Computational Thinking through Geometric Understanding: A Case Study on Programming in Mathematics Education

Trygve K. Løken[1][0000-0001-6883-0973]

[1] Department of Mathematics and Natural Science, Nord University, Norway
trygve.k.loken@nord.no

Abstract. According to the 2020-updated curriculum for the Norwegian lower middle school, algorithmic thinking and computer programming are necessary student skills that are to be implemented in the mathematics education. Despite the curriculum update, many mathematics teachers lack competence and experience in computer programming, and do indeed struggle to integrate algorithmic thinking in their teaching. Therefore, there is a need for specific teacher resources, such as simple but efficient readily developed education programs that can easily be adapted, extended, and applied, either as whole or as inspiration for teachers in their own classrooms. The contribution of this article is twofold: Firstly, we present an education program that aims to introduce basic programming concepts, such as loops and logical conditions, to lower middle school students through their existing understanding of geometry. Secondly, we report on a pilot study where the education program was taught by three teachers in their respective classes over a period of three weeks in a Norwegian lower middle school. The students' development in algorithmic thinking strategies and problem solving skills over the course of the education program was assessed with a generalized pre- and post-test. The results indicate that students with low to medium test score in the pre-test achieved the greatest improvement (approximately 100-300% increase in test score), whereas students with medium to high test score in the pre-test performed slightly worse after the education program.

Keywords: Algorithmic Thinking, Computer Programming, Mathematics Education, Geometry.

1 Introduction

Programming is becoming an increasingly important skill for young students when facing modern society problems [1]. Computational thinking is related to problem solving in an algorithmic manner, i.e., defining problems and breaking them up into smaller parts that are solved stepwise, much like the structure and purpose of a computer code [2, 3]. Several studies argue that programming can motivate students to study mathematics and increase their problem solving skills [4, 5]. The need for including computational thinking in school curricula has led to the introduction of programming in elementary school in many countries, either as a separate subject or particularly as

integrated in mathematics or science education [1, 6, 7]. Norway is a part of this international trend and computational thinking was included in the primary and lower secondary school mathematics education after the curriculum renewal in 2020 [8, 9]. However, many teachers lack the proper competence and experience in computer programming and may struggle to implement computational thinking in their teaching.

Seymour Papert introduced a new way of working with geometry in the 1980s, which was called Turtle Geometry [10, 11]. Papert and his colleagues developed the programming language Logo, which has been used frequently and widespread in schools since the 1980s [12]. Logo is adapted to the use of children, as a mean for young students to explore advanced geometrical figures and patterns, through the control of a physical turtle robot, which could be programmed to move around on a piece of paper to create drawings. Students could relate their body motion to the motion of the robot and transfer this knowledge into the work of learning formal geometry, including the conceptualization of angles.

In this paper, we propose a teaching strategy for computational thinking linked to geometric understanding and drawing of geometric figures via block-based programming. The developed education program is inspired by Turtle geometry, although it has a somewhat inversed approach. Rather than using programming as a mean to explore geometry, we propose to use familiar geometric figures to learn about programming and computational thinking. Teachers may apply the education program as an entry to basic programming concepts through geometrical understanding already possessed by the students, before introducing formal, line-based programming. The developed teaching program was tested in a pilot study, which consisted of three weeks of lecturing in three lower secondary classes. The programming was performed in Edublocks (directly in the browser), which allows for freely available block- and line-based programming in parallel, and it offers several languages, e.g., python [13]. Python was applied in the education program because it is open source, freely available, and it has a Turtle library [14].

This article presents the developed education program and reports on the students' development in programming and algorithmic thinking after participation in the education program, which was assessed with a generalized pre- and post-test. The research questions these contributions attempt to answer, are:

- RQ1: How can geometric understanding be utilized to learn basic programming concepts?
- RQ2: How does the computational thinking of lower secondary school students evolve after receiving basic programming education?

Related to RQ2, we use the definition of computational thinking proposed by [15], namely that it is a focused approach to problem solving, incorporating thought processes that utilize abstraction, decomposition, algorithmic design, evaluation, and generalizations.

