20/1 (2022), 51–66
DOI: 10.5485/TMCS.2022.0536
tmcs@science.unideb.hu
http://tmcs.math.unideb.hu

Teaching
Mathematics and
Computer Science

A case study of the integration of Algorithm Visualizations in Hungarian programming education

Imre Bende

Abstract. In this study, I will introduce how Algorithm Visualizations (AV) can help programming education or, in this case, the acquisition of basic programming theorems. I used two different methods to test this: in the first round, I examined in a larger group how much the students' ability to solve specific tasks changes after being introduced to a visualization tool, and then, what was their motivation and experience during this process. In the second round, I looked for the components that could be important when choosing a tool with the help of an in-depth interview with a smaller number of individuals. In both cases, I describe the research, experience, and results of the study, and then summarize them at the end.

Key words and phrases: algorithm visualization, education, IT.

MSC Subject Classification: 97P10.

Introduction

While many factors can affect the process of understanding and multiple studies have already measured the effectiveness of Algorithm Visualization (AV) (as collected in (Hundhausen et al., 2002)), the main purpose of my test was not the examination of effectiveness, but to find out how popular the method is in the Hungarian educational system, and how the pupils would react on using a specific tool during their university studies. Efficiency in programming education could mean that students can solve more tasks in the same amount of time, they use the learned algorithms for specific problemsolving tasks, altogether they have deeper understanding of the algorithm.

I created a website[1] which contains visualizations of programming theorems. The purpose was to implement a tool which helps teaching and understanding these algorithms, specifically in a Programming course at Eötvös Loránd University (ELTE). It contains pseudocode, specification, description, C++ implementation and a visualization that is step-by-step playable and the input of which is modifiable.

In the first section of my research, I wanted to test the aforementioned Algorithm Visualization: to check if it can be used as an educational tool at a university in the first semester (Q1), while getting additional background information about the students (Q2).

In the second section, my goal was to find out what are the most important elements in a visualization for specific students (Q3) and what extra features they think would be useful for the learning process (Q4). In the past, I already collected and made a component tree based on that (Bende, 2019b), therefore my goal was to validate or expand this graph based on the results.

Related works

Naps' engagement taxonomy (Naps et al., 2003) defines five levels of interaction with Algorithm Visualization: viewing, responding, changing, constructing, presenting. Myller extended this theory with new levels, in this way, the extended taxonomy includes the following engagement levels: no viewing, viewing, controlled viewing, entering input, responding, changing, modifying, constructing, presenting, reviewing (Myller et al., 2009). Many studies tried to prove that higher engagement level means more effective learning process (Hundhausen et al., 2002). Karavirta provides a review of these studies in his thesis (Karavirta, 2009), and a more recent article also found positive results related to this topic (Simonak, 2020). Beside interactive elements, when we use a "good" visualization tool, "the use of the visualization system decreases students' negative emotions, and students experience more positive than negative emotions" (Lacave et al., 2020). This ultimately gives the students positive feedback, emotions, motiveness, therefore the usage of the tool becomes an effective learning method. On the other hand, the design and the use of instructions are as important as the

[1]Website of AlgTan: https://ermi46.web.elte.hu/dev/algotan/

tool itself, as the two together really grabs the students' attention and helps them learn the ways of algorithmic view and programming (Byrne et al., 1996) (Landers, 2015). As we see, there are many factors that can affect the process of learning, therefore, the main focus of my research is not the efficiency in general, but to observe the rate of learning success with the previously mentioned tool, AlgTan.

While we mostly think about an application or website on PC when we speak about visualisations, the mobile platform has now become another important field, which indicates many questions about implementing, using Algorithm Visualization on these devices (Supli et al., 2016).

Algorithm Visualization is not just a tool for learning algorithms, but can be used as a debugger tool as well, which informs users of the status of the program and helps them understand its functionality (for example, PVC.js is visualizing C programs on web browsers (Ishizue et al., 2020)). In programming education, visualizations also help in understanding data structures and their operations, which has the same importance as the algorithms on these (as an experiment showed us by using DS-PITON (Nathasya et al., 2019)).

