Computer programming in the lower secondary classroom: learning mathematics
Programmare il computer nelle scuole secondarie di primo grado: imparare la matematica

Johan Lie*, Inge Olav Hauge and Tamsin Jillian Meaney

Bergen University College, Norway, johan.lie@hvl.no, inge.olav.hauge@hvl.no, tamsin.jillian.meaney@hvl.no

* corresponding author

HOW TO CITE Lie, J., Hauge, I. O., & Meaney, T. J. (2017). Computer programming in the lower secondary classroom: mathematics learning. *Italian Journal of Educational Technology*, *25*(2), 27-35. doi: 10.17471/2499-4324/911

ABSTRACT In this paper, we study computer programming in relationship to the learning of mathematics, particularly processes such as mathematical thinking and problem solving. Computational thinking and programming requires the learner to be immersed and engaged in a continuously changing, problem-solving process. In our research project, we used the graphical program Scratch, which introduces children to the principles of computer programming and computational thinking. The children worked on open problems that they constructed themselves. Interactions between children and teachers during the project were recorded and analysed. Our findings suggests information on the link between the act of computer programming, problem solving and mathematical competencies.

KEY-WORDS Programming, Mathematics, Digital competencies, Learning, Computational thinking.

SOMMARIO In questo articolo riportiamo i risultati di uno studio sulla relazione tra programmazione e apprendimento della matematica; in particolare, i processi legati al pensiero matematico e alla soluzione di problemi. Il pensiero computazionale e la programmazione richiedono che lo studente sia immerso e impegnato in un processo, in continuo cambiamento, di risoluzione di problemi. Nel nostro progetto di ricerca abbiamo utilizzato l'ambiente di programmazione visivo Scratch per introdurre i principi della programmazione e del pensiero computazionale. Gli studenti hanno lavorato su problemi aperti di loro ideazione. Le loro interazioni con gli insegnanti sono state registrate e analizzate. I risultati puntano nella direzione di un legame tra l'atto di programmare, la soluzione di problemi e le competenze matematiche.

PAROLE CHIAVE Programmazione, Matematica, Competenze digitali, Apprendimento.

1. INTRODUCTION

Computational thinking per se is currently not a part of the Norwegian school curriculum (Utdanningsdirektoratet, 2016). Nevertheless, various calls to strengthen children´s digital competencies are being linked to computer programming and this may be incorporated into future curricula, as has been done elsewhere in the world: see for example Department for Education (2013), Euractiv (2015), and Silcoff (2016). Teaching computer programming in schools in Norway also appears to be in alignment with a report ("Official Norwegian Report No. 8, 2015") studying what learners should be prepared for in the future and how the curriculum should be designed to reach these learning goals. The report focuses on "learning in depth" and "cross-disciplinary teaching" (Departementenes Sikkerhets- Og Serviceorganisasjon, 2015).

The Prime Minister of Norway has contributed to the discussion by taking part in activities such as the conference "Girl Tech Fest", where the importance of programming skills for girls was advocated. Children have many opportunities to engage in programming outside the education system, for example through the after-school activity club "La kidsa kode" ("Let Kids Code"), which are being established all over the country. These clubs are popular and interesting, but have not yet been investigated in depth. More recently, the "VilVite" ("Want to know") Science Centre has initiated a research project that investigates children learning algebra while programming Arduinos[1].

The Norwegian Ministry of Education has opened up the possibility for schools to try out "computer programming" as a non-compulsory course in secondary school (learners aged 13-16). This has currently resulted in 153 schools providing a course in programming during the 2016-17 academic year. Modelling, coding and "basic skills" are highlighted as important principles in the course (Utdanningsdirektoratet, 2016a). "Digital skills" are considered as one of the five basic skills (reading, writing, computation, oral skills and digital skills) but can also be seen as an integral part of the other four skills. We interpret digital skills in a broad sense, which includes creative and entrepreneurial skills just as much as technical skills.

