






Bebras Tasks Based on Assembling Programming Code

Jiří Vaníček^(✉) , Václav Šimandl , and Václav Dobiáš 

University of South Bohemia, České Budějovice, Czech Republic
{vanicek, simandl, dobias}@pf.jcu.cz

Abstract. The paper examines the creation and evaluation of so-called situational informatics tasks based on assembling a program from blocks. Blockly technology has enabled us to develop an environment where templates, called “worlds“, can be created. In these worlds, pupils program a certain sprite to solve a problem emerging in a described situation. We created two such templates – the world of Karel the robot and the world of Film animation, differing both in behavior of sprites and set of commands. Each template was supplied with its own set of tasks, differing in topic, subject matter and graphics. As they go through each task, pupils repeatedly run the assembled program, being provided by the system with feedback. That comprises a visual check of how the programmed sprite behaves as well as system-generated notifications reporting whether all the requirements for completing a task have been met. The tasks that were compiled for this purpose were included in the Bebras Challenge. In our paper, we describe each of the templates and look at their didactic background as well as examining findings from the practical use of these tasks in the Challenge and their inclusion in the informatics curriculum. Results show that tasks created for the world of Karel the robot used in the Bebras Challenge are no more difficult than other algorithmic tasks. Moreover, informatics teachers are impressed with these tasks and they find it of utmost importance that the curriculum includes such tasks in order to advance pupils’ informatics skills.

Keywords: Computational thinking · Algorithmization · Block programming · Primary school · Secondary school · Bebras Challenge

1 Introduction

Programming is generally perceived as a matter of specialized professional training. However, Gander claims it is an essential part of general public education for the 21st century [1]. Not only providing us with the opportunity to discover the world from another perspective and understand how a computer works, programming can also be perceived as a microworld that can develop an individual’s mental abilities. The direction of teaching and subsequent choice of appropriate educational content, environment, motivation and teaching methods all derive from its basic definition. From the perspective described above, programming is presented in our article as a training ground for

developing an individual's abilities and competences. The same approach is found in strategic documents like the "Shut down or restart?" study in the United Kingdom [2], the worldwide ACM computing curriculum [3], CSTA K-12 computer science standards in the United States [4], our close neighbors' Štátny vzdelávací program in Slovakia [5] and Podstawa programowa z informatyki in Poland [6].

If we consider education's general aim as being to primarily develop personality, in the field of informatics it is the development of computational thinking [7] as the ability to find a solution to a problem in a form which could be automatically carried out by an information-processing agent. Algorithmization, i.e. identifying a method or process to achieve a goal and formulating it in a way so that such an agent could read and perform it, is a fundamental part of computational thinking. This term became the mainstay for defining school curriculum content in the above-mentioned countries as well as the Czech Republic in its government Strategy of Digital Education [8] and in its proposal for new General Curriculum programs [9].

According to Cuny, Snyder and Wing, the idea of computational thinking for everyone involves abilities such as understanding what aspects of a problem are amenable to computation, evaluating the match between computational tools and techniques and a problem, using or adapting a computational tool for a new use and identifying opportunities to use computation in a new way [10]. In order to be able to develop these abilities through programming, we must try to find suitable approaches, situations, tasks and also environments which will highlight and emphasize these goals. Wittmann defines such an environment as a set of interconnected situations providing problems which enable a pupil to identify important thoughts [11].

Xia defines teaching of programming as supporting students to understand the concepts of programming via hands-on experiences and learning as the activity of obtaining useful programming knowledge and skills by studying [12]. Many approaches to teaching programming favor student activity, active learning, learning by doing, and the construction of knowledge as a result of active creative work. All this with respect to the fact that knowledge and knowing are not transmittable. According to Piaget, knowledge is actively constructed by the learner in interaction with the world, so, as Ackermann [13] suggests, it is worth providing opportunities for children to engage in hands-on explorations that fuel the constructive process. Ackermann quoted Piaget's theory that "children interpret what they hear in the light of their own knowledge and experience", and his belief that "knowledge is formed and transformed within specific contexts, shaped and expressed through different media" [14]. How one constructs knowledge is a function of the prior experiences, mental structures, and beliefs that one uses to interpret objects and events [15].

