Planets: A System for Autonomous Learning of Algorithmic Programming [*]

László Nikházy

Eötvös Loránd University, Budapest, Hungary
nikhazy@inf.elte.hu

Abstract. This paper presents the design of a system where children can explore the world of algorithmic programming through solving problems. Learning by doing certainly increases the engagement of pupils, furthermore, it is much easier to keep the learners' attention when they are actively participating rather than just listening. The author teaches algorithms with a discovery learning method that incorporates presenting well-defined problems to students where the solution requires constructing the algorithm to be taught. To formulate the algorithms, students write programs that can be evaluated with black-box testing. This is very similar to programming competition tasks, and while there is a strong connection to them, the primary goal of the system is education, not assessment. Aiming to make this teaching approach available to a much larger audience, we propose a website that tries to fill the instructor's role in the learning process. Gamification is a central principle throughout the design of the system. Therefore, it is built around a story that supports the methodology: the learner discovers the universe of programming and algorithms by traveling from planet to planet by a spaceship. Planets represent topics and every task is an adventure or a challenge posed by inhabitants of the planet. The article describes features of the designed web application, and the principles of selecting and organizing its content. In addition, the roadmap of topics that contain dependencies is outlined – this determines the order of exploring the planets, which is flexible to some extent. Finally, the author presents an example planet to show a complete piece of the system.

Keywords: Problem-solving · Learning by discovery · Algorithmic programming · Novice programmers · Educational software · Teaching methodology · Self-learning.

1 Introduction

1.1 Motivation

The success and effectiveness of discovery learning is a debated topic [1, 2]. Its application in computer programming remains a challenge. However, the au-

[*] Supported by the project "Integrated program for training new generation of researchers in the disciplinary fields of computer science", No. EFOP-3.6.3-VEKOP-16-2017-00002. The project has been supported by the European Union and co-funded by the European Social Fund.

thor is inspired by the work of Hungarian mathematician Lajos Pósa, which has emerged to an amazing talent education program [3–5] carried on by the community of his former students.

My current research goal is to apply the *Pósa method* [4] in computer programming education. I am focusing on teaching gifted children, because the former experiments of Pósa (unpublished) and the complex mathematics teaching experiment of Tamás Varga [6] show that it is very hard to use discovery learning in public education, while it is successful with young talents [3]. I have been giving individual and small group classes with promising results, and I am starting computer science camps [7]. As a software engineer, I would like to see if it is possible to extend this education's scope by creating an online platform that fills the instructor's role in the discovery learning process. The name of the system will be Planets. This paper presents the implementation of the didactic principles that I follow [8] in an educational website, and also acts as the design document of it. During the design process, the main research questions were: (1) What are the key elements in the Pósa method aiding discovery through problem solving that can be integrated to an automated self-learning environment? (2) How can we use gamification into a website teaching algorithmic programming? (3) What features can be built into the system to ease the lack of a teacher? (4) How can we incorporate existing tasks of past programming contests?

1.2 Current learning materials in algorithmic programming

The term *algorithmic programming* is rather new, it is used when the primary goal of programming is to solve problems of algorithmic nature. Such problems are usually well-defined, there is a clear statement describing the expected output for any given input. The solution requires using problem-solving techniques, data structures and algorithms, and the program code is a way of expressing the solution that allows testing it on a computer. The correctness and effectiveness of the solution are most often verified by running it against a lot of test cases.

The above description fits most of the programming competitions, and while there is a strong connection, algorithmic programming is not the same as competitive programming. Competitive programming is characterized by the competition, the domain of which can vary, for example it can be artificial intelligence, or application design, and also algorithmic programming. However, algorithmic programming is also used to improve problem-solving skills, or to teach algorithms and data structures – implementing a learned concept promotes deep understanding.

The topics of algorithmic programming have four main categories: (1) problem-solving techniques, (2) algorithms or algorithm templates, (3) data structures, and (4) theoretical backgrounds. In the following, the term *topic* is used as a general substitute for any element in them. For example, the greedy method, Dijkstra's algorithm, the heap, or the greatest common divisor are all considered a topic in the context of this article.