There are many studies and many more tools (Schaffer in 2010 for his study already counted and observed 500 (Shaffer et al., 2010)) for Algorithm Visualization, yet, as we will see later from the results of the questionnaire, not many students have used or met these tools during their education in Hungary.

Test of an Algorithm Visualization

Test subjects

I conducted the survey among the first-year computer science students of the Faculty of Informatics of Eötvös Loránd University within the course of Programming. I determined the date of the examination so that they knew the basic structures of programming, they could already use them, but at that time, more serious programming theorems had not yet been introduced for independent problem solving at the university.

Process of test

As the first step, I created an assessment to measure the prior knowledge of the students. Prerequisites may vary, as it is possible that they have already learned programming in high school, or they have not even met with it, but there is one thing in common: the programming theorems have already been presented at the university, but they have not yet been used to solve problems. I examined the background with three tasks that required each programming item. The tasks were as follows:

(1) In a paper collection campaign, everyone recorded how many pounds of paper they brought. Write a program that gives you a sum of how much paper the participants collected!

(2) In a competition, the date of birth of each participant was recorded. Create a program that will determine if the competition had any minor (younger than 18) racer!

(3) In a poker tournament, each participant is recorded as a number, which represents when they were eliminated. Use a program to determine which player was eliminated first!

I only provided a shorter period of time for solving the tasks (20 minutes) not only to examine whether the students would be able to do so, but also to see how quickly and efficiently they would do it. This short timeframe also allows greater scaling. Of course, I pointed out that the goal is not necessarily to solve the tasks completely, but to see how much they manage to solve during this time in the current situation (thus relieving the tension caused by a certain level of accountability).

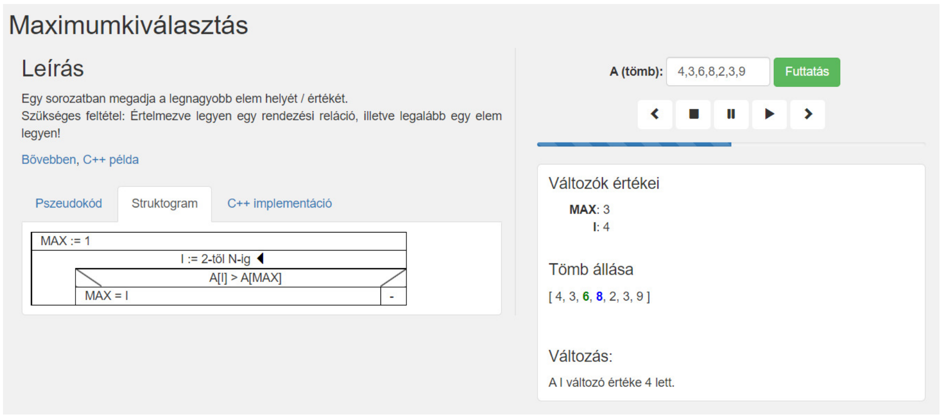
Subsequently, the basic programming theorems (series calculation, counting, maximum selection, decision, selection, linear search) are presented. Here, with the support of the Algorithm Visualization I developed (Figure 1), I presented the purpose, description and step-by-step process of the algorithms. For each one, we also looked at a simpler example application written in C++.

After that, in order to check the improvement, I gave them three more tasks, similar to the ones they did at the beginning of the lesson, with the same criteria. The tasks were as follows:

(1) A group of N friends (N is a constant value) recorded how much they spent each day on a vacation. Create a program that determines the number of days, when more than €100 was spent per person!

(2) On each day of a festival, it was recorded how many tickets were sold on each day. Create a program that determines the days with the most visitors!

(3) Given a series of temperatures, determine if it is strictly monotonically increasing!


Maximumkiválasztás

Leírás

Egy sorozatban megadja a legnagyobb elem helyét / értékét.
Szükséges feltétel: Értelmezve legyen egy rendezési reláció, illetve legalább egy elem legyen!