Two basic types of programming tasks can be found in coursebooks and manuals for the teaching of programming:

- "Études" lasting several minutes, always focused on a specific skill or programming concept, their aim being particular knowledge acquisition;
- Bigger "projects", often in the form of creating stories or games which are more complex, the outcome being a product.

Études allow better detection of a learner’s error and appropriate sorting of such tasks facilitates the creation of mental models of a learner. Projects require a combination of more skills at the same time, including planning and creativity; the created longer programming codes require more knowledge from a learner but this is counterbalanced by higher motivation, as a learner works towards a final product. For example, études were used in the textbook [16], projects were used in the textbook [17].

If a learner solves problems by creating software, specifically by assembling a program, a teacher can get feedback by analyzing the written program to determine how a learner has understood the situation which the problem he/she is solving occurs in; how well he/she understands the concepts he/she uses; what level he/she has reached in terms of elements of computational thinking like algorithmization, decomposition and generalization; or the approaches he/she uses to solve problems [18].

The Bebras Challenge has contributed to the development of computational thinking for a number of years [19], being held in more than 60 countries worldwide [20]. Via an online test, the tasks put learners in a situation where they have to determine the corresponding informatics concept and select an answer by applying their computational thinking. The situational tasks used in the Bebras Challenge – in the Czech version called Bobřík informatiky [21] – are similar to études in their structure and focus on a particular informatics concept. The contest consists of an online test so there are multiple-choice, click on object or drag object tasks. Bebras produces a number of new informatics tasks, contributing to innovations in the school curriculum, some of the tasks being incorporated into new Czech informatics coursebooks [22].

1.1 Motivation and Aim

Situational “Bebras” tasks should develop various aspects of computational thinking including algorithmization. Typical algorithmic tasks used in the contest include identifying start and end state after applying a particular algorithm, comparing several algorithms with a task assignment, considering rules for carrying out a computation, identifying an error in an algorithm or its optimization. The contest did not include tasks to be answered by assembling a program, which restricted the variety of informatics tasks in the contest.

We tried to find a solution that would enhance the existing contest to include tasks where, just like in the programming environments used in schools, contestants could assemble a program from blocks. This would involve using the widespread concept of block programming, known from programming environments like Scratch, Blockly or MakeCode, which learners are familiar with from informatics or robotics coursebooks. Such a solution will bring innovation into the Bebras Challenge which will benefit from this newly developed type of tasks.

2 Methods

Solution methodology proceeds from design-based research as per Trna [23]. To develop the software module, we first analyzed familiar open source block programming environments (e.g. Scratch, Blockly, MakeCode) to determine whether they could be used

to create the software module, primarily in terms of pedagogy and implementation. We then analyzed familiar “worlds” which the programming tasks would be created in (e.g. Karel the Robot, turtle graphics, Baltie the magician) in terms of:

- their ability to cover the curriculum range (minimum of pre-entry knowledge, maximum of educational aims),
- their suitability for the creation of a set of tasks that progress in small steps with regards to acquired knowledge
- their suitability for the creation of particular “contest” tasks, considering the specific nature of each one.

2.1 Design

We designed and developed a software environment where interactive situational programming tasks can be created. A learner solves a problem in it by assembling a program from blocks and is given the possibility of testing and debugging his/her program. We implemented a modified module of Blockly [24] into our environment.

The advantage of block programming is that it prevents syntax errors. In our implementation, it also has a limited set of programming commands, which encourages learners to think rather than searching for a tool that could conveniently solve the problem for them.

As they go through each task, pupils repeatedly run the assembled program, being provided with feedback by the system. That comprises a visual check of how the programmed sprite behaves as well as system-generated notifications as to whether all the requirements for completing a task have been met. The environment enables the creation of sets of tasks that follow on as the learner progresses, similar to Hour of Code activities [25], resulting in tasks of increasing difficulty with the progressive employment of more complex concepts and situations.