In this paper, we document our involvement in a pilot study in programming, where the programming language Scratch is used in the context of mathematical thinking and problem solving. As part of our research project we taught computer programming in a mathematics classroom at the lower secondary level (45 learners aged 10-11) and studied the students' learning processes from the standpoint of Niss' mathematical competences (Niss & Jensen, 2002), Sfard's description of mathematical understanding and thinking (Sfard, 1991), as well as Wing's concept of computational thinking (Wing, 2006). Sfard describes mathematical understanding through the dichotomies of structural thinking and operational thinking, describing this dichotomy as "two sides of the same coin" (Sfard, 1991). Structural thinking is explained as the ability to see mathematical conceptions as mental objects, i.e. something that is not available to our senses. This is a "static" understanding of a mathematical concept. This can be seen in contrast with going through processes, i.e. operational understanding can be dynamic, sequential and detailed. In programming activities in Scratch, learners work in a social environment that is likely to help their development of both operational and structural understanding of the underlying mathematical concepts. On the other hand, Niss discusses how mathematical learning can be categorized into a number of different but interrelated mathematical competencies like communication competence, reasoning competence, problem-solving competence, modelling competence, symbol and formalism competence and representation competence. Learners need to develop all these competences in order to gain a deep and working understanding of mathematics (Niss & Jensen, 2002). Niss' model has been instrumental in the development of the current curriculum in mathematics in Norway (Kunnskapsdepartementet, 2006). It has been criticized, however, for not including the emotional aspects of mathematical learning, and Kilpatrick's five strands for mathematical proficiency are sometimes mentioned as an alternative model for mathematics learning (National Research Council, 2001). We also find that our work is closely connected to computational thinking as understood by Wing, as learners need to think computationally to solve mathematical problems via computer programming (Wing,

[1] www.vilvite.no

2006). The term "computational thinking" has a larger meaning than solely computer programming (Wing, 2006). As she points out, we may think computationally without necessarily programming a computer. This is also one of the main themes of a related long-term ongoing project called "Computer Science Unplugged", where children learn computational concepts in a non-computer environment (Bell, Rosamond, & Casey, 2012). Therefore, we consider computer programming as one of many aids for computational and mathematical thinking, following Wing: "Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability." (Wing, 2006, p. 33.)

2. COMPUTER PROGRAMMING AND MATHEMATICAL THINKING

To gain a deep understanding of the process of computer programming, the programmer is required to develop a certain experience of mathematical thinking. From a historical point of view, computers were constructed to perform mathematical computations. In particular, algorithmic constructs such as flow statements (loops and if-then-or-else, for, until, etc.), variables, types of variables etc. are deeply connected to arithmetical, mathematical and computational thinking. In itself, this might provide us with a possible overlap in the learning of programming and the learning of mathematics and computation. It is our belief that learning a programming language might be enhanced when the student is able to connect her/his work in computer programming to relevant development of the corresponding mathematical language and concepts. We study how the converse situation could also be true: working with computational thinking through computer programming might also enhance the learning of mathematics. In our project, we have sought to study this overlap of learning. Several tutoring situations are actualized in the process of computer programming in a social environment. Examples of situations include cases where the student does not get the "expected" response from the computer program. Several interactions might emerge, including teacher-student, student-student, and student-computer interactions, as well as combinations of these. In this article, we analyse student-teacher interactions in the piloting phase of our project.

2.1. *Scratch – animation through computer programming*

Most environments for beginners in computer programming concentrate on visual elements of the programming process. However, there is considerable variance both between environments and within environments, from purely visual to a combination of visual and textual input from the programmer. From the viewpoint of abstract representations and more concrete representations, using visual representations can be considered as working with half-abstracts or iconic representations in favour of more abstract symbolical representations (Bruner, 1966; Tall, 1994).

A number of more or less complete programming environments have been designed with the aspiring (child) programmer in mind, including Greenfoot (Kölling, 2010), Scratch (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010), Lightbot (Yaroslavski, 2014). Scratch is a free programming language that allows children to build animations. It was developed by the Media Laboratory at the Massachusetts Institute of Technology and was launched in 2007 (Resnick et al., 2009). Scratch relies on both visual and textual input from the programmer, but places a strong emphasis on the visual input. From the very outset of the project we decided to use Scratch as the programming tool, mainly because of its visual, low-abstract interface, but also since Scratch seems to be a frequently used tool in Norwegian schools.