There are a lot of excellent resources available online which promote learning algorithmic programming on an advanced level, some of the most notable examples are Halim's book [9], the massive problem base of past Codeforces [10] and CodeChef [11] contests, the HackerRank platform [12], the GeeksforGeeks [13], CP-Algorithms [14], and the very new USACO Guide [15] websites. However, the content of them is either not structured for learning, or the tasks are presented after the explanation of the algorithm needed to solve them. HackerRank and the USACO Guide are the most similar to our system, but they also lack the motif of discovery learning. In our scenario, the topics involved in the solution should remain unknown to the learner when facing a task.

1.3 Discovery learning in algorithmic programming

The above-mentioned Pósa method for mathematics education is a form of guided discovery learning. According to Bibergall [16], guided discovery learning is characterized by convergent thinking, the educator devises a series of statements or questions that guide the learner step by step, making a series of discoveries that leads to a predetermined goal. In Pósa's math camps, the learner is guided through exercises that have strong interconnection under the surface. Katona and Szűcs [5] describe this as a web of problem threads.

Some alterations are necessary for the Pósa method to apply it in computer programming, due to the differences from mathematics. There are a lot of standard algorithms and data structures which are almost ready-made methods that need to be customized, combined, and applied in numerous different scenarios. Our goal is to teach them through a series of problems, having the students discover them mostly on their own, if possible. However, direct explanation cannot be avoided for every topic. For instance, it would be very hard to make students discover the Fenwick tree data structure [17] or the Edmonds-Karp algorithm [18] by solving problems. In such cases, there is even more emphasis on the applications of the method in different problems.

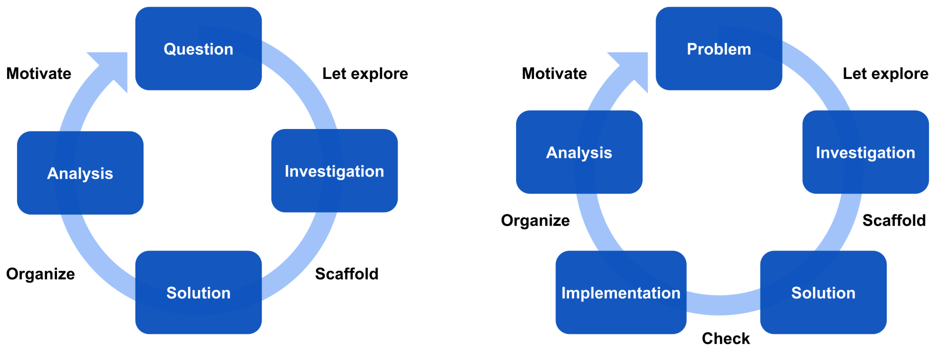


Fig. 1. Process of discovery learning in mathematics (left) and programming (right)

The cycle models in figure 1 demonstrate the discovery learning process in Pósa's mathematics camps and the author's computer programming talent education. The implementation of the solution, also called coding, is an important phase in algorithmic programming, which is a completely extra step compared to the mathematics problem-solving process. It allows automatic evaluation of the solution, while in mathematics, it is the teacher's responsibility to check the solutions, since, in most cases, a proof is necessary for the answer. The computerized evaluation also means that the verification step is much less flexible than in mathematics, where the teacher can lead a discussion about the students' solutions. To reduce the negative effects, the instructor has an important role in discussing the theoretical correctness of the solutions, giving feedback about the students' program codes, and helping to correct errors, especially with novice programmers. It is extremely hard to substitute the teacher with software features at this phase, our system tries easing this problem with simple and meaningful error messages, static code analysis, careful test case design, and pre-written useful advice.

2 System design

2.1 Requirements

It is important to define the requirements towards the system. Later on they are referenced with their abbreviation. The website should

- teach problem-solving strategies, algorithms and data structures (REQ_TPC)
- with minimal amount of direct explanations of the topics taught (REQ_DL);
- include the teaching of the elements of a programming language that are necessary to learn the above topics through solving problems (REQ_PL);
- keep the possibility to adapt it to multiple programming languages (REQ_MPL);
- be available in English and Hungarian, and can be translated to other languages (REQ_LANG);
- use the principle of gamification (REQ_GAM);
- be targeted to 12-18 year old children, but might be appealing to adults as well (REQ_AGE);
- enable the users to track their progress and make them aware of the topics already learned (REQ_AW).

