EXPERIENCING COMPUTATIONAL THINKING AND THE CONCEPT OF LOOPS IN AN OUTDOOR CS UNPLUGGED APPROACH

Sina Wetzel, Gregor Milicic and Matthias Ludwig
Goethe University Frankfurt, Germany

Abstract. *Computational Thinking (CT) is an important skill in the 21st century which should be fostered not only in computer science classes but also in K-12 STEM education in general. Taking a step back ignoring computers and other smart devices, an unplugged approach to computational problems can help students develop competencies related to CT without the need for advanced prior knowledge such as programming skills. Taking unplugged activities outside the classroom can ignite higher motivation within students to engage in these activities and elements of embodiment can lead to a deeper understanding. This paper presents a possible approach to experience CT and the concept of loops outside where students simulate a programmable Turtle on an outdoors area such as the school yard.*

Key words: Computational Thinking, Algorithmic Thinking, Computer Science Unplugged, Outdoor Learning, Turtle Graphics, Loops

THEORETICAL BACKGROUND

Teaching children how to program is not a new endeavor with first efforts to foster school students' programming skills already emerging in the 1970s and 1980s (see e.g. Pappert, 1980). However, in recent years, a different approach can be observed as well in which the focus has shifted from teaching the mere skill of programming to its underlying fundamental strategies and thinking skills. These skills were subsumed by Wing (2006) under the term of Computational Thinking (CT). Although Wing was not the first one to use this term, her paper can be seen as the starting point of the current discussion of CT (Hoppe & Werneburg, 2019, p. 14). Since then, many researchers have tried to capture the essence of CT with most definitions agreeing that it is a specific way of thinking about and solving problems (Bocconi et al., 2016). Core skills of CT include – but are not necessarily limited to – abstraction, algorithmic thinking, automation, decomposition, debugging and generalization (ibid.).

To thrive in the technology-driven world of today, becoming skilled at CT is of uttermost importance (ISTE, 2016). However, there is no consistent embedding of CT in most countries' school curricula with a great variation among the countries (Bocconi et al., 2016). Even though CT is considered a key competency in the digital age, engaging with CT does not necessarily mean working with a computer or other digital devices. The so-called Computer Science (CS) Unplugged approach, which was developed at the University of Canterbury in New-Zealand, aims at engaging people with key ideas of computer science without actually using computers (see Bell, Alexander, Freeman and Grimley, 2009). In mainly kinesthetic activities, participants are confronted with problems which they should solve. While some activities specifically aim at educating the participants in computer science, some also foster the more general skill of CT (ibid.). Since the threshold to participate in CS Unplugged activities is usually quite low as they are independent from any kind of programming knowledge and there is no necessity for any kind of hardware or software, they seem promising to foster CT especially in younger children.

Wetzel, S., Milicic, G. & Ludwig, M. (2020). Experiencing Computational Thinking and the Concept of Loops in an Outdoor CS Unplugged Approach. In M. Ludwig, S. Jablonski, A. Caldeira, & A. Moura (Eds.), *Research on Outdoor STEM Education in the digiTal Age. Proceedings of the ROSETA Online Conference in June 2020* (pp. 187-194). Münster: WTM. https://doi.org/10.37626/GA9783959871440.0.23

With CS Unplugged not relying on the use of technology, appropriate activities can also be transported outside the classroom. As such, they could present an opportunity to experience elements of CT outdoors. Participating in such activities outside the classroom, could increase students' motivation and lead to a higher level of engagement. Depending on the activity, working outside also offers positive side-effects like having more space and not being limited by constraints created through the properties of a classroom. In the following, we present a lesson design and describe an experimental test run of it. Apart from testing the quality and meaningfulness of the lesson design, we exemplarily take this lesson as incentive to explore if this and similar activities can engage younger children with CT outside the classroom.