2 Method

This article reports on a pilot study, where a newly developed education program, designed for integrating programing in mathematics education, was carried out in a lower secondary school in Norway. The research team consisted of three experienced mathematics teachers, Øyvind, Olav, and Anders, who taught a grade 8 class (13-14 years old), a grade 9 class (14-15 years old), and a grade 10 class (15-16 years old), respectively, and one researcher, the author of this article. The education program was taught by the teachers in their respective classes over a period of three weeks, with approximately two to three lecturing hours on average per week. The research team had weekly project meetings where the content of the education program was assessed and improved based on the classroom experiences from the previous week. The students had mixed (i.e., from none to some) programming experience, for example, from primary school programming introduction or from elective lower middle school programming courses.

Data were acquired from the weekly project meetings, which were audio recorded, from classroom observations, and from students' written material in the form of hand-ins. In the students' written material, the students are anonymized, and data has only been included in the study from those students who have given their written consent. This article mainly reports on the student hand-ins from grade 8 and 9. In this section, the education program and the tests for assessing the students' ability of computational thinking are presented.

2.1 Education Program

A short presentation of the education program with emphasis on different geometrical figures and how they are suitable to demonstrate specific programming concepts, follows in this subsection. The education program consisted of presentations given by the teachers, typically in the beginning of each lecture and when introducing new concepts to the class, and group and individual student work where the students were allowed to assimilate their newly acquired knowledge through exploration, discussion among peers, and guidance from the teacher.

Introduction. After giving the students information about the research project and the attached education program, the teachers started the first lecture by guiding the students through the process of setting up a new project in Edublocks. Thereafter, they explained the Turtle library, and that this library contains commands to move a robot object (an arrow by default) around the screen, and that a set of consecutive commands would eventually constitute a computer code, which could draw a desired geometrical figure on a virtual canvas when executed. At this stage, the students were allowed to make themselves familiar with the move forward and rotate commands, and actually experience that the arrow behaved accordingly.

The teachers instructed the students to work in two different tabs; one where they were to copy the teachers code with minor modifications, and the other to work in their own pace. The former pane had a twofold purpose; on the one hand, it served as a "safe

haven" to which the students could always return if they felt insecure about creating their own code, and on the other hand, it ensured that the students had an operative code where they could add new elements presented by the teacher, in case their own exploration drifted off from the course of the teacher or caused the program to crash. The latter pane served as a playground where the students could try out new ideas or advance if they felt that the teacher's progress was too slow. The teachers' presentations were given very slowly and thoroughly to ensure that all the students could follow.

The Square. The first geometric figure was the square, and the teachers initiated a discussion on how a square is defined and what commands they could use to make the arrow draw a square. After establishing that one could, for example, move forward the length of one side, turn 90° in one direction, move forward, turn, and so on, until the square is drawn, the classes tried to formalize this proposition into formal computer code.

After this was achieved, the teachers initiated a discussion on patter recognition and repetition and used this as an entry to introduce the for-loop. The previously coded square is an example that illustrates the repetition of the same commands. In a demonstration, the teachers showed how a for-loop could be used to draw the same square by omitting several lines of code, and pointed out that programing is about formulating commands as compressed and concise as possible. Figure 1 shows an example of how the two different approaches can be represented in Edublocks. Note that the code containing the for-loop is significantly shorter.