2.2 Story: exploring the universe of algorithmic programming

The discovery learning approach (REQ_DL) and gamification (REQ_GAM) is supported by the story of the system: the user explores the universe of programming, algorithms and data structures by traveling from planet to planet by a spaceship. Every visited planet has a civilization and the inhabitants pose challenges to the traveler. Under the surface, the planets represent different topics. In general, every element of the system can be translated to fit the space theme, for example: topic – planet, task – challenge or adventure, curriculum – map, reward – progress points and space currency or badge.

The rewards have a central role in the learning mechanism. Solving a challenge can (1) allow the user to enter a planet where further challenges await; (2) give way to a previously undiscovered planet; (3) award a badge to the user; (4) grant them space currency; (5) increase their progress points. At each challenge there are several hints available that help the students, and they do not affect the rewards they get, except for certain badges. Space currency can be used for further help, for instance revealing a test case or even the solution of a task. It can be used also to unlock a planet without completing all the dependencies of it (intended for advanced level students).

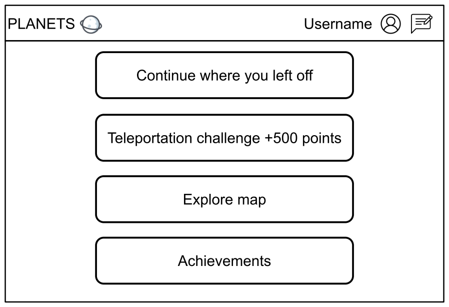
During the gamified learning process, the user solves tasks that are connected to each other in a non-linear structure. The solutions of the tasks rely on experience gained in previous tasks. The tasks are grouped together by topic for visualizing the roadmap of the studies, but the learner is encouraged to take on a task without knowing the topic it belongs to. The dependencies are built into the system on the level of topics – completing a planet can unlock other ones. However, the tasks within a topic also have a suggested order that is used automatically, but the learner has options to skip them.

2.3 User interface

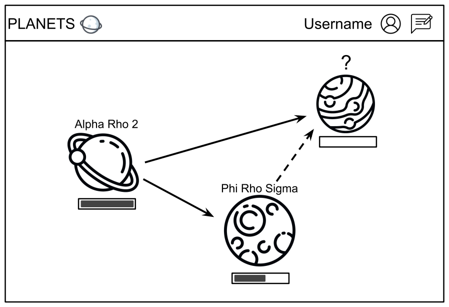
Welcome screen After login, the user is taken to the welcome screen. It has four main items that lead to different pages.

- *Continue where you left off* gives the user a challenge that comes next in their progress, or opens their last saved code if they left in the middle of solving a task. Opens the task view.
- *Teleportation challenge for bonus points* gives the user a challenge from a planet that they already discovered without telling which planet they are at. This way, the domain of the task is unknown to the learner, which is the typical situation in the discovery learning method of Pósa (REQ_DL). Opens the task view.
- *Explore map* shows the user the system of visited and unlocked but not yet visited planets, so they can see their progress (REQ_AW) and select challenges on a chosen planet. Opens the map view.
- *Achievements* presents badges that the user already earned or can get in the future (REQ_GAM). These are special rewards that help keeping motivation. Opens the achievements view.

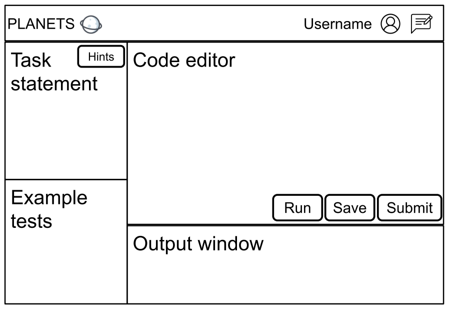
Task view The task view is a simple development environment in the browser, tailored to support the problem-solving scenario. There are some good existing examples like [12, 20, 21]. At our university, a similar system is developed, called ProgEnv. It is close to being fully functional and it needs only a little customization to fit inside the Planets system. The screen design presented in figure 2 has all the typical features of similar problem-solving sites: the task description and sample tests are visible together with the code editor [19]; there are buttons to


PLANETS
Username
Continue where you left off
Teleportation challenge +500 points
Explore map
Achievements

