. Afterwards, these experiences can be used in future tasks. This way, algorithms contribute to the development of the cognitive skills of students.

Educational programming can take a decisive role in improving the students' cognitive skills by teaching programming theorems (basic algorithms); however, based on past experiences – abroad included [11, 28] –, it is rather complicated to learn and teach algorithms. Today's computer science education has an important task in developing teaching methods in this field. [25, 26]

Based on my experience as a teacher, the hardest part of learning programming theorems is searching for the answer to the question why a given algorithm is suitable and why a given theorem solves the problem. Therefore proving if a given algorithm is correct or not –at this level, without using mathematical tools – is difficult, because the pupils need to understand the steps of the algorithm and the correspondence among these steps. Another problem is, provided that the student understands the theorem of the algorithm, how to create the right algorithm for a particular task, and then how to write a well-functioning code. So we can conclude that this in one problem: how can the student apply and *specify* the *abstract* theorem to solve the task.

During my teacher internship, I taught programming theorems in a secondary school in Budapest, Hungary. I demonstrated the mechanism of the given algorithm to the pupils with the help of an animation nested in a presentation. The pupils saw the animation and heard the explanation at the

Received 1 March 2014.

same time. They were able to learn much more this way than had they only saw the text of the algorithm, because I was able to get the material through to them using more channels and dimensions. Beyond the *visual* and *audial* channel, I used the dimension of *time* as well, so the pupils could watch the change of the algorithm in real time. This effect is called the "Multimedia effect" by Mayer in Cognitive Theory of Multimedia Learning, which has five principles [13]:

- 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, has brought positive results in educational environment. [12] I concluded from my experience that if I used tools during teaching that help *imagining* what is happening inside the algorithm, I could improve the process of understanding. Algorithm visualization (AV) tools support this purpose.

One of the strengths of AV could be involving more sensory organs in learning. Kátai et al studies [8, 10] confirmed that if a teaching method impacts on different sensory organs, it can effectively support the teaching and learning of algorithms. Furthermore, AV tools can be used to develop algorithmic thinking not only in computer science students. [9]

2. The history of algorithm visualization

Algorithm visualization (AV) is a subclass of software visualization and it handles the illustration of high level mechanisms of computer algorithms, usually in order to help the pupils understand the function of the procedures of the algorithm better. [7]

It is hard to present an ever-changing process with static tools like texts and pictures in a textbook. This is true for a "classical" presentation as well. However, it is possible to create animation in a presentation, but it needs a lot of work, and it is not possible to reuse the animation for a different task. In teaching practice, when a teacher tries to illustrate a process at the blackboard, he or she executes the algorithm documenting the changes of the algorithm's most typical data (for example variables, arrays, etc.) at every step. Even with this support, it is hard for most of the pupils to understand every detail of the algorithm. That is why foreign teachers have begun to focus on visualization. With it, it is easier to present the dynamism of an algorithm. [5]

AV has been developed since the end of 1970's from batch-oriented software. This software at first allowed instructors to create animation films [1], and it developed into a system with high-level interaction. With it pupils can explore, configure and change the animation of the algorithm dynamically so it fits their needs [4, 21].They can also create their own visualization [6, 22]. Below, I review this historical process briefly.

In the middle of the 1990s, the application of the Internet and Java programming language spread more and more. At the end of the 1990s and at the beginning of the next millennium, a new generation of AV tools have been created. Examples for these systems are: JSamba [I9], JAWAA [18], JHAVÉ [16], ANIMAL [19], and TRAKLA2. [I12] They have one common feature: they were created to give a fully-equipped toolbox for building AV, so they follow the traditions of the pre-Java systems. In the "Java-era", the development process was feasible for those who were ready to invest some work into the development of a spectacular system. JAWAA was a refreshing example of this among this kind of

software, because it was easy for teachers to create visualization, but the system did not support interactivity.

The biggest transition happened simultaneously with the application of Java. It was brought by those AV systems that were unconstrained by both the development system of the AV and the operating system. These systems developed and executed visualization in different systems. This resulted in the change of the developer's role. As a result, projects with separate AV packages were created, not including a development system. Some examples: Data Structure Navigator (DSN) [I3], Interactive Data Structure Visualization (IDSV) [I6], Algorithms in Action [I10], Data Structure Visualization [I4] and TRAKLA2. [I7]

Another impact of using Java was that lecturers and students both began developing AV systems. Many years' persevering work resulted in Java applets being created, which displayed specific topics, such as different searches in trees, spatial indexing, etc. Examples include Binary Treesome [I5], JFLAP [I8], University of Maryland Spatial Index Demos [I1] and a system created by the University of Virginia Algorithm Visualization Study Group (2011). [I11]

In their study, Naps et al expanded [17] the conclusions of Hundhausen et al [7] i.e. a student has to be as active as they can be during visualisation. A taxonomy was defined which determines the level and type of the student's activity. Six levels were stated:

1. *No viewing*: in this cases AV is not used at all.
2. *Viewing*: students only look at the execution and the steps of AV.
3. *Responding*: students are presented with questions during the visualization.
4. *Changing*: students can change data during the visualization.
5. *Constructing*: students constructs the algorithm's visualization themselves.
6. *Presenting*: students explains the algorithm using visualization and asks for feedback and discussion.