Bővebben, C++ példa

Pszeudokód | Struktogram | C++ implementáció

```
MAX := 1
I := 2-től N-ig
    A[I] > A[MAX]
    MAX = I | -
```

A (tömb): 4,3,6,8,2,3,9 Futtatás

Változók értékei
MAX: 3
I: 4

Tömb állása
[4, 3, 6, 8, 2, 3, 9]

Változás:
A I változó értéke 4 lett.

Figure 1. Used Algorithm Visualization (AlgTan)

I evaluated the solution of the tasks based on three factors: whether

(1) the program is correct (10 points),

(2) the code has good style (2 points),

(3) the student chose and used the correct programming theorem (2 points).

In the third part, I only checked the output, while during the deeper examination, I looked at the code purity, the use of variable names, and the appropriate indents.

At the very end, I asked the students some additional questions in order to get a better understanding of their impressions on the tools introduced during the test. The questions can be divided into two groups:

(1) Preliminary fact-finding:
    (a) Did you learn to program during your high school years? If so, how long?
    (b) Have you ever encountered an Algorithm Visualization during your high school years? If so, which one?

(2) Gathering feedback:
    (a) If you have ever encountered Algorithm Visualization, how much has it helped you to master the algorithms?
    (b) How much did the visualizations seen in the class helped you to understand the algorithms?

(c) What other components and tools can you name that could help with programming education?

(d) Other comments

About the tasks

The first set of assignments consisted of three basic tasks, which can be solved with one or other of the programming theorems within an array (these are summation, decision, selection). In the second part, however, it was necessary to make minor modifications on them (as they were somewhat more complex, more difficult tasks, which did not require simply entering the basic algorithm/program provided by the programming theorem):

- Task 1: counting item, but the condition contained a constant value.
- Task 2: maximum selection.
- Task 3: decision, but compared to the original algorithm, it was worth negating the condition of the logical variable, that is starting from the assumption that the statement is true (strictly monotonically increasing), until we find an adjacent element pair that refutes the property.

Results

The first significant information is how many students have already studied programming before their university studies (Figure 2). More than three-quarters of the students had prior programming education, and all of them indicated that they had studied it at least weekly for more than a year.

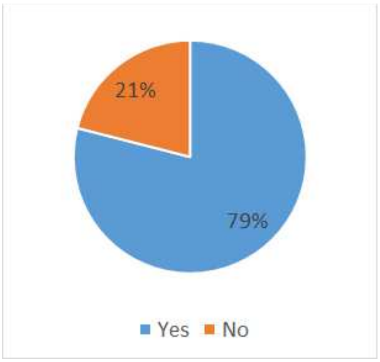


Figure 2. Proportion of students who studied programming prior to the test

Surprisingly, even though many students had already learned programming, none of them had yet encountered Algorithm Visualization or a similar type of tool. There are two possible explanations for this result: either they were not aware that they had used or seen it before (even as a game), or the learning of algorithms did not appear as a separate step in programming education, but only as a tool (which can easily happen due to the low number of tutorial classes for programming in schools).

After evaluating the solutions, I examined how much the average total score of the students changed (Figure 3).

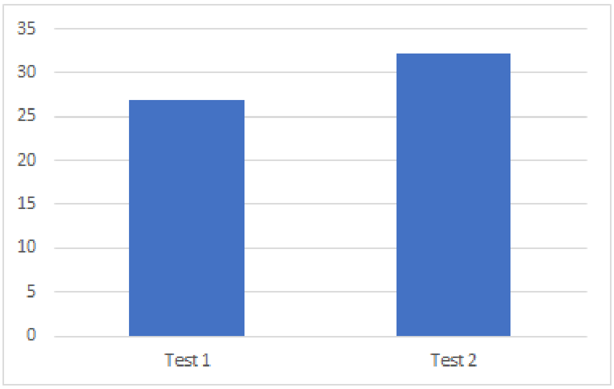


Figure 3. Average total student score for each test (the maximum was 3 tasks * (10 + 2 + 2) = 42 points)

The proportion of students who scored less, equal, or more points in the second round is shown in Figure 4. A significant achievement was that nine students had a rate of development of at least ten points. This actually means that these students managed to complete an extra task in the same amount of time.