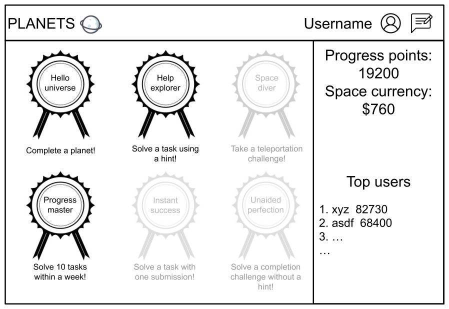
(a) Welcome screen



(b) Map view



(c) Task view



(d) Achievements

Fig. 2. Schematic design of the four screens

save, run and submit the code; an output window shows the messages printed by the program and the compiler. There are two significant improvements to support educational purposes. Firstly, many small example test cases are provided with the task description, which can be plugged in easily to the execution of the program. Secondly, there is a very important small feature that comes from the teaching method of Pósa, the *Hints* button next to the task statement. With this button, the user can reveal several hints that provide gradually increasing amount of help in solving the problem. After submitting, a popup is shown in the middle of the screen containing the results of each test case. If the solution gets accepted (passes all test cases), the rewards are shown, along with two short feedback questions and the sample solution. The sample solution is crucial, because it takes the role of the teacher's explanation on how to solve the task most elegantly. The users can compare it with their own solution and conclude what they could do better. The two feedback questions are: (1) How difficult was the challenge for you? (2) How did you enjoy the challenge? The answer to both questions is a scale from 1 to 5 stars.

Map view The map view visualizes the known topics and the connections between them. This makes the students aware of their knowledge and provides a systematic overview of the topics learned and exercises solved. It is a frequently visited reminder of past challenges. In Pósa's talent education program, the same goal is achieved by a document called the camp summary, which lists all the solved problems and discussed questions organized according to mathematics areas and problem-solving methods.

The planets are presented in a 2D layout which is designed to reflect the structure of the curriculum. There are arrows between the planets which show dependencies. Below each planet there is a progress bar indicating the amount of completes challenges and at a suitable zoom level, the names of challenges solved on that planet are shown. A planet can be in five basic states:

- *Explored*: the user solved sufficient challenges to consider the topic of the planet known. Dependent planets are unlocked.
- *Discovered*: the user entered the planet and started solving challenges, but it is not explored yet.
- *Unlocked*: the user can enter the planet, but have not done so. The name of the planet is not visible.
- *Locked*: the user cannot enter the planet yet, because at least one required planet is not explored yet. At least one dependency of it is unlocked (otherwise it is invisible).
- *Hidden*: the planet is not shown yet to the user, because no dependency of it is unlocked. The benefit of this is to avoid overwhelming the learner with a big amount of unknown items.

2.4 Content

The main content of the system are the tasks, which are organized into topics. If some topic needs direct explanation, it can be put inside a task statement. To design the set of tasks, first the topics need to be identified and arranged to a system with dependencies (partial ordering). This network of topics serves as the curriculum, and a lot of work has been done regarding it [7].

Topics We analyzed many existing learning materials and topics at programming contests to create the curriculum for our talent education program in [7]. There are more than 50 topics presented in that article, with interconnections between them. We also collected some nice tasks for teaching each topic. They will form the planets in this system with some adjustments and extensions. For example, teaching elements of a programming language is not in the scope of [7], but it will be part of the Planets system. Table 1 shows the list of planned planets to achieve this goal together with some of the first topics in algorithmic programming to provide further context. As mentioned above, our preference is to teach only the most important toolset of programming required to implement the given algorithms. Out of the listed 24 planets only 10 of them (1-7, 14, 16, 22) are meant to teach new language elements.

Tasks There should be 5–10 challenges on each planet. There are two special challenges: the entry challenge and the completion challenge. The entry challenge