Commands are not written as plain text in Scratch, but are rather picked from intuitive categories and fitted together almost like Lego. In this way, programming in Scratch avoids many of the technical difficulties that are inherently present in traditional programming languages like Java, C, C++, Python, Fortran, etc. For example, picking wrong types of variables (e.g. floats instead of integers) is simply not possible in the Scratch programming language; you cannot use a variable unless it has the correct footprint. When programming in Scratch, the learner can therefore work at a lower abstraction level than when using fully text-based programming languages. Programming statements, objects and variables are picked from a toolbox, and a program is built brick by brick, almost like Lego. This makes a strong connection between abstract mathematical objects and visual representation of the object. Objects that do not suit each other will not fit together in the programming environment. This does come at the cost of not using a "proper" real world programming language like Java, Python or any variant of C; these have much more general applicability but also a much steeper learning curve. The underlying structures (which are strongly connected to computational thinking) are, however , the same in most programming languages. Therefore, we consider using Scratch to be a useful first step for the aspiring computer programmer.

3. METHODOLOGY

During a period of three weeks, for approximately four hour each week, the learners were first introduced to Scratch as a programming tool, and then used it for problem solving in the computer lab at their school. Their two teachers and the three researchers acted as facilitators while students worked on open, explorative problem-solving tasks. To aid the children's learning motivation, we allowed them to have full flexibility concerning what kind of problem they were supposed to solve. We let the children themselves choose what to work on, in the spirit of "Learners generating examples", a methodology for children's creative use and exploration of mathematics by defining their own problems and corresponding solution process (Watson & Mason, 2005). For most of them programming was a new experience, so at the beginning of the project we demonstrated the basics of the Scratch programming environment and language. This included a short session about a Pacman-like game, and some programming of specific constructs like loops and assignment of values to variables. Even though in principle the learners could choose their own examples of problem-solving tasks, we observed that the introductory demonstration led some of them to construct examples that were quite closely related to the demonstrations. Thus, even if the problems and the solutions were supposed to be learner generated, we observed that the learners' starting points were quite close to the demonstrated examples. In this phase of the project, we did not look at this as problematic. Learners need to start at some point, and their starting point is necessarily close to what they have experienced through the initial instructional process.

The learners were not required to produce a specific result. Instead, they were encouraged to produce a working computer program that they could demonstrate to the other students, to their parents, to their teachers and to the researchers. Student products included anything from "an animated digital postcard" to a clone of well-known computer games like Pacman, and their own simple games invented during the project. Some of the students became motivated by the programming sessions, and developed their programs further at home. However, this was not a requirement that we set. During the programming sessions, most of the learners seemed to have a good time and seemed motivated by the project. Their teachers have also reported that some of the children were highly motivated by the project and continued programming after the project formally finished.

During the project period, the students worked on their computer programming projects in small groups of two to four members. We collected qualitative data through audio and video recordings; these provided contextual information, which gave us more possibilities to analyse the situations occurring, and contributed to a more precise understanding of the dialogues in the given context than could be gained through mere dialogue transcriptions. In this paper, however, we concentrate on transcriptions of a few dialogs, and sparingly provide some additional details about the situations, e.g. body language.

3.1. *Example 1: movement in the coordinate system*

In Scratch, the programmer works in a graphical window where the coordinate system implicitly plays an important role. The following student-teacher dialogue took place during the first day of the project. Some of the students made a simple computer game while slowly gaining an understanding of how the movement of an imaginary creature is controlled, and which operations are sufficient for complete freedom of movement. In this particular case, the students are programming a game that is quite similar to Pacman. Before this dialogue was recorded, the students had already drawn the main character of the game: a round creature (a face) with a mouth and eyes pointing towards the right on the screen. Even though the students do not realize it at first, the shape of the creature (non-symmetric placement of the mouth and eyes) meant that its movement in the plane was in fact not as easy to handle as they had anticipated.