After completing the survey, I also asked the students how useful and helpful the Algorithm Visualization was in the learning process (Figure 5). This could be indicated on a scale of 1 to 6 (1 - not at all, 6 - completely). The answers to this question show that with a few exceptions, everyone considered the understanding or learning of the algorithm by visualization to be more positive.

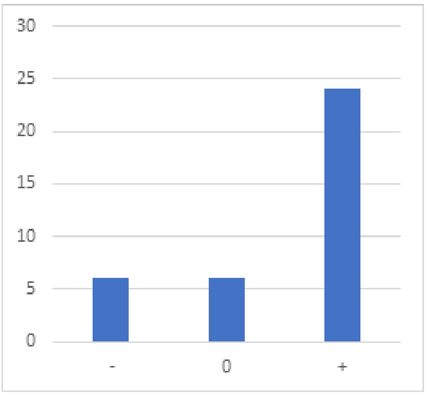


Figure 4. Changes in the results based on the evaluation of the second test (number of students with less/equal/more points)

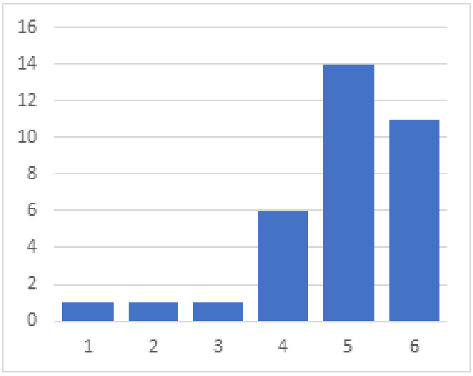


Figure 5. Students' evaluation of the tool/method

Search for major components in AVs

Methodology

While the first analysis showed that an interactive AV with oral explanation can result in a positive outcome, my purpose was to get additional information on which elements of these tools could be more important for students in this type of teaching method. In order to get detailed information about this, I needed to collect qualitative data. I could get more answers for the future usage of AV with this mixed-method research, which is a “dynamic option for expanding the scope and improving the analytic power of studies“ (Sandelowski, 2000). For the qualitative research, I conducted in-depth interviews with multiple subjects of different backgrounds. This method is useful for “getting people to talk about their personal feelings, opinions, and experiences. It is also an opportunity to gain insight into how people interpret and order the world“ (Milena et al., 2008). Thanks to the open-ended questions and the semi-structured format, the interviewees could speak freely and highlight key points in specific Algorithm Visualizations without strict predefined rules.

Process of an in-depth interview

The interview consisted of four major parts:

(1) Short conversation, acquaintance, discussion of previous programming skills.

(2) Showcasing Algorithm Visualizations; the interviewees list the components they have seen before and classify them. The following devices, applications were examined:

   (a) Sorting out Sorting: video about the sorting algorithms with commentary.

   (b) VisuAlgo: it introduces basic algorithms and data structures with visualization.

   (c) AlgTan: I made this website to support the teaching of algorithms at the university. The basics are similar to VisuAlgo, but it contains visualizations for the programming theorems too.

   (d) CodeCombat: simple online game (website) with many small tasks, which requires basic command-based solutions.

   (e) CodinGame: website with more complex tasks.

   (f) CS Unplugged: collection of different offline tasks, which require algorithmic solutions.

(3) A description of the components of the Algorithm Visualizations I have collected previously (Bende, 2019b), and a review of the results of the previous section and the components found in them.

(4) Summary based on what has been seen, discussion of additional components/possibilities.