Table 1. The first 24 planets in the system

#	Name	Description	Dependencies
1	Output, Operators	Print messages and results of simple calculations with constants.	
2	Input, Variables	Read data from user input and perform calculations with variables. Primitive data types.	1
3	If-then-else	Conditional statement and boolean expressions.	2
4	For Loop	Repetition: for loop with an index variable.	3
5	Strings	Work with character strings, perform simple transformations on them.	4
6	While Loop	Repetition with a boolean condition: while loop.	4
7	Arrays	Use 1D arrays to store multiple items of primitive data.	5
8	Sum	Calculate the sum of a series of numbers.	7
9	Count	Count the elements of a series with a certain property.	8
10	Search	Search for element(s) with a given property in a sequence.	6, 7
11	Minimum Selection	Select the minimum/maximum in a sequence.	9
12	Programming Patterns	Combinations of the above (Sum, Count, Search, Minimum Selection), more difficult problems. Nested loops.	9, 10, 11
13	Histogram	Count the occurrences of each element in a sequence using an array of counters.	9, 11
14	Functions	Write functions for repeatedly used code.	4, 5
15	Basic Number Theory	Calculate the number of divisors and sum of divisors of an integer, search for primes, prime factorization, Euclidean algorithm for GCD.	9, 10, 14
16	2D Array	Use 2D arrays to represent the data in a table/matrix, process them with combinations of simple programming patterns.	12
17	Sort	Sort a sequence with simple algorithms, like bubble sort, min. selection sort, counting sort, etc. Problems that require sorting.	12
18	Merge	Compute the intersection / union of two sets: ordered list of data, using the linear merge algorithm.	17
19	Greedy Algorithms	Solve problems using greedy decisions, recognize whether it leads to the optimum.	17
20	Prefix Sum	The prefix sum / cumulative sum of a sequence, as a data structure.	16
21	Two Pointers	The two pointers principle for speeding up some optimization tasks.	13, 17
22	Struct	Group data with the composite data type (struct).	16
23	Recursion	Get familiar with the power of recursion in typical scenarios.	14
24	Merge Sort	Recursive sort algorithms: merge sort and quicksort.	18, 23
...	...	...	

should be designed to promote discovery of the planet's topic. After solving several challenges, the user arrives at the completion challenge, which is a complex task designed to test whether the user can apply the learned methods, and later we can rely on this knowledge. Solving the completion challenge unlocks dependent planets. The completion challenge is not necessarily the most difficult on a planet, there can be some more complicated extra tasks. There is a suggested order of tasks on a planet coded into the system, which is suitable for beginners, some tasks are marked optional, and more advanced students can skip them. Former programming competition tasks inspire the majority of the tasks in the system. These are aimed to test the algorithmic thinking and problem-solving skills of the participants. There is a great amount of publicly available tasks online, and our university has a big collection of former Hungarian competition problems, which is extended with a lot of exercises used only for education [22].

Adapting competition tasks At competitions, the test cases need to be kept secret. Revealing the test cases for educational purposes is beneficial in the author's experience. Codeforces [10] shows the first few hundred bytes of every test case in practice mode, in HackerRank [12] there is a possibility to unlock inputs one by one. Looking at an example, where the program gives wrong answer, can be very helpful in finding bugs or understanding why the solution is incorrect. The size of the example is a key factor, it should be easily processable by the human brain. So, for every task in the Planets system, a lot of small test cases are generated, which are visible to the user at the time of the submission, and a handful of them are shown already together with the task statement. The learner gets a partial reward for a solution that is correct for all of them. A separate test group are the big tests used to check the efficiency of the solution. They are kept in secret by default, but can be unlocked by paying some space currency.

An important element of Pósa's pedagogy is giving hints to students who have difficulties to find the path of the solution. He argues that solving a problem with some help is much better than listening to the solution. It is not easy to give good hints for every task, but they can be designed by the teacher in advance. This is exactly what the Planets system contains, there are at least three prewritten helping texts for each task. They come in a self-service fashion: the learner can retrieve the increasingly significant hints one by one. If the system detects that the user is struggling with the solution, the button for the hints will be highlighted. Finally, the sample solution can be unlocked by paying sufficient space currency. This is to make progress possible even if the student cannot find a bug. The sample solution is automatically shown after a correct solution. This means that the sample code should be understandable and well commented, it acts like the explanations of the teacher.

3 An example planet

The planet is about the **prefix sum** data structure, also called as partial sum, or cumulative sum. The basic idea is to calculate the prefix sums of a sequence,

$S_i = A_1 + A_2 + ... + A_i$ for every i. Using these values, the sum of any range of the sequence can be obtained with a single subtraction: $A_L + ... + A_R = S_R - S_{L-1}$.