Student: *We want to make Pacman. We'll teach him to jump.* [The student is fumbling with the programming process, and seems eager to get things to work.]

Teacher: *Have you thought about what you want it to look like when you are finished?*

Teacher: *What have you done so far in the script?*

Student: *Have made it move by pushing the keyboard.*

Student: *One thing we need help with is getting it to turn so that the cat* [the object/sprite that is being manipulated on the screen] *goes in the other direction without going backwards.*

Teacher: *Is it going to eat forwards? You have made it go forwards.*

Student: *Oh, yes.* [Sounding enthusiastic.]

Teacher: *You have made it change x by -5, right, and then you get it moving backwards. But before you make it change x by 5, you could maybe try to see if you can turn it around in some kind of way.*

Student: *Then it will just turn like this.* [The object starts rotating.]

Teacher: *If it is going to turn 15 degrees, maybe, ... Did you mean that if you turned it, it might be upside down?*

Student: *If you take this one here, for example, it would just start spinning, really…* [pointing to the screen while demonstrating what happens when rotation is applied to the character.]

Teacher: *Since every time you push it, it will move a little.*

In this example, the student wants to understand how to make a character (in this case a cat) move across the screen in a predetermined way, and how to make the character change direction based on commands given in the main loop of the program. We observed that the student already has some understanding of translation and rotation of objects in the plane, even though an informal language is used in the dialogue with the teacher ("we will make it jump", "if you take this one, for example"). With help from the teacher, the student broadens her view on which operations are desired: translation, rotation and flipping. The flipping operation seems new to the student. The character starts to rotate, while the student actually wants the character to move along a line, pointing 'face forwards'. Making the cat's face point in the right direction (and not backwards and upside down) simply cannot be achieved by mere rotation and translation, since the character has to be "flipped" 180 degrees to "point in the right direction". This means that, through this sequence, the student seems to extend her understanding of the needed operations (rotation, translation and flipping instead of just rotation and translation), or at least is broadening her view on the problem. Getting the character to move around in the coordinate system in a precise, predetermined way requires the learner to develop both a structural and processual understanding of the coordinate system, i.e. the "complete picture" of the movement needs to be established. The programmer can situate objects, such as sprites in a graphical window, and for example assign properties to them, or give instructions on how the objects are going to act under given circumstances. To navigate objects in the coordinate system, the programmer needs to develop some kind of abstraction and inner visualization to make the objects perform the desired actions. For the objects to follow the programmers' intended path, precise instructions must be provided through the programming language blocks. To some extent these instructions can be applied on a trial-and-error basis, however we do believe that the practical knowledge acquired in this manner contributes to the learning of mathematics. Computational and mathematical thinking is present when the learner needs to make an inner model for how the movement can be achieved. To do this the learner needs to have a fair understanding of the underlying coordinate system, or else develop this understanding during experimentation by following the instructions for succeeding with the programming tasks. The student gets immediate feedback on the outcome from watching the actual movement of the object on the screen when the program is run, and thus might get a better understanding of the coordinate system through exploration. We do believe that working with the coordinate system in this way helps the student getting a deeper understanding of the coordinate system.

The situation described above is quite closely related to the mathematical concept of a group, more specifically the Euclidean group (translation, rotation and line reflection). The various symmetries and non-symmetries of the character make a difference with regards to which operations are needed for a complete, closed description of the movement of the character. Mathematically, the characters in Scratch are equivalent to the turtles of the Logo programming language. This particular description of the equivalent Logo-problem has been studied in a more mathematical context by Zazkis and Leron (1991). In the Logo-programming language, the concept of flipping the character is required to describe the movement of the corresponding Logo turtle. In the Scratch language, probably the simplest way to implement a flip is by letting the character have different outfits corresponding to which direction the character points, and this is what we encouraged the students to do in this particular case. We did not focus on the mathematical concept of a group in the project, but it is clear that this could have been one way to work further to help the students gain an understanding of the underlying mathematical concepts. In the short time frame of the project, we did not focus on this. However, we do see this as a great opportunity for further investigation by students and teachers in the mathematics classroom. The concepts of movement and reflection in the Euclidean plane is a central part of the mathematics curriculum of Norway.