This taxonomy relates to Bloom's hierarchy. [3]

Urquiza–Fuentes and Velázquez–Iturbide [27] conducted a meta-study on the pedagogical value of AV systems. The paper focused on the areas where education benefitted from AV systems. This study reconfirmed the results of Hundhausen et al (2002) (and it contradicted the taxonomy above), which states that level 2. does not improve students' results. Unfortunately, many systems are only on this level today. [21]

The meta-study above also concluded that the higher level the student is able to apply the AV system, the more beneficial the AV system will be for learning.

Myller, Bednarick, Sutina, and Ben–Ari conducted a study [15] on the impacts of activity in a collaborative learning environment. They extended the aforementioned taxonomy, in order to define the students' behaviour better. Four new levels were added to the taxonomy, two of them are mentioned here: *controlled viewing* and *entering input*. These two levels are between levels 2. and 3. During *controlled viewing* the students control the visualization. E.g. they can stop it, or move it one step back, or change its speed. At the *entering input* level, the students are able to input data before or during the running of the visualization. This is only possible at the points where the program asks for data.

Myller et al proved [15] their theory with experiments: they worked in pairs with students using BlueJ [20] and Jeliot 2000 [2] systems. Students' communication was examined and recorded during the experiment. The conclusion was that the extended taxonomy correlates well with the measure of interaction. Interaction is one of the key elements of computer aided teamwork. [14]

3. Presentation of the experiment, evaluation and analysis of its results, experiences

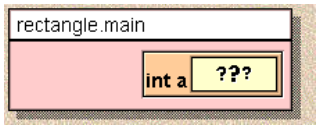
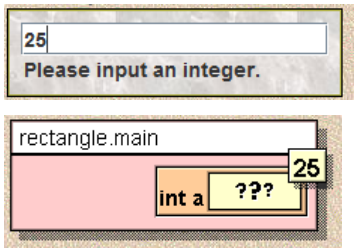
During the autumn semester of the academic year 2012/2013, I performed a combined, two-group pedagogical experiment at ELTE Faculty of Informatics. I investigated the work of 1st grade informatics students in the "Basics of programming" course. 5 seminar groups took part as the experimental group (78 students) and 5 seminar groups as the control group (75 studetns), 153 students altogether.

3.1. Introduction to the "Basics of programming" course's syllabus

This course is a grounding subject during the first grade. According to my survey, half of the students has no knowledge of programming before entering the course; that is why the syllabus starts from the very beginning. The goal of this course is to raise students to the same level and to systematize and formalize the expected programming knowledge from secondary school. Students use the Code::Blocks [I2] IDE*, and they code their tasks on C++ language.

3.1.1. Programming environment, I/O operations, declaration

During the first topic, students get to know the IDE, the input and output commands and the concept of declaration. They have to understand that principle that data (a variable) can only be used (i.e. giving a value to it, counting with it) if and only if it is declared first. With the support of the Jeliot system, this sequence can be showed visually. After the "/" sign I wrote the Java command as it appears in Jeliot.

Code	Visual representation
`int a;`	
`cin >> a; / a = Input.readInt();`	

This animation above could help the student in understanding that the variable has to be created first, then a value needs to be given to it before the variables could be used.

3.1.2. Control structures

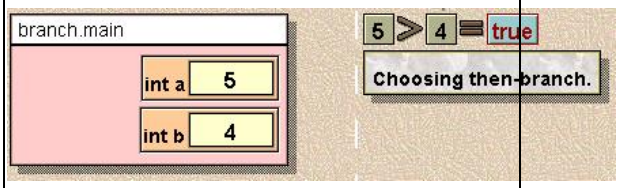
During the introduction of control structures, the first topic shows the given control structure, and answers what kind of process runs when it is executed and why it is executed so. The supporting animation helps understand the control structure.

Second topic: branches. This is the first control structure because it is the simplest to understand and its practical advantages – e.g. checking a precondition – can be used soon.

Students understand the operation of branches if they know which branch is to be executed (with knowledge of the concrete values, of course).

* Integrated Development Environment

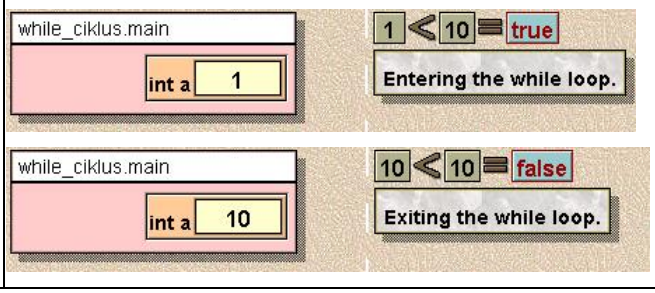
Code ♣	Visual representation

`if (a>b)`
`  {`
`   cout << "a greater↵`
`  than b";`
` / Output.println("a greater↵`
`  than b");`
`  }`
`else`
`  {`
`   cout << "a is not greater ↵`
`  than b";`
` / Output.println("a not↵`
`  greater than b");`
`  }`
	

Jeliot prints the result of the logical examination of the condition, then the answer as well, i.e. which branch executes the program according to the result of the examination. The reason for the program running on a particular branch is unambiguous for the students.

Third topic: loops. The concept of loops is more difficult to understand than the concept of branches, because loops are processes stretching over time. Students understand the operation of loops when they are able to 1) interpret the loop's algorithm and determine when the program does not execute the core of the loop, or when they are able to 2) write the algorithm and the code of a self-described loop.