The entry challenge (given as Example 1 below) is the basic task of calculating range sums quickly, but with special attention drawn to the prefix sums. We list the hints for this task as an example. Since the solution has one main idea, all the hints are elaborating that with increasing details. The next challenge on this planet (Example 2) is inspired by [23]. It is an easy application of the prefix sum data structure. We include a problem on this planet (Example 3) that has another solution with greedy strategy. Still, the solution with prefix sums makes a very good practice and enhances understanding of the data structure. The fourth challenge (Example 4) involves a nice idea that can be viewed as an application of prefix sums. Given some intervals, we are interested in how many intervals cover each point on the number line. If we substitute the start of each interval with +1 and the end of it with -1 then the prefix sum of all these numbers gives the answer at any point. The completion challenge of this planet (Example 5) is a reinterpretation of [24]. This task is building on the solution of Example 2 and 4.

Example 1. Welcome to our planet! We need your help! It has been N days since the asteroid-rain started to fall on us ($N \leq 100\,000$). We recorded the number of asteroids that we detected each day ($A_i, i = 1..N, A_i \leq 10\,000$). Our mage, Sigissimus says that he can disrupt the force to stop the catastrophy, but he needs to know instantly the answer to two types of questions: (1) How many asteroids were detected in the first K days altogether? (2) How many asteroids were detected in total between two given days (including the given days)? Sigissimus asks a lot of questions ($Q \leq 100\,000$), please help us in giving the answers!

1. Hint: Fill an array with answers to all possible questions of the first type. Can you calculate the answers to the second type?
2. Hint: If we have $S_i = A_1 + A_2 + ... + A_i$ for every i, then the sum of a range ($A_L + ... + A_R$) can be obtained with one operation. Can you find out how?
3. Hint: Let us calculate the so called prefix sums, $S_i = A_1 + A_2 + ... + A_i$ for every i. This can be done with one loop that sets every $S_i = S_{i-1} + A_i$, with initially $S_0 = 0$. The answer to the first type of question will be S_K. The answer to a second type of question, the total asteroids between day L and R (inclusive) will be $A_L + A_{L+1} + ... + A_R = S_R - S_{L-1}$

Example 2. You are given a 0-1 sequence $s_1, s_2, ..., s_n$ ($s_i = 0$ or 1, $n \leq 100\,000$), and q queries ($q \leq 100\,000$). Each query is described by a pair of integers l, r ($1 \leq l < r \leq n$). The answer to the query l, r is the number of such integers i ($l \leq i < r$), that $s_i = s_{i+1}$.

Example 3. Given a sequence of (positive or negative) integers $A_1, A_2, ..., A_N$ ($-1000 \leq A_i \leq 1000$, $N \leq 100\,000$), can you tell which range of it has the greatest sum? Output the L, R indexes for which the sum $A_L + A_{L+1} + ... + A_R$ is greater or equal to the sum of any other range of consecutive elements.

Example 4. On our planet, the dates are simply given with one number, the number of days from the very start. We have N ($N \leq 100\,000$) heroes and we know the birth and death dates of them ($1 \leq B_i < D_i \leq 1\,000\,000$). Can you tell what was the maximal number of heroes living at the same time?

Example 5. There are N ($N \leq 100\,000$) cities on the road from Phyria to Sigmaria. M ($M \leq 100\,000$) travel books recommend different journeys, each of them consists of consecutive cities on the road. The journeys are given with their start and end city ($1 \leq S_i < E_i \leq N$). A city is *interesting* if at least K books recommend it. You are given Q ($Q \leq 100\,000$) queries, in each query you have to tell how many interesting cities there are between two given cities.

4 Conclusion

This article presented a design of a website that teaches algorithmic programming through solving problems, inspired by the Pósa method for teaching mathematics and taking certain elements of it. It demonstrates that in the field of computer programming, it is possible to give young talents the opportunity of learning by discovery in a fully automated, interactive, self-learning environment. The elements of the system are based on the action research of the author regarding discovery learning in computer science talent education. The web of problem threads [5] theoretical frame in the Pósa method can be integrated into the system with careful problem selection and organization. The map-like visualization of solved challenges and learned topics is an improvement to the camp summaries used by Pósa. Both the curriculum and the progress of the student is well observable, while it fits into the gamified theme of the system. The story of exploring the universe of algorithms provides an opportunity for gamification and attractive presentation, moreover, it creates a direct correspondence to learning by discovery. The teacher's scaffolding role is substituted by multiple hints for each task, and careful test case generation, which can help to detect errors. Well-written and commented sample solutions act as the explanations of the teacher, and visualizing the connections with other topics and tasks provides the organization of acquired knowledge. Students can be motivated with standard elements of gamified environments, like achievements, rewards, and leaderboards.