3.2. *Example 2: a control statement*

In this example, two students are trying to make characters on the screen send messages to the program when the character is touching something. It turns out that assigning a value to a variable is what the students end up doing.

Teacher: *We have to find out what happens when the ball hits the ghost…*

Student 1: *In other words when it touches the ghost.*

Teacher: *Then I do believe that you should use the "control command". Then you have to use an "if"* […]

Student 1: *Then we will need to have one "if" here and another "if" here.*

Student 2: *... it is being touched ...*

Teacher: *It has to send a message to the other one there ... It might have to change the outfit to nothing, or maybe we can look at „events".*

Student 1: *When I receive...*

Student 2: *Isn't it "send message" first?*

Teacher: *Yes, we have to make a new message. What name should we give it?*

Student 2: *It should be called "disappear".*
Teacher: *Then we will probably have to say what the message means…*
Student 2: *Maybe we have to look at its movement… Yes, go away or go back?*
Student 1: *When I receive "disappear"…*
Teacher: *Yeah, there … when I receive "disappear", we will have to look at what is going to happen.*
In this example, the two students are exploring two concepts that seems completely new to them: passing messages between characters and setting a variable state based on whether or not the characters are touching. This involves using the control-statement "if", which determines two different states of a Boolean variable. If the control statement is "true", the ball is hitting the ghost, and this should lead to the death of the ghost. The "if" control statement seems to be a natural construct for the students; they seem to understand its use. Although in the situation of message passing, the for the students new concept of a Boolean variable, they might be able to couple the object of interest with a variable that determines the state of the object. In that way the students can control the object by connecting the "on" or "off" construct to properties of the object. At the same time, the students use the "if" control statement. Control statements are of fundamental importance in computational thinking, and can be used in computer programming for things like automation of tasks. The students use the terminology of the "if" control statement in a natural manner, even though the language they use here is also a bit informal. We feel that the students show an elementary understanding of automation, which is closely connected to computational thinking. In terms of Niss' mathematical competences, representation competence, reasoning competence and communication competences are all in play with regards to the state and control statement in this situation Niss (2002). In this example too we see a great opportunity for students and teachers to bring computational thinking into the mathematics classroom. Things like working with states of variables, how objects can be controlled based on their properties, how we can assign values to variables etc., are all of fundamental importance in the mathematics classroom.

4. CONCLUSIONS

In both of the dialogue examples provided here we find examples of computational thinking in the context of open problem-solving tasks. In the first example, we especially see the modelling of character movement to be a computational problem that is solved by a succession of mathematical commands related to the rotation, translation and flipping of the character. The movement in itself is trivial to the students, but implementing the mathematical/computational way of handling the movement is in no way trivial to them. We observe that they are able to translate their intuitive understanding into mathematical commands, both through trial-and-error and through mathematical reasoning. However, it is not easy to say something about how this newly gained knowledge can be transferred to the mathematics classroom. We observe that the coordinate system is used in a different way from how students normally do, and thus the their concept of the coordinate system might have been extended during the project. In the second example, computational thinking is found to be more in the sense of modelling the process of message passing by the "if" control statement. With regard to the students' mathematical understanding, we observe that to some degree the student in the first example expands her understanding of admissible and possible movements that a character can undergo in order to move around the computer screen. These commands can be seen as mathematical operations that correspond to the mathematical concept of a group. For the time being, we feel that we have obtained indications pointing in the direction of students' learning and expanding mathematical concepts related to their computer programming, both in structural and operational thinking, and this has encouraged us to continue work on the project. We also observe that the students have developed mathematical competencies, especially reasoning competence, communication competence, modelling competence and problem solving competence. The project has motivated us to investigate further how programming in the classroom can help students learn mathematics.