Code	Visual representation

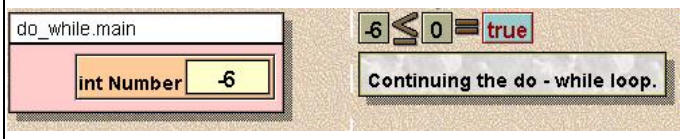
`while (a<10)`
`{`
`  ++a;`
`}`
	

The AV program's text explanation gives the students the answer as to why the program enters into or exits from the loop.

During do-while loops, Jeliot writes the *continuation* of the loop when the condition is true, so the operation and the role of this kind of loop can be differentiated unequivocally from the while loop.

♣ Due to editing reasons, the line wraps were signalised with a ↵ sign. The commands, which are broken into more lines this way, are actually in the same line.

Code	Visual representation

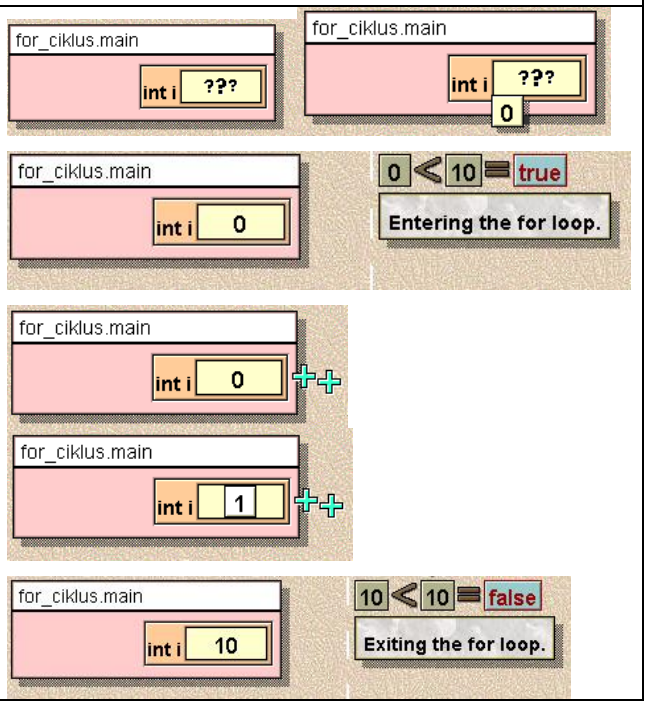
do
{
cout << "Give a↵
positive number!")
/ Output.println("↵
Give a positive↵
number!");
cin >> Number;
/ Number = Input.readInt();
if (Number <=0)
{
cout << ("Error,↵
give a positive↵
number, please!";
/ Output.println("Error↵
give a positive ↵
number, please!");
}
}while (Number <=0);
	

Understanding the for loops are simpler after the while loops have been understood. At this point, Jeliot supports the understanding of the role of the loop's expressions.

The steps can be easily followed on the animation below: variable i is created, then it gets a value, then the entering condition is checked and evaluated, then the loop-core executes, and finally, the loop variable (i) is incremented.

Understanding control structures is essential to use programming theorems solidly.

Code	Visual representation

for (int i=0; i<10;
++i)
{
cout << i;
/ Output.println(i);
}
	

3.1.3. Using arrays

The next topic deals with arrays. It is not difficult to understand the definition and the goal of an array, but indexing the first element with 0, the second element with 1 and so on can be complicated to understand, because students who have experience not based on C-based languages struggle with the indexing.

Jeliot gives support with the visual representation and the debugging. Visual representation supports displaying the actual state of the array, so the programmer can follow the changes, which helps later in exploring errors. The AV program is also able to show how the operation of indexing works (see Figure 1).

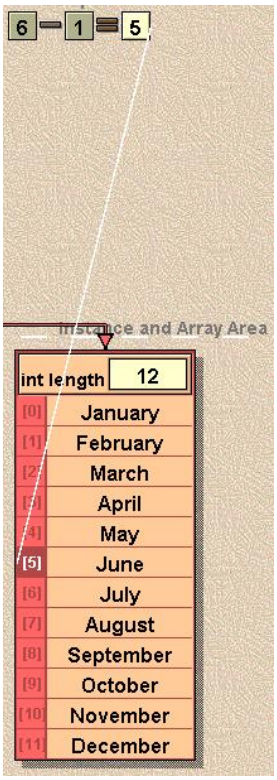


Figure 1. *Jeliot 3: The actual state of an array and operation of indexing*

3.1.4. Programming theorems

The last big topic of the semester deals with programming theorems. Let us consider a simpler algorithm as an introduction: linear search. According to the conventions which are applied at Eötvös Loránd University of Budapest [24, 25]:

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

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

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

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

This kind of convention builds well upon prior secondary school knowledge and improves it by "forcing" students to think the precondition through and to document. (Actually, the concept of a precondition is not new: checking the domain of a function, which is a part of the secondary school syllabus, is the same. Thus, this mathematical thinking needs to be applied and reviewed by the student.)

The linear search specified above, according to the "philosophy" of the algorithm below, searches for the element with *F* feature in an array, provided that it exists within that feature (see Figure 2).