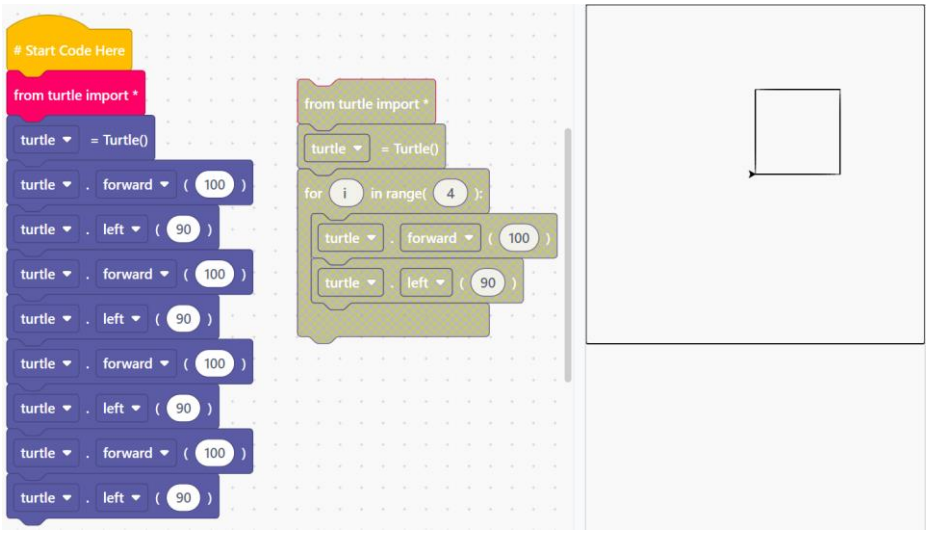


Fig. 1. Two examples of block-based code that produces a square with 100 steps side lengths. The code to the left repeats all the commands, while the code to the right utilizes a for-loop to repeat the forward and turn command four times. The window to the right shows the resulting square. Source: *Edublocks.org.*

The Rectangle. In the second week of the education program, the teachers started to introduce logical conditions through the rectangle. As with the square, the teachers encouraged the students to define a rectangle, and how a drawing of a rectangle may be formalized with code. A rectangle consists, like a square, of a repetition of straight lines and right angles. However, since the length of the side lines are alternating in pairs, the forward and turn commands cannot be repeated four times in a simple for-loop. An easy but not so elegant way of obtaining a rectangle is to write out all the sequential forward and turn commands, i.e., with a brute force strategy. If one would like to solve this by using a four-times repeating for-loop, this can be done by utilizing an if-else statement that checks if the loop iteration is an odd or even number, and apply different side lengths accordingly. The teachers demonstrated the latter strategy as an entry to the topic of logical conditions. Figure 2 shows an example of how the two different approaches can be represented in Edublocks.

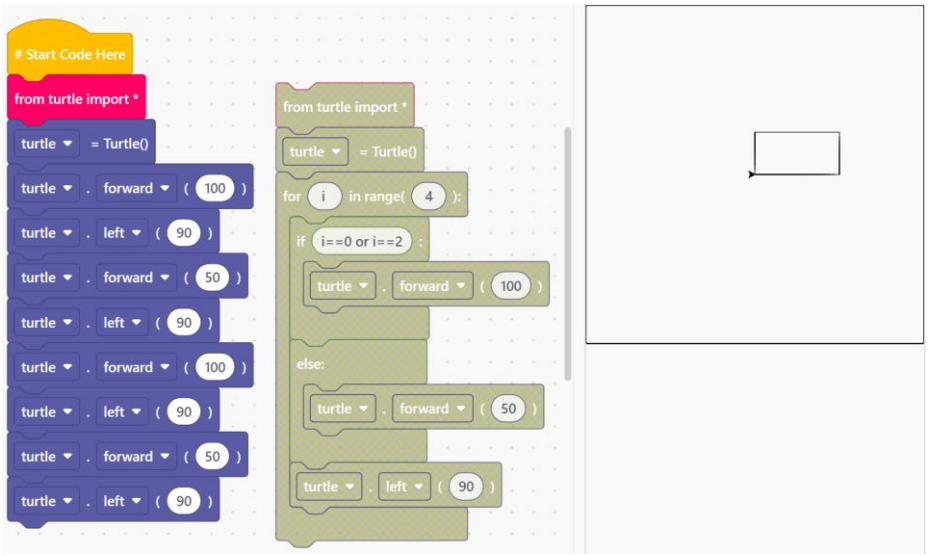


Fig. 2. Two examples of block-based code that produces a rectangle with 100 and 50 steps side lengths. The code to the left repeats all the commands, while the code to the right utilizes a for-loop to repeat the forward and turn command four times and an if-else-statement to alternate between long and short side lengths. The window to the right shows the resulting square. Source: *Edublocks.org.*

Open Tasks. In the third week of the education program, the teachers demonstrated the use of nested for-loops to create repeating figures, e.g., a set of increasing squares placed outside of each other to illustrate similar figures. After this concept was practiced with a few examples, the students now possessed three powerful tools for creating geometrical figures and patterns, namely logical statements and single and nested for-loops. Next, the students were encouraged to design their own code that would create a geometrical figure of their choice. The task was presented as a programming competition, where creativity and concise coding would be rewarded with a price to the winner,

as an attempt to motivate the students. After the students had made their own code, the codes were swapped among the students, and they were instructed to interpret and draw the outcome of a fellow student's code.