DESIGN OF AN OUTDOOR CT LESSON

Bell et al. (2009) argue that a single lesson where students experience a CS Unplugged approach can already have a significant impact on the students' concepts of computer science. This clearly distinguishes CS Unplugged approaches from other, more complex approaches that cannot exist as stand-alone lessons but have to be embedded in a profound curriculum. We aim at transferring this positive effect to an outdoor lesson targeting young children in grade 5 or 6 where they can intuitively experience some of the core skills of CT. The lesson we present revolves around the idea of a Turtle (see e.g. Pappert, 1980) which is a programmable object with a position and a direction sitting on a canvas on which it can draw. Pappert (1980) argues that by identifying with the moving Turtle and imagining what they would do if they were the Turtle, children can combine the knowledge about their own body and its movement with a problem they aim to solve (p. 56). As such, a Turtle can be an intuitive approach to programming for younger children. However, solely imagining the Turtle's movement might not always suffice as Benton et al. (2018) argue that students often confuse directions in which the Turtle should move. They thus propose to engage in related unplugged activities to help diminish these errors. In our approach, instead of only controlling the Turtle, students also act as the Turtle which strongly relates to the idea of embodiment as proposed by Tall (2003). Tall defines embodiment as "thought built fundamentally on sensory perception" (p. 4) and argues that an embodied approach to a topic can help students build meaning.

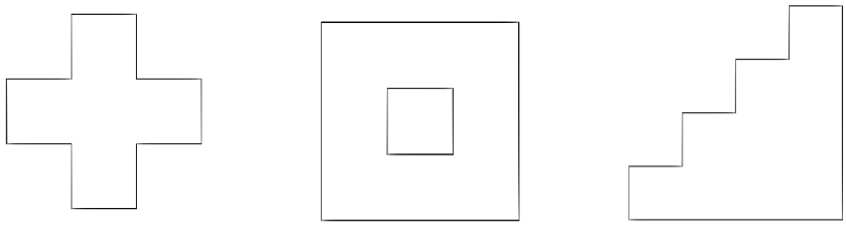

Figure 1: Target figures to draw.

In the lesson, which is designed for ninety minutes, students develop algorithms to draw a designated object which is made up of squares since squares are easy to draw and can also be found outside, for example on paved yards. Students work together in groups of three and each team member gets assigned his or her own target figure (see Figure 1). The students shall use a predefined set of instructions for their algorithms (see Figure 2). Instead of only providing the students with textual commands, we decided to illustrate

them in the form of blocks, similar to Scratch (https://scratch.mit.edu/) or other block languages. Available blocks include forwards movement, ninety degree turns to the right and left, instructions if the chalk should be used or not, a command to go to the starting point and a bigger block for a loop, as it is represented in scratch, with the possibility to insert the commands that get repeated inside. The provided code blocks can be printed and cut out so they can be used as a kind of puzzle, however, working with blocks made of paper can prove quite fiddly and it takes a long time to cut out enough blocks. Another possibility is to just provide these blocks as a guideline to the students and let them draw the blocks on paper. As such, the students can flexibly adjust the size of the loop block, which is also possible in Scratch (Maloney et al., 2010).

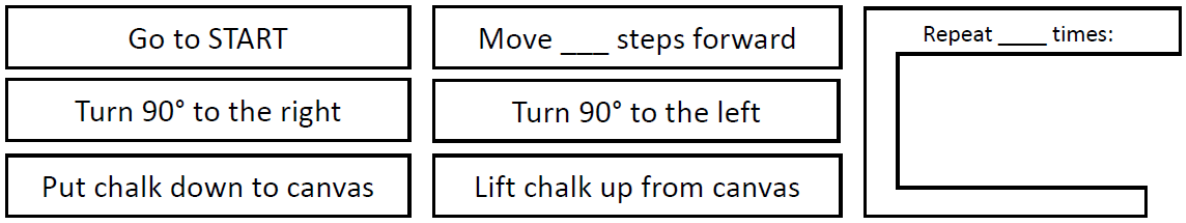

Figure 2: Set of commands.