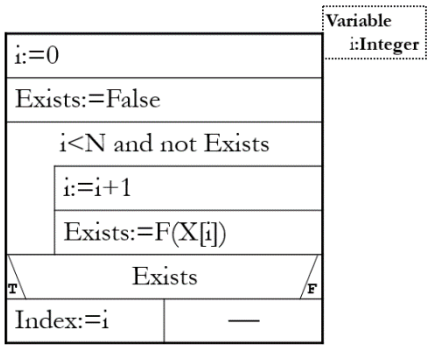


Figure 2. *Solution of linear search with structogram*

Let us consider a different algorithm, which solves the task above (see Figure 3):

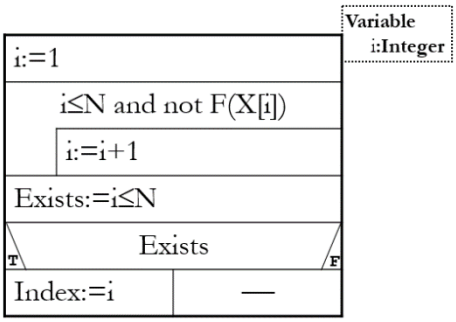


Figure 3. *A different solution of linear search*

Why is it worth to create more variants of an abstract algorithm for a programming theorem? Because with them the students can practice the interpretation of the algorithm, the parallel examining of the two algorithms can support the improvement of abstract thinking, and it makes the student perform a non-formal verification.

The pupils understand the algorithm if they can answer the two following questions correctly: 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, as Figure 4. shows.

In the figure below, we can see the solution for the following task, where Linear search needs to be used: „A sequence of integers is given. Find and write down the first number that is greater than 10. If there is no such number, inform the user."

As we saw before, Jeliot 3 evaluates the condition at every branch and loop, and this way it gives a reason why the program runs the way it runs. Using the step below (see Figure 4), the teacher can point out the role of the loop conditions and they can further take the thread towards the cases where we exit from the loop and explain why. They can also explain why it's important for the solving of the task to know which condition being false exited our loop.

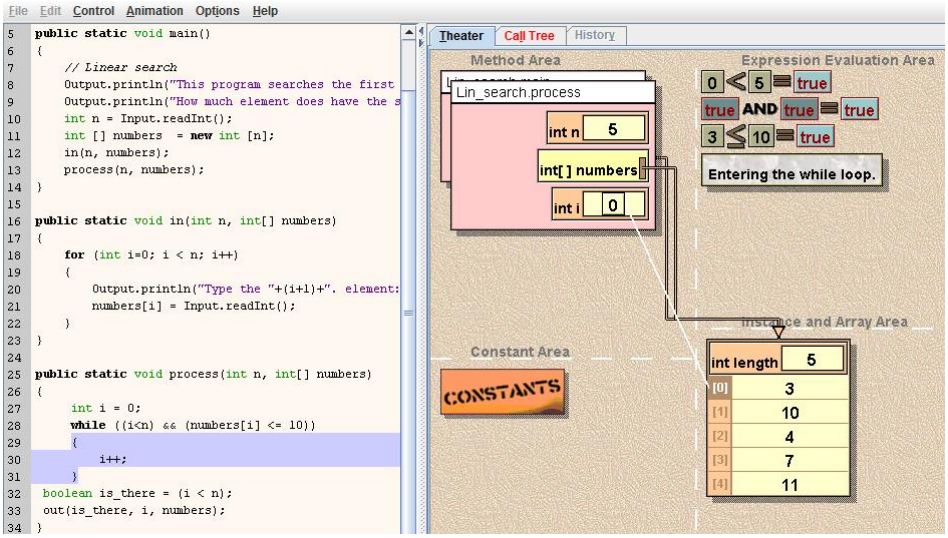


Figure 4. *Evaluation of the loop's condition*

The inner state of the program tells everything (see Figure 4): 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 in. On the Basics of programming course we use C++ programming language, but the difference between C++ and Java language is not big. It can only be detected with input-output operations, but the coding of data structures and the main part of the algorithm are literally the same.

We can set Jeliot 3 to ask questions about the values of variables, as displayed in Figure 5.

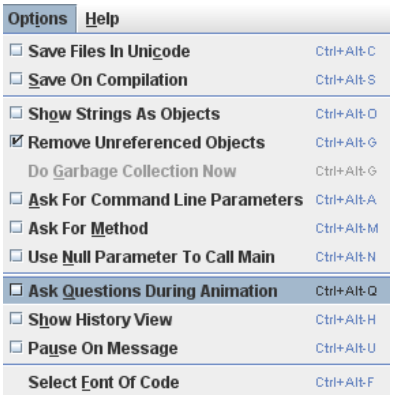


Figure 5. *If we check the checkbox, we get questions during animation*

Unfortunately, the questions are too "simple"; they only inquire about the values of variables'/expressions' and their changes. However, considering our algorithm, one of the tasks deals with a key factor of understanding the algorithm (see Figure 6); the environment is useful for at-home learning, self-learning and debugging.