Each time a learner asks for a program to be run by clicking on Run button, the system simultaneously saves the learner’s program along with information as to whether his/her solution has met all assigned requirements and the number of attempts the learner required to create the right program.

To accompany this software environment, we developed two templates of programming tasks, so-called “worlds”, each having program-controlled sprites that behave differently and each having different sets of basic commands. Each template was supplied with its own set of tasks, differing in topic, subject matter and graphics.

1st World: Controlling the Robot.

The first template simulated “the world of Karel the robot” [26], a sprite that walks around a system of squares picking up and putting down objects. The basic set of commands can move the programmed sprite around the game board one square at a time, make a quarter turn in both directions, detect objects on the square where the sprite is located and remove such an object from a square or place an object on a free square. It can also detect an obstacle on an adjacent square in the direction the sprite is facing. The basic language commands were supplemented with a Repeat structure, constituting a loop with a fixed number of repeats (see Fig. 1).

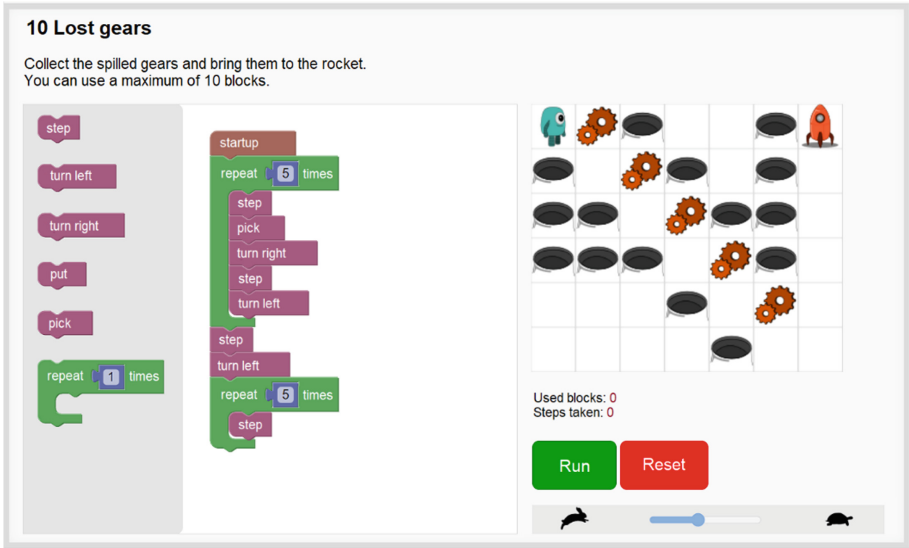


Fig. 1. The world of Karel the robot and the task in which learners are assigned to collect gears. The program created by a learner to complete the task is in the middle.

We chose “the world of Karel the robot” as it is simple enough for a learner to understand and manage the basics of the language so that he/she can quickly move on to more complicated tasks. The advantages of this “world” include the possibility to create real situations, the visual clarity of the sprite’s status, facing one of the four main directions so not requiring a turning angle parameter as well as the limited number of basic language commands (step forward, turn, pick, put). Another advantage is the absence of more complicated terms like object, coordinates, procedure or variable. This reduces the amount of time needed to get acquainted with the environment, meaning that learners can soon progress onto and concentrate more intensively on the algorithmic core of the solved problems.

Typical tasks in this world were to go to a particular place, avoid an obstacle, pick up equally distributed objects or find a way composed of multiple parts. In this world, we have created a set of programming tasks which gradually increase in difficulty. The loop programming concept was used to repeat one block, assemble a program with blocks preceding and/or following a loop; with several blocks in the loop body; with several loops following each other in a program and find a way to complete a task using the shortest possible programming code. From the 4th task on, the number of blocks that could be used in a program was limited, forcing learners to shorten code and use the loop.