Programming is very suitable for investigating how the conditions in geometrical problems affect the solution. For example, by changing one or two variables in the square-code, such as the number of iterations combined with an increasing side length, the result is suddenly a square spiral. The teachers continuously challenged the advanced students by expanding the given tasks. For example, if a student easily managed to make the square, the teacher could encourage the students to make a pentagon, an octagon, or even a generalized code to make any desired regular polygon.

2.2 Pre- and Post-Test

The research team designed a written task with the objective of testing the students ability of computational thinking. The students were given the identical task before and after the education program (i.e., a pre- and a post-test, respectively) and had to solve it individually without receiving help from the teachers. The test was called Robot control (see Appendix for details), and as the name suggests, the students were to give specific commands to make a robot move in a given pattern. The task requires algorithmic thinking, as one has to give sequential commands while imagining how the robot moves around. The test requires no programming knowledge or specific language syntax. On the one hand, there are some similarities between the education program and the test as geometrical figures are involved. On the other hand, the test is about moving a physical robot with written commands, while the education program is about moving an arrow around the screen with code blocks. Thus, the idea is that the students should solve the post-test with their acquired algorithmic thinking skills without directly associating it with the education program.

More specifically, the test consisted of two tasks. In the first task, the robot is supposed to walk along a square with side length seven steps. In the second task, the robot should walk along three squares with side length three, five, and seven steps. The three squares are placed next to each other with some empty space between them, and their lower left corners are marked with the points A, B, and C, respectively (see Appendix for details). The students were allowed to give the robot the following commands; walk forward x steps (where x is a number of their choice), turn x degrees to the right or left, and jump to a specified point. They were also instructed to give the commands as simple as possible, i.e., with as little text as possible, and if they saw a repeating pattern in their commands, they could mark these commands with a star and omit the repeating text.

A set of four assessment criteria for the test was decided by the research team:

1. Can a human understand what kind of figures the student has instructed the robot to make?
2. Can a machine reproduce the correct figures (if the right syntax and setup is neglected)?
3. Are repeated commands omitted and marked with a star? There are two levels of repetition; firstly, the forward and turn commands may be repeated

four times to create a square, and secondly, the squares may be repeated three times with increasing side lengths in the second task.
4. Does the student apply the jump command to make the robot move between the squares?

Criteria 1-2 are inspired by Wing's description of computational thinking as presenting problems and solutions in such a manner that humans and/or machines can execute the computing processes [16]. The tests were graded according to the criteria listed in Table 1, and a maximum number of eight points could be obtained in each test.

Table 1. Assessment criteria for the tests with maximum points indicated.

Criterium	Points	Comments
1. Explain human	2	-0.25 points for each missing side (task 1-2).
2. Explain machine	2	-0.25 points for excessive text, -0.25 points for repeating single commands (e.g., 4 forward, then 4 turn), and -0.25 points for not specifying four repetitions (task 1-2).
3. Repeat	3	2 points for single repetition (task 1-2) and 1 point for double repetition (task 2).
4. Jump	1	Jump between squares (task 2).

3 Results

Out of the three participating classes, grade 10 was affected by several external disturbances during the three weeks of lectures, such as an absent teacher due to illness, exam preparations, and mandatory courses for graduate students, which conflicted with the completion of the education program. Therefore, the test results from the 10th grade students are not considered in this study. However, the remaining two classes carried out the planned education program, and the students' results from the pre- and post-tests are presented in this section.

3.1 Quantitative Perspectives

Among the approximately 25 students in each class, 14 and 12 students gave their consent to participate in the research project in grade 8 and 9, respectively. In grade 8, the lowest and highest achieved score was (0, 7) and (3, 7) for the pre- and post-test, respectively. In grade 9, the corresponding lowest and highest scores were (1, 6.75) and (1.25, 6.75). Key figures of statistical average and dispersion are given in Table 2. The results of the grade 8 students are more dispersed than those of the grade 9 students in the pre-test, while the situation is opposite in the post-test, although the difference is less prominent here. However, the most striking result is that the grade 8 students improved their results by a 48% increase in test score on average, whereas the grade 9

students did not develop much between the tests, with a 5% decrease in test score on average. Note that one grade 9 student decreased his or her score by 4.75 (from 6 to 1.25) points between the two tests, which is a far greater decrease than any other, which influences the statistical average and dispersion.