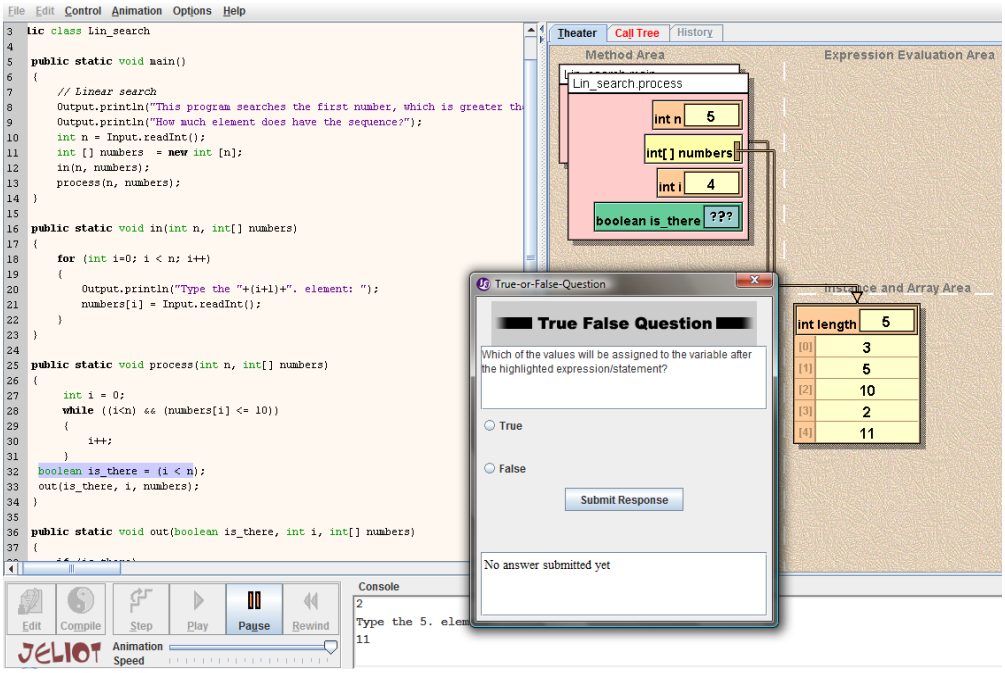


Figure 6. *If we understand the inner state correctly, we can get the right answer*

The support provided by the AV can be used the same way regarding the other programming theorems.

3.2. Method of the experiment

The first part of the experiment was a questionnaire on previous studies which was filled out by the experimental and control groups (this questionnaire can be found in the appendix of this paper). The goal of the questionnaire was to quantify how many lessons did the students have on informatics and programming in secondary schools, and to measure the state of algorithm thinking of the students before the course would impact on it.

During the experiment, the students got a score and a grade on questions no. 3-5 about algorithm thinking; this way the students' progress can be measured. Question no. 5 measured their ability on abstraction. During the questionnaire's evaluation, I investigate the correctness and the elaboration of the answers. Elaboration is important because with it the multiplicity of the student's thinking can be measured: how much attention the student pays to the details of the task. The score was higher if the student gave more important preconditions. Maximum score for questions no. 3-5 was 14 points.

In order to define the progress, the evaluation should be converted to a five grade scale, because the latter tests are in a five grade system also. The scoring system was the following:

bottom point limit	grade	bottom point limit's ratio
0	1	0%
5.5	2	39%
7.5	3	54%
10	4	71%
12	5	86%
14	(maximum)	100%

After defining the grades of the input tasks, we have a starting point from which we can determine the measure of their progress. During the semester, the students had four tests:

- Seminar group test no. 1: two coding tasks which can be solved by simple programing theorems. The students get the specifications and the algorithms for the tasks (goal of the test: mainly, understanding the algorithm + coding knowledge),
- Seminar group test no. 2: writing algorithms with a structogram for three tasks which can be solved by simple programing theorems. The students get the specification only (goal of the test: to understand the specification + to create algorithms, this test needs more abstraction ability than the previous one),
- Submitting the home-exercise: the students have to solve 4 tasks, one of them can be solved by complex programming theorems, the other 3 by simple ones (goal of the exercise: to demand a solution of a larger task, „product-construction"),
- Semester closing test: the students have to solve 4 tasks, 2 of them can be solved by complex programming theorems, the other 2 by simple ones (complete solution of a bigger problem).

When I was defining the measure of progress I didn't involve the home-exercise because there is little guarantee of the independent work of the students so the authenticity could be questioned. I weighted the other 3 tests according to time, because the further we are (in time) from the first lesson, the deeper should be the student's understanding. I defined the measurement of progress (*P*) with the following formula:

$$P = [(SGT_1 - PS) + 1{,}5 * (SGT_2 - PS) + 2 * (SCT - PS)]/18,$$

where SGT_1 is the grade of seminar group test no. 1, SGT_2 is the grade of seminar group test no. 2, *SCT* is the grade of the semester closing test, and *PS* is the grade of the questionnaire on their prior studies (questions no. 3-5). The result of the formula above is a real number between -1 and 1. The mathematical sign shows the direction of the progress, the value shows the measure.

The results of questions no. 3-5 were similar, I did not experience significant difference (comparing these values with Ordinary Least Squares (OLS) test, p=0.84531, ceteris paribus[†]). This way, it can be stated, that the two groups started from similar levels so the differences of the progress values are not caused be the input knowledge.

3.3. Results of the experiment

The table below shows that how big an impact does teaching with AV system has on the students' progress and whether the control group or the experimental group has developed more.

Prior studies points[♣]	**Count**		**%**		**Average progress**	
	Experimental group	**Control group**	**Experimental group**	**Cont. group**	**Experimental group**	**Cont. group**
0	2 pcs.	0 pcs.	3%	0%	0.07	-
1	3 pcs.	0 pcs.	4%	0%	0.71	-
2	3 pcs.	4 pcs.	4%	5%	0.45	0.67

[†] ***Ceteris paribus*** or ***caeteris paribus*** is a Latin phrase, literally translated as "with other things the same," or "all other things being equal or held constant."

[♣] Grade x means points in the interval [x,x+1), because prior studies points are real numbers.