2nd World: Animation

The other developed template was the so-called “World of Film”, in which a sprite is programmed to change its position and size over time. The sprite has four parameters: position X, position Y, size and rotation (as opposed to one basic direction). The programming language has one basic command Sprite, which draws a sprite on the game

board in the place given by the parameters of X, Y positions, its size given by a parameter and rotated by a given number of degrees. This command was supplemented with a block for creating mathematical expressions with basic arithmetic operations and the If structure controlling the time condition (e.g. whether time has exceeded a certain value).

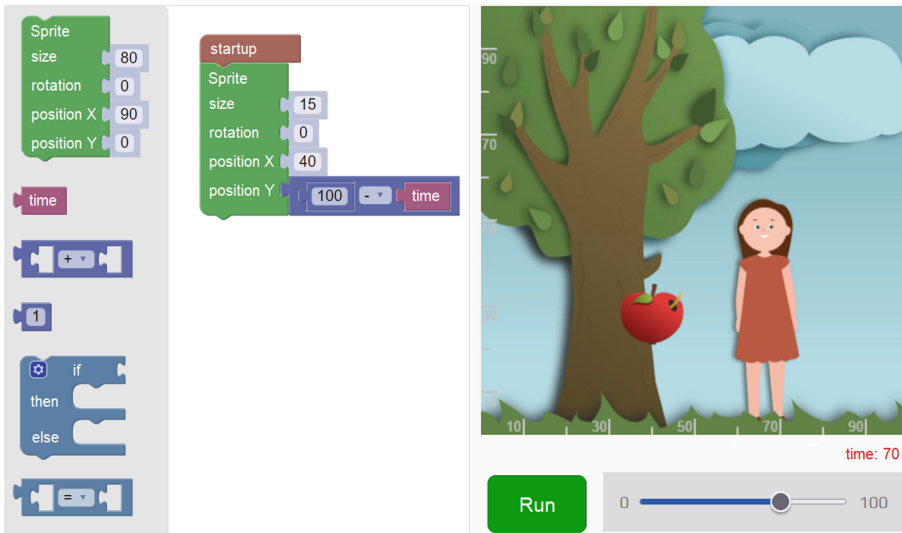


Fig. 2. The World of Film. Learners are assigned to animate an apple falling from a tree (the situation in the picture having a time value of 70). The correct solution to the task can be seen in the middle, the time variable having been used in the expression.

Animation is carried out in such a way that when the program is run, the time variable continuously changes its value from 0 to 100 and the Sprite command is performed for each of these values. The time variable can be used as a parameter in a command. If the value of the X position parameter is set equal to the time variable, the X position will continuously change from 0 to 100 and the sprite will move uniformly from left to right over the whole game board.

The learner is assigned the programming task by running animation of the programmed sprite's shadow. The learner's task is to create a program (i.e. assemble parameters of the Sprite block) to make his/her sprite behave in the same way as its shadow, i.e. both objects should overlap each other throughout the animation (see Fig. 2). The learner has the possibility of running the program repeatedly, animation having the time value of 0 to 100. He/she can also use a scroll bar to manually set any value of the time variable and analyze the situation at a given time (see Fig. 3).

Whereas the didactical aim of the world of Karel the robot is to get fluent with loops, the world of Film aims to understand procedures with parameters. The world of Film is based on the parameter concept, working with the variable and primarily with expressions. The method of programming in this world is close to functional programming. Consisting of more complicated concepts than the world of Karel the robot, it is more suitable for learners at high school or in their final years of lower secondary school.



Fig. 3. Phased animation of a task requiring a spaceship to land on a planet in time values of 0, 25, 50, 75, 100, showing the shadow of the planet getting closer and bigger over time.