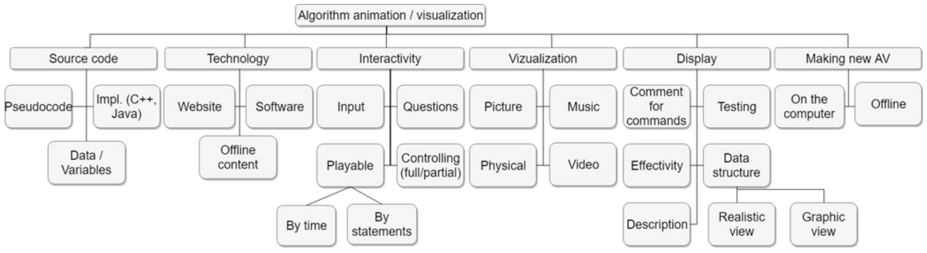


Figure 6. Possible components of Algorithm Visualization (Bende, 2019b)

Test subjects

In my research, I gathered students with different depths of knowledge and experience. I chose one person to represent each of the following four categories (hereafter referred to as this number):

(1) has not yet studied programming at university, has a basic knowledge of basic concepts;

(2) a first-year student who has already started to get acquainted with the university material, but only knows how to use the basic algorithms;

(3) a final year student who already has sufficient experience, has been studying for several years, knows and can use the taught algorithms;

(4) has completed his university studies (minimum BSc level), understands the process of algorithmizing, programming, and already has minimal work experience.

Goals

My goal is to examine the following:

- in the first step, what elements are recognized on the categorized Algorithm Visualizations already mentioned above (Bende, 2019a) and based on Myller's extended taxonomy (Myller et al., 2009);
- in the second step, which concepts mentioned in Figure 6 can be mapped to the devices shown;
- conclusions can be drawn from these results, for example, which components are considered important, and which could be used in self-directed learning;
- it can be an extra result to omit elements that are considered "redundant" when creating future visualizations and to "discover" new ones.

Interviews

Below, I will present the most important remarks of the interviewees on each tool and give a brief summary of the personal interviews.

First subject:

Sorting out Sorting: a more outdated look that uses different colours, markings, and an oral explanation associated with the video.

VisuAlgo: pseudocode is displayed, interactive elements (speed, adjustability of steps), the interface is transparent, colourful.

AlgTan: displays the pseudocode indicating the current state, using different colours, but a diagram is missing to visualize the data (for example, a column display for each element in an array).

CodeCombat: a design for children, elements of gamification that can reduce learning efficiency (too many games will lead to less new material being learned).

CodinGame: more complex application, visualizations are nicely executed, it is possible to use different programming languages.

CS Unplugged: close-to-experience structure, interactions with other people, no deeper knowledge required to try the tasks.

The subject found detailed colour visualizations to be an important component, as well as interaction elements (such as the potential to watch the process step-by-step) to aid in the use of the tool, and hence in the learning process. In addition, the display of tasks and algorithms of various difficulty was mentioned, which can thus cover programming education more widely.

Second subject:

Sorting out Sorting: easy to understand. So far, the subject has encountered only one type of visualization, which was a video like this, showing different sorting algorithms with a spectacular visual and audio association.

Visualgo: new features include modifiability, pseudo-code display, and the ability to interact.

AlgTan: the visibility of the time of the process has been revealed, the pseudocode clearly shows at which step we are, the display of the array is similar to the data storage. In addition, it was suggested that the values should be displayed in columns here as well, and that the specification could be included in the interface (this is a key concept in programming teaching at ELTE).

CodeCombat: uses player elements that indirectly help to learn algorithmizing, programming; interactive elements appear: the code is written by the user.

CodinGame: the application allows you to compete, which is accompanied by a reward system (gamification), you can view other people's solutions or even communicate with other users.

CS Unplugged: it turned out as a positive method, that you can learn and master algorithms a little further away from the computer.

The subject has identified the following items that they would consider important in a toolkit: detailed visualization, source code that appears, data display, performance testing ("worst case" examples), and the ability to test. The usability of the social elements also appeared as a novelty: the subject found it very useful that in case of specific tasks, the users can access the solutions of others and highlighted the possibility of communication with other users. Also, it has been suggested that sometimes less interaction could be better, because it is considered less effective if the user is spending too much time on side activities.

Third subject:

Sorting Out Sorting: in the case of the video material, the quantitative and colourful representation of the data and the presentation of each step with animation were mentioned.