Prior studies points♣	Count		%		Average progress	
	Experimental group	Control group	Experimental group	Cont. group	Experimental group	Cont. group
3	***6 pcs.***	***7 pcs.***	***8%***	***9%***	***0.60***	***0.60***
4	***14 pcs.***	***16 pcs.***	***18%***	***21%***	***0.79***	***0.37***
5	***14 pcs.***	***7 pcs.***	***18%***	***9%***	***0.56***	***0.00***
6	***12 pcs.***	***23 pcs.***	***15%***	***31%***	***0.51***	***0.26***
7	***6 pcs.***	***4 pcs.***	***8%***	***5%***	***0.20***	***0.03***
8	***9 pcs.***	***10 pcs.***	***12%***	***13%***	***0.07***	***0.03***
9	5 pcs.	1 pcs.	6%	1%	-0.09	0.50
10	0 pcs.	0 pcs.	0%	0%	-	-
11	3 pcs.	2 pcs.	4%	3%	-0.36	0.14
12	0 pcs.	1 pcs.	0%	1%	-	-1.00
13	1 pcs.	0 pcs.	1%	0%	-0.75	-
14	0 pcs.	0 pcs.	0%	0%	-	-

The great majority of the students (84%) got their score in interval [3,9) in the prior study questionnaire (highlighted in the table above). According to the results, 78% of the experimental group and 89% of the control group are in this interval. These two diagrams below show the measure of the students' progress. The second diagram shows the highlighted interval [3,9):

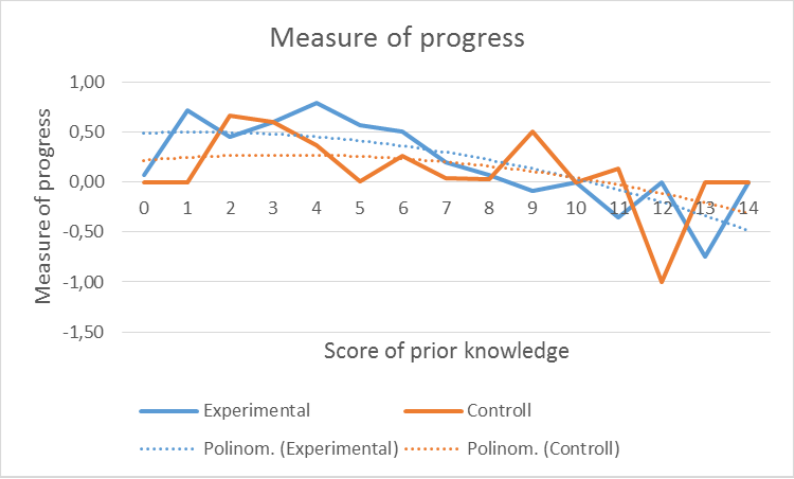


Figure 7. *Measure of progress.*

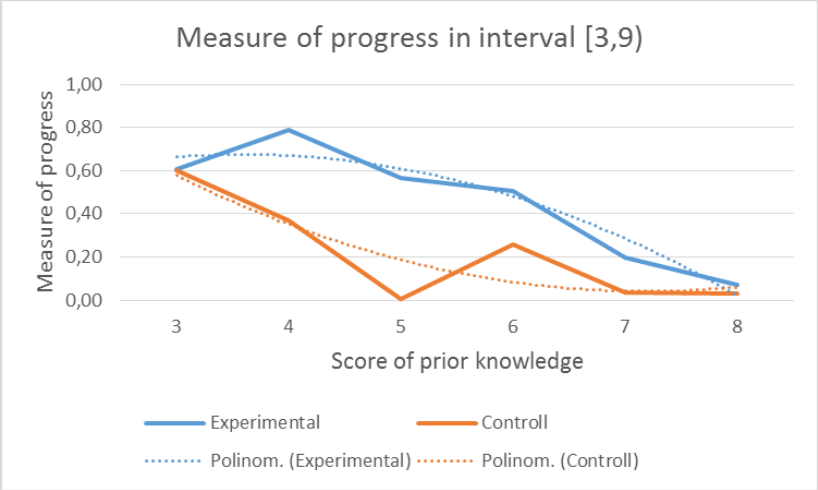


Figure 8. *Measure of progress in interval [3,9).*

The conclusion of the diagrams (see Figure 7 and 8) above is that 84% of the students who participated in the experiment and used AV during their studies, have progressed significantly greater than the students in the control group (using the OLS test, *coefficient*=0.274152, p=0.0002<0.05, ceteris paribus). Teaching with the AV system supported the nearly average students the most and the difference between the two groups was significant, favoring the experimental group (using the OLS test, *coefficient*=0.281085, p=0.00312<0.05, ceteris paribus). Based on the experiment, the average students has their score in interval [4,7), because the average score of the participants in the experiment was 5.74. The greatest progress (0.79) was reached by the students of the experimental group with their score being in interval [4,5).

This diagram shows the comparison of the results at the end of the semester (see Figure 9).