In this world, learners were typically assigned to move a sprite horizontally or vertically in uniform motion (motion from right to left is more complicated than motion from left to right due to position becoming smaller as against time), to move it around more slowly and more quickly, to combine motion in those directions, to make the sprite grow or shrink over time and to combine growth with the motion of the sprite. In this world, we created a set of programming tasks which gradually increase in difficulty. More simple tasks include placing a sprite in a specific position in the coordinate system or increasing one of its coordinates in relation to time. More complicated tasks include the use of expressions to decrease one of the sprite's coordinates or its size as time increases, the combination of several parameters dependent on time (e.g. motion on the diagonal or simultaneous motion and shrinking of a sprite). The most difficult tasks combined several motions over time (e.g. motion there and back during one animation), applying decision-making.

2.2 Evaluation

The created environment for assembling programs from blocks was implemented as a module in the Czech edition of Bebras Challenge. We used the created tasks in two ways.

As programming is not a compulsory part of Czech Informatics curricula, we supposed that many pupils have no programming skills. Thus, we created a special set of tasks called Blocks which contained tasks from the world of Karel the robot. We offered this set of tasks to schools as preparation for the national round of the contest. During September and October 2021, this test of 11 questions was taken by 45 000 learners at lower secondary and high schools. Having examined findings drawn from feedback from schools and from a consulting expert's review, we made improvements to the environment and tasks. Problems with graphics not working properly in some tasks in some browsers were most common. There were also reports of difficulties in transition between task 4 and 6, caused by a very large cognitive step. We solved this problem by inserting another task 5 and adding explaining elements to the task questions.

The national round of the Bebras Challenge was another iteration for verification. Each of the 109 442 contestants worked on 3 completely new tasks from the world of Karel the robot (the total number of tasks being 12). Tasks in older age categories were based on more complex algorithmic situations. This iteration allowed us to determine to what extent these new tasks are more difficult than other algorithmic tasks and to what extent they are more difficult than the average task (see Results for further details).

Verification of the world of Film was also carried out in two iterations, despite fewer contestants having participated. During January and February 2022, a set of 12 tasks of this type served as a practice set for contestants that had qualified for the central round in the category for the oldest pupils. 546 learners worked on this set of tasks.

The second iteration was carried out in the central round itself, which 358 contestants took part in. The test was made up of 15 tasks, 3 of which were from the world of Film (one of them is shown in Fig. 4) and another 3 from the world of Karel the robot. It means that 40% of the tasks involved programming by assembling programming code from blocks. Verification showed that these tasks can be used at such a high level as the central round of a nationwide contest.

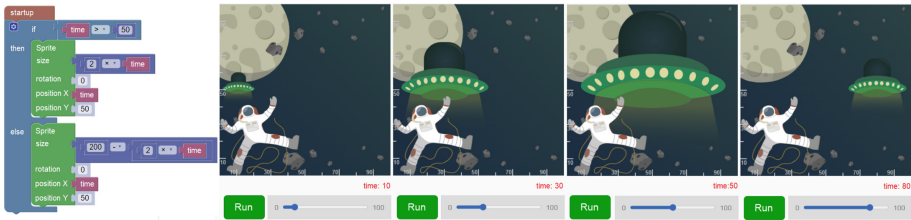


Fig. 4. A complex task where the sprite first moves closer and then moves away, taken from the central round of the contest, using program branching depending on the time parameter. The correct solution being on the left, its phased animation is on the right in time values of 10, 30, 50 and 80 (at time 0 and 100, the flying saucer measured zero).

Following verification, we had to improve the method of the learner's data evaluation while solving a task. There were deviations in computations when using parameters in combination with multiplication and division (e.g. multiplication by 0.001 and division by 1000 did not give the same result), which, in rare cases, led to incorrect evaluations of learners' solutions.

It also emerged that rotating a sprite visually by angular degrees is not optimal as it had rotated the sprite by 100° by the end of the animation. Learners can easily recognize a 90° rotation but for the sprite to rotate by 90° during a time interval of 100, it would have to be rotated in grades rather than in angular degrees, i.e. in units that learners are not acquainted with in schools. For that reason, we finally decided not to use the rotation parameter in tasks.