VisuAlgo: colour data representation, pseudocode is displayed next to the visualization, the speed of the process can be adjusted, the input can be modified (also randomized).

AlgTan: compared to the previous one, there is no column appearance here, the algorithm also appears in the form of pseudocode, structure program and C ++ code, the algorithm can be scaled, but it can also be played automatically.

CodeCombat: possibility for playful learning, detailed visual display, easy-to-understand description associated with tasks, playable step-by-step .

CodinGame: the possibility of playful learning, more difficult tasks are available, more detailed, task-specific visualizations are displayed, several programming languages can be used, not only does the website allow individual practice, but we can also compare our knowledge to that of others'.

CS Unplugged: it is possible to think (about algorithmic solutions) without a computer, it is also a useful tool for young people, teacher communication helps the learning process.

According to the interviewee, the most important elements were the possibility of interaction, the introduction of playful elements and the detailed visual representation, and also that the description of the algorithm could be displayed in as many programming languages as possible. While the subject never met a tool like these, found that it could be very useful in the programming education.

Forth subject:

Sorting out Sorting: graphic view, verbal commentary.

VisuAlgo: pseudocode, description, interactive elements (playable step-by-step, changeable input), similar graphic view as the previous video.

AlgTan: pseudocode, C++ implementation, interactive elements (playable step-by-step, changeable input), does not have a detailed graphic view as we saw before, but the values are coloured for highlighting key points/values.

CodeCombat: interactive application with gamification, unique graphic view, the user has the full control over the visualization, easier tasks with basic descriptions.

CodinGame: similar concept as CodeCombat (interactive application with gamification), but with more difficult tasks, many test cases, unique visualization for many exercises, community support (but does not have real teacher–student interaction).

CS Unplugged: offline thinking developer, great way for young kids to use algorithmic solutions, it requires a mentor, who helps in the process.

The subject highlighted that interactive elements can help in the process of understanding the algorithms of the tasks, and the components of gamification could be good for younger students. On the other hand, the assistance of a teacher is necessary for new educational materials, because understanding the used tool could takes too much time, which reduces the efficiency of the learning process.

Summary

Limitations

Based on the results of the test, it is not possible to clearly link the improvement to the use of the Algorithm Visualization, as students have already encountered these algorithms before, and during the presentation C ++ sample program appeared in addition to the Algorithm Visualization, and the teacher's explanation can also help in some cases or for certain individuals.

Conclusions

The mostly positive results of the feedbacks showed that the introduced AVs were useful, helpful for university students in the learning process (Q1). A surprising result of the test was that while most of the students have already learnt programming in the past, almost none of them met with Algorithm Visualizations (it applies to the subjects of the interviews as well) (Q2). AV could make the process of mastering algorithm more colourful, more interesting, sometimes easier too, but I understand that the teachers do not have enough time to search for usable tools and then actually introduce them to the students (Naps et al., 2003).

The in-depth interviews gave two results. The first one is that interactivity (step-by-step playability, input changing) and gamification (tasks based on games with detailed visualizations, reward system) components are important parts of an ideal Algorithm Visualization, but the level of these should be adjusted based on the age of the target audience (Q3). The second one is that we discovered a few new elements that could expand the categorisation shown in Figure 6: first, the gamification components should be a part of it, because their role has an important impact on the motivation and efficiency in the learning process. The second component is the "community support": in addition to solving tasks with visualizations, speaking with other users or students can deepen the knowledge or even helps to get the correct solution (Q4). But we have to use gamification elements carefully, because a competitive environment could make the students nervous, and the lack of success could result in negative influences.