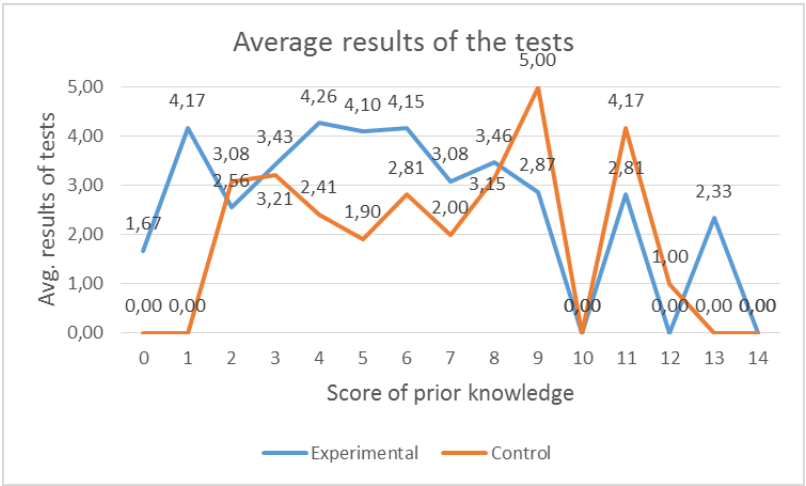


Figure 9. *Average results of the tests.*

I took notice of the test's average grade, because it shows the progress of the students better than only the term mark itself.

In interval [3,9) (which contains the great majority of students, see Figure 10) there was a significant difference favouring the experimental group (using the OLS test, *coefficient*=0.926838, p=0.00008<0.05, ceteris paribus). This result is related to progress as well, so teaching with AV tools has a positive impact on students who has an average prior knowledge.

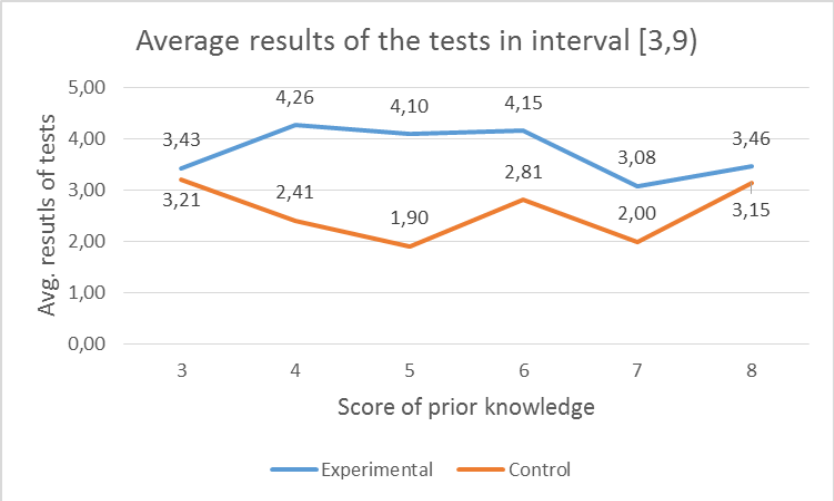


Figure 10. *Average results of the tests in interval [3,9).*

The table shows that the experimental group's average grade is almost one grade better than of the control group's.

	Experimental group	Control group
Average of tests	3.58	2.65

Learning with AV programs has an impact on the failure ratio as well because significantly less students have failed the course in the experimental group than in the control group. The two third of the experimental group reached the 3.5 average tests grade or above (only 37% in the control group).

The impact of AV on abstraction ability shows the results of question no. 5 and seminar group test no. 2. The score of question no. 5 showed where this ability was at the beginning of the semester, and the grade of seminar group test no. 2 showed if this ability has progressed or not.

	Experimental group	Control group
Avg. result of question nr. 5.	1.60	1.67
Avg. result of seminar group test nr. 2.	4.13	3.18

According to the table above, the experimental group starts somewhat more behind than the control group, but the former reached a much better result than the control group. This means that the student's abstraction ability improves better with AV tools than without (using the OLS test, *coefficient*=0.967692, p=0.00035<0.05, ceteris paribus).

4. Summary

I introduced the history of AV from the 70's till the present, and I highlighted the development of AV's role in programming education.

I added the Jeliot 3 AV system to the syllabus of Basics of programming course. I highlighted which parts of the syllabus supported by the students' learning and understanding with the AV system.

During the autumn of 2012, I performed a combined, two-group pedagogical experiment at ELTE Faculty of Informatics, where I investigated the works of 1st grader informatics students in the course: Basics of programming. Both the experimental and control groups have consisted of 5-5 seminar groups.

During the first class, students filled a questionnaire on their prior knowledge in programming. Two questions were about how many classes they had in secondary school on informatics and

programming. Three other questions contained simple algorithmic tasks, with which I measured the students' level of algorithmic thinking.

This input test gave the basis on measuring how much progress did the students make, during the semester.

The experimental group performed better, according to the measure of progress and the final results of the course. This was especially true to the students with close-to average abilities. This is why, it is proven that it is worthy to use AV tools in programming teaching, especially for novice students.

References

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

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

[3] Bloom, B. S., & Krathwohl, D. R. (1956). *Taxonomy of educational objectives: The classification of educational goals*. Handbook I: Cognitive domain. Harlow, England: Longmans.

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

[5] Fouh, E., Akbar, M. & A. Shaffer, C. (2012): *The Role of Visualization in Computer Science Education*, Computers in the Schools, 29:1-2, 95-117

[6] Hundhausen, C. D. (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.

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

[8] Kátai, Z., Juhász, K., Adorjáni, A., K., *On the role of senses in education*, Computers & Education (2008), Vol. 51, No 4, 1707-1717, ISSN: 0360-1315.

[9] Kátai, Z., Kovács, L. I., Kása, Z., Márton, Gy., Fogarasi, K., Fogarasi, F.: *Cultivating algorithmic thinking: an important issue for both technical and HUMAN sciences*, Teaching Mathematics and Computer Science, 9 (2011) 1, 1-10.