3 Results

In order to compare the difficulty of programming tasks involving assembling blocks as against other algorithmic task, we worked with proportion of contestants that had been able to solve a task, which is the factor for describing task difficulty [27]. While devising tests for the Challenge, we made efforts to create programming tasks which difficulty coincided with the overall difficulty level of algorithmic tasks. A verification process was used to ascertain whether we had managed to do so.

First, we used the Anderson-Darling test to ascertain whether success rates for programming tasks and other algorithmic tasks are normally distributed. The hypothesis for testing normality of data was rejected at a significance level of 0.05. We then tested the equality of variances of both samples. At the level of 0.05 the null hypothesis was not rejected. Therefore, it can be claimed that variances of both samples are equal.

We subsequently used the two-sided non-parametric Wilcoxon test to ascertain whether the means of both samples are equal. Since the null hypothesis was rejected at a significance level of 0.05, we used the one-sided non-parametric Wilcoxon test. This enabled us to verify whether the mean of the success rate for the programming tasks was equal to or lower than for the other algorithmic tasks. As this hypothesis was rejected at a significance level of 0.05, it can be claimed that the success rate for programming tasks involving assembling blocks is significantly higher than for other algorithmic tasks.

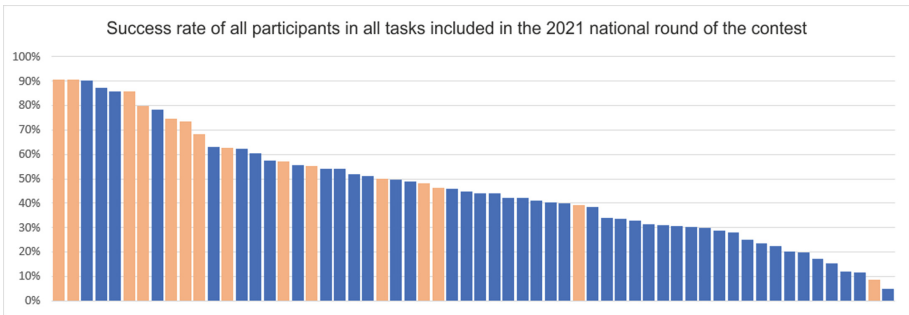


Fig. 5. A comparison of the difficulty level of all tasks included in the 2021 national round of the contest; tasks that involve assembling programming code are marked in a light color.

When comparing programming tasks with all contest tasks in this year’s national round, we discovered that, apart from two exceptions, all of these tasks from all age categories were placed in the top half according to the proportion of correct answers (see Fig. 5). In this figure, programming tasks involving assembling blocks are marked in a light color. To confirm a statistically significant difference in the difficulty of programming tasks as against other task, a verification process was used.

First, we used the Anderson-Darling test to ascertain whether success rates for programming tasks and other tasks are normally distributed. The null hypothesis for testing normality of data was not rejected at a significance level of 0.05. We then tested the equality of variances of both samples. At a significance level of 0.05 the null hypothesis was not rejected. Therefore, it can be claimed that variances of both samples are equal. We subsequently used the two-sided Student’s t-test to ascertain whether the means of both samples are equal. Since the null hypothesis was rejected at a significance level of 0.05, we used the one-sided Student’s t-test. This enabled us to verify whether the mean of the success rate for the programming tasks was equal to or lower than for the other tasks. As this null hypothesis was rejected at a significance level of 0.05, it can be claimed that the success rate for programming tasks is significantly higher than for other tasks.

It means that programming tasks involving assembling blocks can be declared as being demonstrably easier for learners. This may be due to the fact that the tasks provided feedback and learners could have several attempts to iterate their answers, unlike in regular Bebras tasks, where they click on objects or answer multiple choice questions without receiving any feedback. The attractiveness of this new type of tasks is another factor that has to be taken into account.

In November 2021 we asked 939 school coordinators who were responsible for organizing the contest in their school to fill in a questionnaire. 199 replies were received, representing a 20% rate of return. Therefore, it can be regarded as a statistically representative sample and the views expressed can be taken into consideration.