One key element of the lesson is the division of the students into groups of three. Each student gets assigned one target figure which no one else is allowed to see. All three students individually conceptualize an algorithm with which the image can be drawn. Thus, all of them get the opportunity to equally engage in algorithmic thinking. During the testing phase of their program outside, each student of a group has a different role. Students take turns so that every student adopts each role once. The student who wrote the code is the developer who, in a real programming situation, would have no influence on what the program does after hitting the run button. He or she can only wait and see if something goes wrong and, if necessary, ponder possible errors in the code. Thus, in our lesson, the student who wrote the code is a quiet and passive observer while the other two students actively work with the code. One of the remaining two students reads out the code line by line, not knowing what the result is supposed to look like and thus not able to correct the code while reading it out. This student therefore serves as the machine or compiler translating the code into something that can be executed. The third student is the Turtle and executes the commands exactly like they are read out. This strict separation of responsibilities helps the students to recognize the difference between the developer, the machine or compiler and the executing unit, in our case the turtle, with the latter two both knowing nothing of the developer's intentions.

Learning goals of the lesson include a first introduction to sequential coding and an intuitive understanding of the advantages of loops. Concerning the core skills of CT following the notion of Bocconi et al. (2016), firstly, students engage in algorithmic thinking when conceptualizing and writing their code. During the lesson, it should be highlighted that debugging is a normal part of every development process and that students should not fear errors but instead analyze them closely, ideally learn from them and then change their code accordingly. They thus practice debugging, which, according to Bocconi et al. (2016) is "… the systematic application of analysis and evaluation". Since all three target figures contain repetitive elements and parts of squares, students can simplify their code considerably if they recognize these patterns and use loops which is a part of generalization (Bocconi et al., 2016). However, recognizing the squares and solving the

problem for a single square first can also be considered a part of decomposition: instead of solving the whole problem at once, a smaller and easier problem namely the one of a single square gets addressed first. The lesson could hence foster the four core CT skills algorithmic thinking, debugging, generalization and decomposition.

METHODS

According to Bell et al. (2009), a CS Unplugged approach should be developed in three steps: designing, testing and adjusting. It is thus essential to do a test run of a proposed lesson. We tested the lesson with students of grade 5 in January 2020. As this trial run was meant to pilot the lesson so that the concept could, if necessary, be modified and improved before conducting it with a complete class of 30 students, we decided to invite six students to participate in the test run so that we would have two groups of three students each.

Apart from observing the students during such a test run, it can provide additional insights if students fill in pre- and post-questionnaires. The purpose of the questionnaires is threefold: we want to raise data concerning the students' pre-knowledge and knowledge gain through the lesson, ask them for their opinion on outdoor learning before and after the lessen as well as generally find out how they like the lesson. As one goal of the lesson is to introduce the concept of loops, we ask students before and after the lesson whether they know what a loop is and, if they do, whether they can describe its purpose and advantages. All other data, i.e. their attitudes towards computer science and outdoor learning as well as whether they like the lesson, is raised using a four-point Likert scale. We decided to use a Likert scale to better describe whether there are differences before and after the lesson.

EXPERIENCES DURING THE TEST RUN

The lesson started with a short input phase inside in which we explained the goal of the lesson and the commands, with special focus on how to use the loop block, as well as the structure of and responsibilities in the groups. Exemplarily, one of us read out some lines of code while someone else embodied the Turtle to show the students how the role division works when executing someone's code. We also gave them two examples how a loop command could be read out by either repeating the code inside the loop as many times as needed and leaving out the loop completely or by making clear which commands needed to be repeated how many times.