[10] Kátai, Z., Toth L., *Technologically and artistically enhanced multi-sensory computer programming education*, Teaching and teacher education 26 (2010), 244-251.

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

[12] 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.

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

[14] Meier, A., Spada, H., & Rummel, N. (2007). *A rating scheme for assessing the quality of computer-supported collaboration processes.* International Journal of Computer Supported Collaborative Learning, 2(1), 63–86.

[15] Myller, N., Bednarik, R., Sutinen, E., & Ben–Ari, M. (2009). *Extending the engagement taxonomy: Software visualization and collaborative learning.* Transactions on Computing Education, 9(1), 1–27.

[16] Naps, T. L., Eagan, J. R., & Norton, L. L. (2000). *JHAVÉ—An environment to actively engage students in Web-based algorithm visualizations*. SIGCSE Bulletin, 32, 109–113.

[17] Naps, T. L., Rossling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C. D., Velazquez–Iturbide, J. (2003). *Exploring the role of visualization and engagement in computer science education.* SIGCSE Bulletin, 35(2), 131– 152.

[18] Pierson, W. C., & Rodger, S. H. (1998). *Web-based animation of data structures using JAWAA.* SIGCSE Bulletin, 30, 267–271.

[19] Rossling, G., Schuler, M., & Freisleben, B. (2000). *The ANIMAL algorithm animation* tool. In J. Tarhio, S. Fincher, & D. Joyce (Eds.),Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education (pp. 37–40), Helsinki, Finland.

[20] Sanders, D., Heeler, P., & Spradling, C. (2001). *Introduction to BlueJ: A Java developmentenvironment.* Journal of Computing Sciences in Colleges, 16(3), 257–258.

[21] Shaffer, C. A., Cooper, M., Alon, A., Akbar, M., Stewart, M., Ponce, S., & Edwards, S. H. (2010). *Algorithm visualization: The state of the field.* ACM Transactions on Computing Education, 10, 1–22.

[22] Stasko, J. T. (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.

[23] Szántó, S.: *Improving algorithmic thinking in elementary school.* New Pedagogical Review, 2002/05, in Hungarian

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

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

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

[27] Urquizza–Fuentes, J., & Velazquez–Iturbide, J. (2009). *A survey of successful evalu- ations of program visualization and algorithm animation system.* ACM Transac- tions on Computing Education, 9(2), 1–21.

[28] Urquiza-Fuentes, J., and Velázquez-Iturbide, J. (2013): *Toward the effective use of educational program animations: The roles of student's engagement and topic complexity*, Computers & Education, vol. 67, pp. 178 - 192

References from the Internet

[I1] Brabec, F., & Samet, H. (2003). *Maryland spatial index demos Web page.* http://donar.umiacs.umd.edu/quadtree - Downloaded: June 18. 2013.

[I2] Code Blocks page. http://www.codeblocks.org – Downloaded: July 23. 2013.

[I3] Dittrich, J.-P., van den Bercken, J., Schafer, T., & Klein, M. (2001). DSN: *Data structure navigator.* http://dbs.mathematik.uni-marburg.de/research/projects/dsn/index.html.bak - Downloaded: June 18. 2013.

[I4] Galles, D. (2006). *Data structure visualization.* http://www.cs.usfca.edu/~galles/visualization - Downloaded: June 18. 2013.

[I5] Gustafson, B. E., Kjensli, J., & Vold, J. M. (2011). *Binary treesome Web page.* http://www.iu.hio.no/~ulfu/AlgDat/applet/binarytreesome – Downloaded: June 18. 2013.

[I6] Jarc, D. J. (1999). *Interactive data structure visualization.* http://nova.umuc.edu/~jarc/idsv - Downloaded: June 18. 2013.

[I7] Korhonen, A., Malmi, L., Silvasti, P., Nikander, J., Tenhunen, P., Mård, P., Karavirta, V. (2003). *TRAKLA2.* http://www.cse.hut.fi/en/research/SVG/TRAKLA2 - Downloaded: June 17. 2013.

[I8] Rodger, S.H. (2008). *JFLAP Web page.* http://www.jflap.org – Downloaded: June 18. 2013.

[I9] Stasko, J. T. (1998). *JSamba.* http://www.cc.gatech.edu/gvu/ii/softvis/algoanim/jsamba - Downloaded: June 17. 2013.

[I10] Stern, L. (2001). *Algorithms in action.* http://ww2.cs.mu.oz.au/aia - Downloaded: June 18. 2013.

[I11] *Virginia Tech Algorithm Visualization Research Group Web page.* (2011). http://research.cs.vt.edu/AVresearch - Downloaded: June 18. 2013.

Appendix

Prior studies survey

1. How much classes did you have in secondary school on informatics per week?
   - 0
   - 1
   - 2
   - 3
   - 4
2. How much classes did you have in secondary school on programming altogether?
   - 0
   - 1-4
   - 5-8
   - 9-14
   - 15 or more
3. Write an algorithm (with your own words) on how you cross a traffic light intersection!
4. Write an algorithm (with your own words) on how a coffee machine should be used!
5. Write an algorithm (with your own words) for the following task: Think of a positive integer and read it! In case it'sn not a positive integer, signalize the error and exit the program. After reading the correct number, write the read number on the screen!

Author

Gábor Törley, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary, e-mail: pezsgo@inf.elte.hu