One of the questions referred to how teachers rated the new type of Bebras tasks, based on assembling a program from blocks. Using the Czech school grading system, almost three quarters of them rated this type of tasks “excellent”, one sixth “very good” and the remaining tenth “good”, “satisfactory” or “unsatisfactory”.

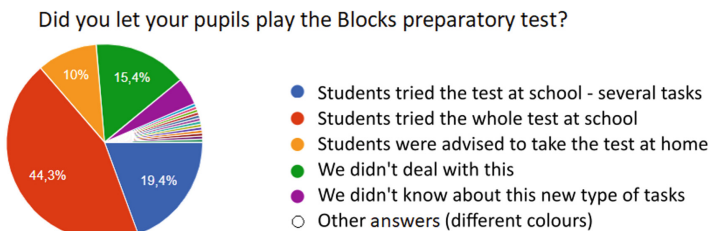


Fig. 6. Results of the poll question asking teachers whether and how they gave their learners the opportunity to play the Blocks preparatory test in the world of Karel the robot.

Another question inquired into the extent teachers used their lessons to give learners the opportunity to work on the type of tasks used in the world of Karel the robot. Results show that almost two thirds of schools involved in the poll had given pupils the opportunity to prepare for the contest during lessons (for details see Fig. 6). Schools can therefore be regarded as perceiving this type of tasks to be an appropriate innovation to the curriculum for advancing learners’ informatics skills.

4 Conclusion

We introduced a new type of informatics tasks into the Czech edition of the Bebras Challenge, not having previously been used in that informatics contest. The tasks involve the use of blocks to assemble programming code. Considering the potential of the block environment, such tasks are of value both from a motivational and a pedagogical respect. Apart from their significance in the Bebras Challenge, they will also be of vital importance in the teaching of computing in primary and secondary schools. The developed tool can be used to create and test sets of tasks focusing on one concept or one programming skill, with the possibility of later implementing them into the school curriculum. In the future the developed module can be enhanced by adding more “worlds” such as turtle graphics or the world of Baltie the magician.

Being able to continuously save created programs, the environment allows monitoring of the way a learner deals with a programming task or the way he/she progresses in a set of tasks. That will enable future research to examine the programming mistakes a beginner might make, what kind of instructions might contribute to or eliminate their occurrence or to reveal misconceptions that might prevent beginners from solving programming tasks in a block environment. This could be useful for future compilation of the programming curriculum for beginners, where appropriate tasks could be chosen either to prevent typical mistakes or, contrarily, to lead a learner into making them, allowing the potential of failure to be used to develop a learner's understanding.

Acknowledgement. The research was supported by the project TAČR TL03000222 “Development of computational thinking by situational algorithmic problems”.

References

1. Gander, W.: informatics and general education. In: Gülbahar, Y., Karataş, E. (eds.) ISSEP 2014. LNCS, vol. 8730, pp. 1–7. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09958-3_1
2. The royal society: shut down or restart? The Way Forward for Computing in UK Schools. The Royal Society, London (2012). https://royalsociety.org/~media/royal_society_content/education/policy/computing-in-schools/2012-01-12-computing-in-schools.pdf
3. K-12 Computer Science Framework Steering Committee: K-12 Computer Science Framework. ACM, New York, NY (2016). <https://dl.acm.org/doi/book/10.1145/3079760>
4. CSTA: K-12 Computer Science Standards (2011)
5. Blaho, A.: Informatika v štátnom vzdelávacom programe (Informatics in a state educational programme). In: Kalaš, I. (ed.) DidInfo 2012, pp. 7–14. Matej Bel University, Banská Bystrica (2012). http://www.didinfo.net/images/DidInfo/files/didinfo_2012.pdf
6. Sysło, M.M., Kwiatkowska, A.B.: Introducing a new computer science curriculum for all school levels in Poland. In: Brodnik, A., Vahrenhold, J. (eds.) ISSEP 2015. LNCS, vol. 9378, pp. 141–154. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25396-1_13
7. Wing, J.M.: Computational thinking. *Commun. ACM* **49**(3), 33–35 (2006). <https://doi.org/10.1145/1118178.1118215>
8. Ministry of education, youth and sports of the Czech Republic: Strategie digitálního vzdělávání (Strategy of digital education). Ministry of education, youth and sports of the Czech Republic, Praha (2014). <https://www.msmt.cz/uploads/DigiStrategie.pdf>
9. Ministry of education, youth and sports of the Czech Republic: Rámcový vzdělávací program pro základní vzdělávání (Frame educational programme for basic education – basic version). Ministry of Education, Youth and Sports of the Czech Republic, Praha (2021). <https://www.edu.cz/wp-content/uploads/2021/07/RVP-ZV-2021.pdf>
10. Wing, J.M.: Computational thinking: what and why? Carnegie Mellon University, Pittsburgh (2010). <https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf>
11. Wittmann, E.H.: Developing mathematics education in a systemic process. *Educ. Stud. Math.* **48**(1), 1–20 (2001). <https://www.jstor.org/stable/3483113>
12. Xia, B.S.: A pedagogical review of programming education research: what have we learned. *Int. J. Online Pedagog. Course Des.* **7**(1), 33–42 (2017). <https://doi.org/10.4018/IJOPCD.2017010103>