The composition of the students may vary from group to group, which determines which trends and methods are favourable for them. It is worth to try as many tools as possible and decide, based on the feedback and results, which one we will continue to use during programming teaching in the future. In the presented case, the two groups showed positive improvement and gave positive feedback, but I did not use the tool alone to demonstrate the algorithms, but also provided an oral explanation and sample programs were available too. With these in mind, we can say that the Algorithm Visualization could improve and help in the Hungarian programming education but needs further work to promote and introduce them among teachers. The most important/difficult question is how to do it. For that we have to make a small, but more widely usable collection and then show it to the teachers as many ways as possible (conferences, seminars, teacher certification programs). In order to do so, we need further research on integration of AV in programming education among teachers and expanded testing of these tools in order to ensure that the usage of the collected tools will indicate improvements and positive feedback.

References

Bende, I. (2019a). Algoritmusok oktatása algoritmus vizualizációval. In *Proceedings of InfoDidact'2018.* Webdidaktika Alapítvány.

Bende, I. (2019b). The structure of algorithm visualization tools. In *Proceedings of XXXII. Dismattech 2019.* Trnava University in Trnava.

Byrne, M. D., Catrambone, R., & Statsko, J. T. (1996). Do algorithm animations aid learning? *Technical Report GIT-GVU-96-18*.

Hundhausen, C., Douglas, S. A., & Stasko, J. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, *13*(3), 259–290. doi: 10.1006/jvlc.2002.0237

Ishizue, R., Sakamoto, K., Washizaki, H., & Fukazawa, Y. (2020). PVC.js: Visualizing C programs on web browsers for novices. *Heliyon*, *6*(4). doi: 10.1016/j.heliyon.2020.e03806

Karavirta, V. (2009). *Facilitating algorithm visualization creation and adoption in education.* [Unpublished doctoral dissertation]. University of Turku.

Lacave, C., Velázquez-Iturbide, J. A., Paredes, M., & Díaz, A. I. M. (2020). Analyzing the influence of a visualization system on students' emotions: An empirical case study. *Computers & Education*, *149*(3). doi: 10.1016/j.compedu.2020.103817

Landers, R. N. (2015). Developing a theory of gamified learning: Linking serious games ad gamification of learning. *Simulation & Gaming*, *45*(6), 752–768. doi: 10.1177/1046878114563660

Milena, Z. R., Dainora, G., & Stancu, A. (2008). Qualitative research methods: A comparison between focus-group and in-depth interview. *Annals of Faculty of Economics*, *4*(1), 1279–1283.

Myller, N., Bednarik, R., Sutinen, E., & Ben-Ari, M. (2009). Extending the engagement taxonomy: Software visualization and collaborative learning. *ACM Transactions on Computing Education*, *9*(1), 1–27. doi: 10.1145/1513593.1513600

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., ... Velázquez-Iturbide, J. A. (2003). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*, *35*, 131–152. doi: 10.1145/782941.782998

Nathasya, R. A., Karnalim, O., & Ayub, M. (2019). Integrating program and algorithm visualisation for learning data structure implementation. *Egyptian Informatics Journal*, *20*(3), 193–204. doi: 10.1016/j.eij.2019.05.001

Sandelowski, M. (2000). Combining qualitative and quantitative sampling, data collection, and analysis techniques in mixed-method studies. *Research in Nursing and Health*, *23*(3), 246–255. doi: 10.1002/1098-240X(200006)23:3⟨246::AID-NUR9⟩3.0.CO;2-H

Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S., & Edwards, S. H. (2010). Algorithm visualization: The state of the field. *ACM Transactions on Computing Education*, *10*(3), 1–22. doi: 10.1145/1821996.1821997

Simonak, S. (2020). Increasing the engagement level in algorithms and data structures course by driving algorithm visualizations. *Informatica*, *44*(3), 327–334. doi: 10.31449/inf.v44i3.2864

Supli, A., Shiratuddin, N., & Syamsul, B. Z. (2016). Critical analysis on algorithm visualization study. *International Journal of Computer Applications*, *150*(11), 18–22. doi: 10.5120/ijca2016911633

IMRE BENDE
FACULTY OF INFORMATICS, EÖTVÖS LORÁND UNIVERSITY,
H-1117 PÁZMÁNY PÉTER SÉTÁNY 1/C, BUDAPEST, HUNGARY

E-mail: `beiraai@inf.elte.hu`

(Received January, 2022)