Implementation of the web application has started, we are committed to making it available for everyone and publishing our findings in a follow-up paper. This paper described the MVP (minimum viable product) version of the system. There are many features that would be very important to include, we have a lot of further plans outside the scope of this article.

References

1. Kirschner, P.A., Sweller, J. and Clark, R.E.: Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. Educational psychologist, 41(2), 75–86 (2006) https://doi.org/10.1207/s15326985ep4102_1

2. Schmidt, H.G., Loyens, S.M., Van Gog, T. and Paas, F.: Problem-based learning is compatible with human cognitive architecture: Commentary on Kirschner, Sweller, and Clark (2006). Educational psychologist, 42(2), 91–97 (2007) https://doi.org/10.1080/00461520701263350
3. Győri, J., Juhász, P.: An extra-curricular gifted support programme in Hungary for exceptional students in mathematics. In: Taber, K., Sumida, M., McClure, L. (eds.) Teaching Gifted Learners in STEM Subjects, pp. 89–106. Routledge, London (2017). https://doi.org/10.4324/9781315697147-7
4. Juhász, P., Katona, D.: Pósa method: Talent Nurturing in Weekend Math Camps. In: Including the Highly Gifted and Creative Students – Current Ideas and Future Directions : Proceedings of the 11th International Conference on Mathematical Creativity and Giftedness. WTM, Münster, pp. 270-276. (2019)
5. Katona, D. and Szűcs, G.: Pósa-method & cubic geometry: A sample of a problem thread for discovery learning of mathematics. In: Karlovitz, T.J. (ed.) Differences in pedagogical theory and practice, pp. 17–34. (2017) https://doi.org/10.18427/iri-2017-0079
6. Halmos, M., Varga, T.: Change in mathematics education since the late 1950's – ideas and realisation: Hungary. Educational Studies in Mathematics 9(2), 225–244 (1978)
7. Nikházy, L.: A Problem-based Curriculum for Algorithmic Programming. Central-European Journal of New Technologies in Research, Education and Practice 2(1), 76–96 (2020) https://doi.org/10.36427/CEJNTREP.2.1.399
8. Nikházy, L.: Algoritmusok tanítása problémaközpontú módszerrel. In: Bihari, Erika; Molnár, Dániel; Szikszai-Németh, Ketrin (eds.) Tavaszi Szél - Spring Wind 2019. I. kötet, pp. 557–570. DOSZ, Budapest, Hungary (2020)
9. Halim, S., Halim, F., Skiena, S.S., Revilla, M.A.: Competitive Programming 3. Lulu Independent Publish (2013)
10. Codeforces, https://codeforces.com/. Last accessed 22 Jul 2020
11. CodeChef, https://www.codechef.com/. Last accessed 22 Jul 2020
12. HackerRank, https://www.hackerrank.com/. Last accessed 22 Jul 2020
13. GeeksforGeeks, https://www.geeksforgeeks.org/. Last accessed 22 Jul 2020
14. CP-Algorithms, https://cp-algorithms.com/. Last accessed 22 Jul 2020
15. USACO Guide, pre-release version, https://usaco-guide.vercel.app/. Last accessed 22 Jul 2020
16. Bibergall, J. A.: Learning by discovery: Its relation to science teaching. Educational Review. Vol. 18(3) 222-–231 (1966) https://doi.org/10.1080/0013191660180307
17. Fenwick, P. M.: A new data structure for cumulative frequency tables. Software: Practice and Experience. **24**(3), 327—336 (1994)
18. Edmonds, J., Karp, R. M.: Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM. **19**(2), 248—264 (1972)
19. Monaco Editor, https://microsoft.github.io/monaco-editor/. Last accessed 22 Jul 2020
20. Repl.it https://repl.it/. Last accessed 22 Jul 2020
21. CodeInterview https://codeinterview.io/. Last accessed 22 Jul 2020
22. MESTER – online programming task archive (in Hungarian) http://mester.inf.elte.hu/. Last accessed 22 Jul 2020
23. Codeforces Problem 313B. Ilya and Queries. https://codeforces.com/problemset/problem/313/B. Last accessed 22 Jul 2020
24. Codeforces Problem 816B. Karen and Coffee. https://codeforces.com/problemset/problem/816/B. Last accessed 22 Jul 2020