Table 2. Test results, where μ and σ symbolize mean and standard deviation, respectively.

Grade	σ_{pre}	σ_{post}	μ_{pre}	μ_{post}	$\mu_{difference}$ [%]
8	2.29	1.15	3.57	5.29	48
9	1.39	1.71	5.38	5.10	-5

In order to investigate the students' different levels of understanding, the students are ordered in groups according to their pre-test results, independent of grade. Four groups with two score points bin size are made, and the students' average score within the group are calculated and compared with the average post-test score of the same students. The results are presented in Fig. 3. The groups, when arranged from lowest to highest score, contained 5, 4, 8 and 9 students, respectively. It is evident that the students with the lowest score in the pre-test achieved the greatest improvement between the two tests (+2.95 points), followed by the students with the second lowest score in the pre-test (+2.56 points). The students with the second highest score obtained almost the same result on average in the two tests (-0.06 points), while the students with the highest score performed slightly worse on average in the post-test (-0.41 points). Note that the latter observation is affected by the outlier mentioned in the previous paragraph, and if the outlier is neglected, there is a slight improvement of +0.13 points on average between the two tests.

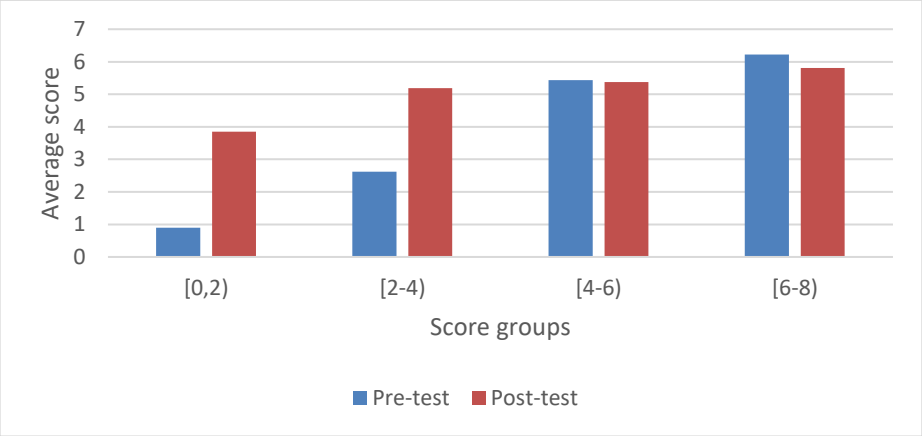


Fig. 3. Grouped, average test results in two score points intervals from the pre-test (blue), and the post-test (red) from the same student groups.

3.2 Qualitative Perspectives

As a general observation, most of the students were able to use the repetition command in the post-test. Some students marked the repetition with a star (as instructed in the assignment), while other wrote explicitly that some commands should be repeated. Both syntaxes were equally rewarded in the assessment process. Use of the repetition command is interpreted as the equivalent to for-loops in the following.

Lowest Score. Hand-ins that were categorized in the lowest score group typically contained uncomplete descriptions of the geometrical figures, such as "Walk forward 7 steps turn 90° to the right and do it all the way around" in task 1. A human may be able to reason that this will eventually form a square, but the "do it all the way around" command is quite diffuse. A machine on the other hand, would not be able to interpret the latter command and would not know when to stop. There are no attempts on loops in this answer, and algorithmic thinking is sparse. Therefore, the following score was given in this example; explain human: 0.5, explain machine: 0.25, and repeat: 0. Other students in the same group gave repeated, direct forward and turn commands that would only form two or three sides of a square. Others again made up or mis-used the allowed commands, such as "Jump 7 steps forward", which can be understood by a human, but not by a machine since jumping was only allowed to a specified point.

These students progressed a lot in terms of algorithmic thinking during the education program and their post-test hand-ins typically contained more or less successful attempts on using single for-loops. In general, their language was more precise in the post-test compared with the pre-test. However, none of them thought about jumping between figures or applying double repetition in task 2.