13. Ackermann, E.: Constructivism(s): shared roots, crossed paths, multiple legacies. In: Clayson, J.E., Kalaš I. (eds.) *Constructionism 2010: Constructionist Approaches to Creative Learning, Thinking and Education: Lessons for the 21st Century: Proceedings for Constructionism 2010*. Comenius University, Bratislava (2010)
14. Ackermann, E.: Piaget's constructivism, Papert's constructionism: what's the difference? (2001). http://learning.media.mit.edu/content/publications/EA.Piaget%20_%20Papert.pdf
15. Jonassen, D.H.: Objectivism versus constructivism: do we need a new philosophical paradigm? *Educ. Tech. Res. Dev.* **39**, 5–14 (1991). <https://doi.org/10.1007/BF02296434>
16. Kalaš, I.: *UCL Scratchmaths curriculum*. University College London, London (2017). <http://www.ucl.ac.uk/ioe/research/projects/scratchmaths/curriculum-materials>
17. *The LEAD Project: Easy LEAD: Super Scratch programming adventure!* No Starch Press, San Francisco (2012)
18. Chao, P.-Y.: Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Comput. Educ.* **95**, 202–215 (2016). <https://doi.org/10.1016/j.compedu.2016.01.010>
19. Dagiene, V.: The bebras contest on informatics and computer literacy – students drive to science education. In: *Joint Open and Working IFIP Conference, ICT and Learning for the Net Generation*, pp. 214–223. Kuala Lumpur (2008). <https://www.bebas.org/sites/default/files/documents/publications/DagieneV-2008.pdf>
20. Bebras Challenge. <https://www.bebas.org/>
21. Bobřík informatiky (Beaver of Informatics). <https://www.ibobr.cz/english-uk>
22. Berki, J., Drábková, J.: *Základy informatiky pro 1. stupeň ZŠ (Basic of informatics for primary school)*. Textbook. Technical University of Liberec, Liberec (2020). <https://imysleni.cz/ucebnice/zaklady-informatiky-pro-1-stupen-zs>
23. Trna, J.: Konstruktivní výzkum (design-based research) v přírodovědných didaktikách. *Scientia in educatione.* **2**(1), 3–14 (2011). <https://ojs.cuni.cz/scied/article/view/11/12>
24. Blockly. <https://developers.google.com/blockly>
25. Hour of code. <https://hourofcode.com/>
26. Pattis, R.E.: *Karel the Robot: Gentle Introduction to the Art of Programming with Pascal*. Wiley, Hoboken (1981)
27. Vaníček, J., Šimandl, V.: Participants' perception of tasks in an informatics contest. In: Kori, K., Laanpere, M. (eds.) *ISSEP 2020. LNCS*, vol. 12518, pp. 55–65. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63212-0_5