Afterwards, the students each received a target figure (one of those in Figure 1) which they were not allowed to show to anyone else. Before they started to develop their programs, we informed them that there would be a debugging phase after the first test execution of their programs. Each student received a clipboard with a sheet in foil on which they should write their code with a non-permanent marker so that they could easily change their code. Students designed their programs inside before we went outside for the first test run. Even though everyone was occupied at first, some students were significantly faster than others. For example, the student who wrote the code depicted in Figure 3 on the left finished first. By recognizing that a pattern was underlying his target figure and consequently using a loop, his code was considerably shorter than that of another student in Figure 3 on the right who took longest and did not use any loops. However, the code of the student on the left was erroneous and did not produce the target figure since he used only left turns but

never a right turn (the translated version of his code is displayed in the bottom right corner of the picture). When asked to recheck his code while he was waiting for the others, the student assured us that his solution was correct and that the task was far too easy. Hence, he could not recognize his mistake in the conception phase of the lesson.

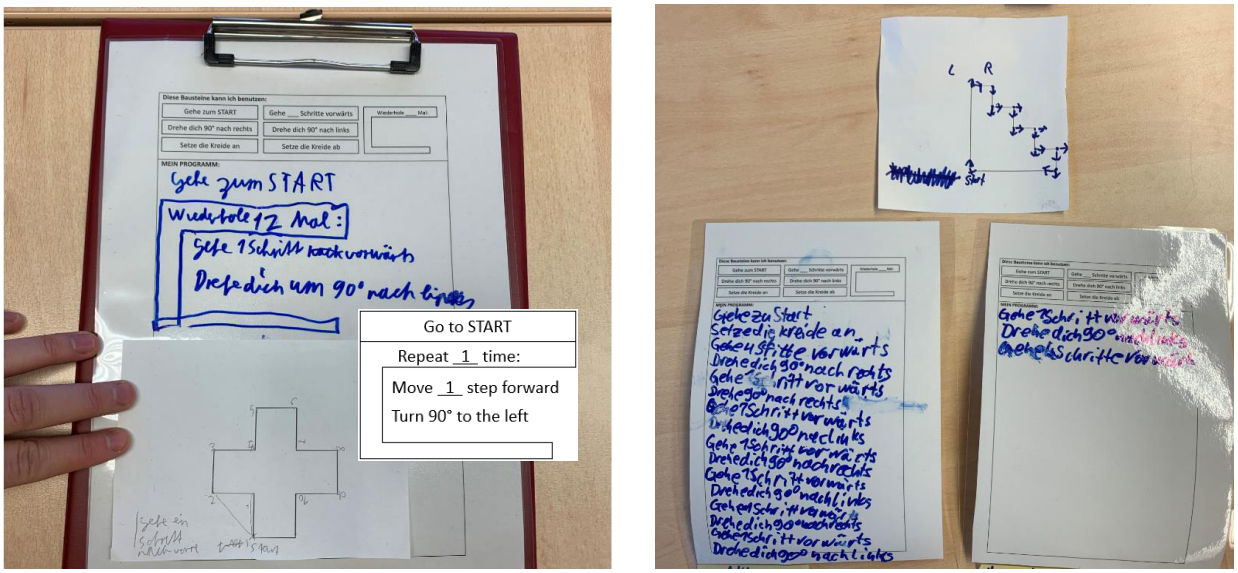

Figure 3: Two student solutions, one with a loop and one without one.

For the drawings, we used the yard in front of the university which is paved with square tiles (see Figure 4) to facilitate the drawing process. Even though the weather conditions were not perfect, the students were eager at work. The code of one member of each team already worked correctly in the first test run while the others' codes contained errors. The errors were mostly related to the direction in which the Turtle should turn. When there was an error and the code did not produce the intended result, the student in the role of the supposedly passive developer often showed signs of discontent and attempted to interact with his team mates to correct the errors. We therefore needed to remind the students to simply observe and think about possible corrections which could be implemented in the following phase but not to intervene during the execution of the code. Some of the students also had difficulties in the role of the compiler when reading the code out loud, especially when dealing with a loop. Keeping count of the number of iterations and giving the corresponding commands in the right order proved challenging for some of them. After a while, though, the students got more proficient and gave the commands in the right order.