Second Lowest Score. Students who obtained the second lowest score in the pre-test were in general more precise than the students who obtained the lowest score. They often successfully described the correct geometric figure, at least in task 1, but few attempted to use for-loops. Like the previous group, these students significantly improved their algorithmic thinking, and used single for-loops and an even more precise language in the post-test. For example, one student progressed from "Walk forward, turn 90°" to the more descriptive "Walk 7 steps, turn 90° left". None of the students in this group applied the jump command or a nested loop in task 2.

Second Highest Score. These students mostly used single for-loops in the pre-test. Their answers typically had some minor shortcomings compared with the highest score group, such as not specifying how many times the loop should be repeated, or applying the repetition twice, i.e., after the forward and the turn command ([forward]*, [turn]*) instead of applying the repetition once after both commands were given ([forward, turn]*). None of these students applied the jump command, but a couple of them showed tendences of thinking in the direction of a nested for-loop in task 2, either in the pre- or the post-test. One of the students wrote: "First walk forward 3,5,7 steps then turn 90° to the left and repeat 4 times", and others gave similar answers. Three of the students had a slightly negative development (from -0.5 to -1.5 points) due to untidy

and less accurate second hand-ins compared with the first. For example, one of these marked the repetition correctly in the pre-test but incorrectly in the post test (see the example above).

Highest Score. The students in this group used a precise language when applying single loops, and about half of them applied the jump command. Approximately 20% of the students in the group showed more advanced algorithmic thinking as they produced good attempts on nested loops in either or the pre- and post-test. An example of such an attempt is shown in Fig. 4, which translates: "(*4 times: [Walk 3 steps. Turn 90° to the L]. Jump to next point). Walk 5. Walk 7". A circle is drawn around the commands that are placed inside the round brackets, which we interpret as "repeat this two more times, but replace three with five and seven steps". A better answer would explicitly state that the actions inside the round brackets should be repeated three times. Nevertheless, this answer is still rewarded 1,5 out of 2 possible points for repetition. However, as seen in Fig. 3, the students in this group did not improve on average, but rather achieved a slightly lower score in the post-test.

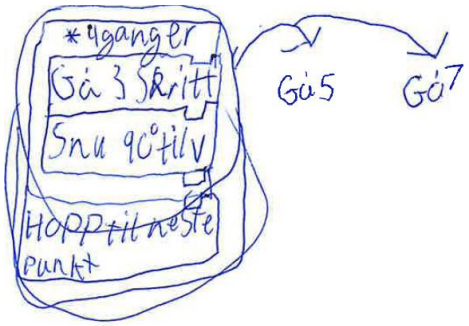


Fig. 4. An example of a students' answer of task 2 in the post-test. The text translates: "*4 times. Walk 3 steps. Turn 90° to the L. Jump to the next point. Walk 5. Walk 7".

4 Discussion

Although the students with low score in the pre-test developed their algorithmic thinking in a positive direction during the education program, the development of the students with high score is unclear. According to the test, these students had a slight decrease in performance after receiving programming lectures. This observation has two possible explanations; either the education was not well adapted to students with some previous programming experience, or the test was not designed well enough to measure improvement of experienced students. In the former scenario, the education program should have lasted longer so that the complexity of the tasks would have increased enough to challenge the experienced students. This observation also argues for a compulsive programming education as a separate subject in Norwegian lower middle schools, to ensure that all students have some experience in algorithmic thinking as a minimum. In the latter scenario, the test should have been designed as a spectrum of

tasks with varying difficulty. However, few of the students who obtained a high score thought in the direction of nested for-loops in task 2, which may indicate that the test was challenging enough for most of the students after all. In retrospect, we should have spent more time on nested for-loops during the education program and also indicated clearer in the assignment text that the repetition command could be used at several levels.