5. REFERENCES

Bell, T. R., Rosamond, F., & Casey, N. (2012). Computer Science Unplugged and related projects in math and computer science popularization. In H. L. Bodlaender, R. Downey, F. V Fomin, & D. Marx (Eds.). *The multivariate algorithmic revolution and beyond: Essays dedicated to Michael R. Fellows on the occasion of his 60th Birthday*, pp. 398–456. doi:10.1007/978-3-642-30891-8_18

Bruner, J. (1966). *Towards a Theory of Instruction.* New York, NY: Norton.

Departementenes Sikkerhets- Og Serviceorganisasjon (2015). *Norges offentlige utredninger, 2015: 8. Fremtidens skole. Fornyelse av fag og kompetanser.* Oslo, NO. Retrieved from https://www.regjeringen.no/contentassets/da148fec8c4a4ab88daa8b677a700292/no/pdfs/nou201520150008000dddpdfs.pdf

Department for Education. (2013, September 11). *Statutory guidance National curriculum in England: computing programmes of study*. Manchester, UK: Department for Education. Retrieved from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

Euractiv. (2015, October 16). Infographic: Coding at school — How do EU countries compare? Retrieved from euractiv.com: https://www.euractiv.com/section/digital/infographic/infographic-coding-at-school-how-do-eu-countries-compare/

Kölling, M. (2010). The Greenfoot programming environment. *ACM Transactions on Computing Education (TOCE), 10*(4), 14. doi:10.1145/1868358.1868361

Kunnskapsdepartementet. (2006). *Kunnskapsløftet.* The Norwegian Directorate for Education and Training.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education (TOCE), 10*(4), 16. doi:10.1145/1868358.1868363

National Research Council. (2001). *Adding it up: Helping children learn mathematics*. J. Kilpatrick, J. Swafford, & B. Findell (Eds.). Mathematics Learning Study Committee, Center for Education, Division of Behavioral and Social Sciences and Education. Washington, DC: National Academy Press. Retrieved from https://www.nap.edu/read/9822/chapter/1

Niss, M., & Jensen, T. H. (2002). *Kompetencer og matematiklæring: Idéer og inspiration til udvikling af matematikundervisning i Danmark (Vol. 18)*. Copenhagen, DK: Undervisningsministeriet. Retrieved from http://www.gymnasieforskning.dk/wp-content/uploads/2013/10/Kompentecer-og-matematikl%C3%A6ring1.pdf

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., ... & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60-67. doi:10.1145/1592761.1592779

Sfard, A. (1991). On the dual nature of mathematical conceptions: Reflections on processes and objects as different sides of the same coin. *Educational Studies in Mathematics, 22*(1), 1-36. doi:10.1007/BF00302715

Silcoff, S. (2016, January 18). B.C. to add computer coding to school curriculum. *The Globe and Mail.* Retrieved from https://www.theglobeandmail.com/technology/bc-government-adds-computer-coding-to-school-curriculum/article28234097/

Tall, D. (1994, July). *A Versatile Theory of Visualisation and Symbolisation in Mathematics.* Paper presented at the Plenary Presentation at the Commission Internationale pour l'Étude et l'Amélioration de l'Ensignement des Mathématiques, Toulouse, FR.

Utdanningsdirektoratet. (2016, Oktober 20). Læreplanverket. Retrieved from http://www.udir.no/laring-og-trivsel/lareplanverket/

Utdanningsdirektoratet. (2016a). *Forsøkslæreplan i valgfag programmering*. Retrieved from http://www.udir.no/kl06/PRG1-01

Watson, A., & Mason, J. (2005). *Mathematics as a constructive activity: Learners generating examples. (Studies in mathematical thinking and learning).* Mahwah, NJ: Lawrence Erlbaum Associates Publishers.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35. doi:10.1145/1118178.1118215

Yaroslavski, D. (2014, July). How does Lightbot teach programming? Retrieved from http://lightbot.com/Lightbot_HowDoesLightbotTeachProgramming.pdf

Zazkis, R., & Leron, U. (1991). Capturing congruence with a turtle. *Educational Studies in Mathematics, 22*(3), 285-295. doi:10.1007/BF00368342