The test was designed with the aim of objectively measuring the students' level of algorithmic thinking without creating too close associations to the research program. It is difficult to assess whether the test was successful regarding this objective, but approximately 1/3 of the grade 8 students gave their answers in the form of drawn blocks that resembled the ones found in Edublocks in the post-test. An example of this can be seen in Fig. 4, which may suggest that the test contained too explicit references to the procedures taught in the class. However, none of the grade 9 students did this. Although the test may have a high validity for the type of tasks to which it is designed to test, it probably lacks applicability to other contexts of computational thinking. In hindsight, the test should have contained a variety of tasks to assess the students ability of abstraction, algorithmic thinking, decomposition and generalization without creating close associations to the lectures and at the same time without discriminating students who lack programming experience. Several tests with these aims are presented in the literature [17, 18], which could be used as a starting point and adapted to the level of lower middle school students. However, there is a limit to how extensive such a test can be before the students at this age lose their patience.

During the education program, the teachers received several questions on why it was necessary to use if-else-statements to create a rectangle. The students argued that it was simpler to apply a series of direct forward and turn commands, i.e., a brute force strategy, and that such a solution would not require more lines of code than a for-loop with an if-else-statement, which is correct deeming from Fig. 2. The rectangle was only used as a familiar entry to logical statements and if-else-statements. Besides, the rectangle is a logical transition from the square, and the students had no problems with learning the concept, and seeing the benefit of loops when working with squares and regular polygons. However, the students had difficulties accepting the purpose of the if-else-statement with regards to the rectangle. Therefore, it would make sense to use other examples to introduce logical statements. For example, if-else statements could be introduced when working with nested for-loops as an entry to geometric similarity, e.g., in an assignment where several similar figures are to be made and one specific figure should be drawn with a different line color. This can only be reasonably achieved with an if-else-statement. Conditionals could also be motivated by including user inputs in the program which can be adapted to user wishes, e.g., whether a square should be filled or not, or whether to draw an equilateral triangle pointing upwards or downwards.

The open task given in the education program was presented as a coding competition, as mentioned in Section 3.1, in an attempt to motivate the students, as competitions has been found to positively affect the learning quality when programming [19, 20]. According to the teachers, many of the students were also motivated and eager to win the price. However, some students expressed that they were stressed and anxious about the competitive element, because someone would then assess their performance and

discover potential errors and mistakes. This observation agrees with the findings of Cheng et al. [21]. The fear of social comparison may have caused these students to go for the simple but safe geometric figure, such as the square, because the teacher had already shown code examples for this figure which made them confident. The competition may be removed from the education program in the future, but the part where the students swap their codes with their classmate to interpret someone else's code is a good exercise in error search and debugging.

5 Conclusions

In this article, an education program designed to enhance algorithmic thinking and programming skills in the context of geometrical figures has been presented, where figures and patterns are created with sequential commands that eventually form a code. The students understood the concept and the necessity of for-loops when working with regular polygons due to the repeating nature of these figures. However, the students found less need for if-else-statements when working with rectangles, as this appeared more cumbersome to them than applying a series of direct commands (brute force strategy).

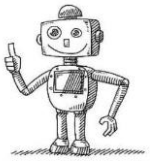
The students development in algorithmic thinking was assessed through a generalized pre- and post-test, which was carried out before and after the lectures. It was found that the students who obtained a low score in the pre-test improved greatly (approximately 100-300% increase in test score) during the education program, and most of them progressed from applying repetitive direct commands to single for-loops and developed a more concise language. On the other hand, the students who obtained a high score in the pre-test had a slight negative development in test score on average, with some exceptions where the students started developing strategies for nested for-loops. Although self-explanatory, we emphasize the importance of challenging the more experienced students in parallel with the basic education of the beginners to stimulate all the students with different programming backgrounds.

Acknowledgements. This project has received funding from Nord University, Norway, through the *Såkornmidler* grant. The author thank Marthe Måløy for contributing with many good ideas and discussions. The author is grateful for the participation of Øyvind Nilsen, Olav Myrvoll, and Anders Solbakken, who invited the project into their respective classrooms.

Appendix – Robot Control

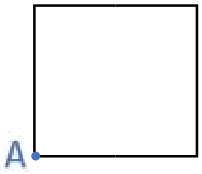
You have been given a robot which you can command to move around on the floor. The only motions it is capable of doing, are:

- walk x (x is a number you can choose) steps forward,
- turn x degrees to the right or left, and
- jump to a point of your choice.

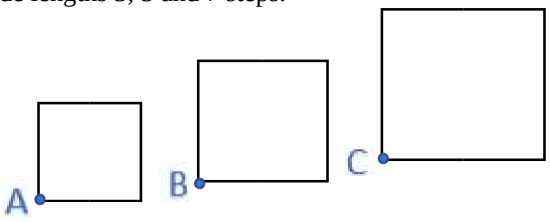


How would you explain to the robot as simple as possible (with as little text as possible) that it should walk along the following lines? The robot always starts in point A. If you see a pattern in its motion, you can tell it to repeat the previous commands that you mark with a star, so that you don't have to repeat the same commands.

1. A square with side length 7 steps.



2. Three squares with side lengths 3, 5 and 7 steps.



References

1. Kaufmann, O.T. and B. Stenseth, *Programming in mathematics education.* International journal of mathematical education in science and technology, 2021. **52**(7): p. 1029-1048.
2. Barr, V. and C. Stephenson, *Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community?* Acm Inroads, 2011. **2**(1): p. 48-54.
3. Berland, M. and U. Wilensky, *Comparing Virtual and Physical Robotics Environments for Supporting Complex Systems and Computational Thinking.* Journal of science education and technology, 2015. **24**(5): p. 628-647.
4. Barak, M. and M. Assal, *Robotics and STEM learning: students' achievements in assignments according to the P3 Task Taxonomy—practice, problem solving, and projects.* International journal of technology and design education, 2016. **28**(1): p. 121-144.
5. Sinclair, N. and M. Patterson, *The Dynamic Geometrisation of Computer Programming.* Mathematical thinking and learning, 2018. **20**(1): p. 54-74.
6. Grover, S. and R. Pea, *Computational Thinking in K—12: A Review of the State of the Field.* Educational researcher, 2013. **42**(1): p. 38-43.
7. Weintrop, D., et al., *Defining Computational Thinking for Mathematics and Science Classrooms.* Journal of science education and technology, 2016. **25**(1): p. 127-147.
8. Brandsæter, A. *Programming in Mathematics Education: An Intermediary between the Real-World and the Mathematical Model*. in *Norwegian ICT conference for research and education*. 2021. Trondheim.

9. The Norwegian Directorate for Education and Training, *Curriculum in Mathematics (MAT01-05)*, M.o.E.a. Research, Editor. 2019.
10. Kynigos, C. and M. Grizioti, *Programming Approaches to Computational Thinking: Integrating Turtle Geometry, Dynamic Manipulation and 3D Space.* Informatics in Education An International Journal, 2018. **17**(2): p. 321-340.
11. Papert, S.A., *Mindstorms: Children, Computers, and Powerful ideas*. 1980, New York: Basic books, Inc., Publishers.
12. Kafai, Y.B. and Q. Burke, *Computer programming goes back to school.* Phi Delta Kappan, 2013. **95**(1): p. 61-65.
13. Pounder, L., *Edublocks: Block party.* Linux Format, 2018(232): p. 58-59.
14. Laura-Ochoa, L. and N. Bedregal-Alpaca, *Incorporation of Computational Thinking Practices to Enhance Learning in a Programming Course.* International journal of advanced computer science & applications, 2022. **13**(2).
15. Selby, C. and J. Woollard, *Computational thinking: the developing definition.* University of Southampton, 2013.
16. Wing, J.M., *Computational thinking.* Commun. ACM, 2006. **49**(3): p. 33–35.
17. Dolgopolovas, V., et al., *Exploration of computational thinking of software engineering novice students based on solving computer science tasks.* The International journal of engineering education, 2016. **32**(3): p. 1107-1116.
18. Lafuente Martínez, M., et al., *Assessing Computational Thinking: Development and Validation of the Algorithmic Thinking Test for Adults.* Journal of Educational Computing Research, 2022: p. 07356331211057819.
19. DeLeeuw, K.E. and R.E. Mayer, *Cognitive consequences of making computer-based learning activities more game-like.* Computers in human behavior, 2011. **27**(5): p. 2011-2016.
20. ter Vrugte, J., et al., *How competition and heterogeneous collaboration interact in prevocational game-based mathematics education.* Computers and education, 2015. **89**: p. 42-52.
21. Cheng, H.N.H., et al., *Equal opportunity tactic: Redesigning and applying competition games in classrooms.* Computers and education, 2009. **53**(3): p. 866-876.