In a subsequent debugging phase, students should adapt their code collaborating with their team mates. We observed that the students actively used their bodies during this phase to decide whether the Turtle should turn left or right. Already on the second attempt, all students' programs worked. Students were also asked to simplify their code if possible, which prompted some students to use a loop who had not used a loop before. In the end, five of the six students had used loops. At the end of the lesson, the concept of loops and its usefulness in programming was discussed briefly.

Figure 4: Students in action.

RESULTS AND DISCUSSION

When we asked the students what a loop was in the pre-questionnaire, most drew a ribbon since both words translate to “Schleife” in German. Interestingly, both students who said they had no prior programming experience could give a basic definition of loops as “something that gets repeated”. In the post-questionnaire, all students were able to give a basic definition of loops and gave answers such as “needs less space” or “one does not have to write everything numerous times” when asked for its advantages. They therefore constructed a first understanding of loops for themselves. Even though they are limited, the definitions and advantages given by the students seem age-appropriate. However, when conducting such a unit, one must be careful not to create misconceptions or one-sighted views of a concept, in our case loops. When Benton et al. (2018) asked students of grade 6, who had only worked with code blocks so far, to describe how a given algorithm could be extended to a more complex problem, some could not answer this question in words as was expected but instead chose to draw code blocks as an answer. After our unit, we made a similar observation as one student focused mainly on the block aspect and wrote that a loop was “a block that repeats things”. It is thus important to at least mention that a loop does not have to be a block. Keeping this aspect in mind, however, the lesson seems appropriate to let the students construct their first concept of loops. We also asked students about outdoor learning. Most indicated that they only rarely, if at all, went outside for a lesson in their school. When asked if they liked having lessons outdoors most agreed at least partially before our lesson. One student who answered “don’t know” and one who partially agreed both completely agreed in the post-questionnaire. Apart from one student, all completely agreed that they had had fun during our lesson. Since all students had produced working code at the end of the lesson and also indicated to have had fun, the motivational aspect seems to be high as well. However, this can probably at least partly be attributed to the general novelty aspect of the lesson.

In accordance with the design, test, adjust cycle as described by Bell et al. (2009), the lesson should be analyzed and, where necessary, adapted. In a study executed by Benton et al. (2018), students of grade 6 where asked to compare algorithms and many struggled to recognize that shorter and more readable algorithms should be preferred over long and

complicated ones. Due to the limited space on the task sheet in our unit, most students realized that producing numerous lines of code is not desirable if it can be avoided. Providing the students with a sheet to write their code on can thus create the inherent need for loops and seems favorable. However, we would not give out sheets in foil again, since the writing got smeared after some time and thus almost unreadable. It is also important to consider ways to deal with students who are very fast in the initial design phase. One possibility could be not to assign the groups in advance but to always let the fastest three, the second fastest three and so on form a group and allow them to immediately start their first test run.

The division of responsibilities within each team into different roles (developer, compiler and Turtle) worked mostly well. Some problems arose when students had to read out loop blocks. Despite a demonstration at the beginning of the lesson how they could do this, most of the students had some difficulties during this translation process. To avoid this source of confusion and reduce the probability for errors, it might be helpful to discuss loops further at the beginning of the lesson and give more examples how they can be read out by the student who simulates the compiler. The division of roles also requires a high level of discipline from the student who acts as the developer to not interfere during the execution of the code. A possible adaption could be the addition of a forth student to each group acting as a referee and supervising the whole process. Whenever needed, this student could remind the developer not to interfere. This adaption would also offer some benefits when conducting this lesson in a class with more students as it would either reduce the total number of groups or would provide some flexibility allowing groups of both three and four students.

CONCLUSION AND OUTLOOK

In alignment with the presumption of Benton et al. (2018), embodying the Turtle and executing its actions themselves helped students to improve their understanding of direction and, as a consequence, their algorithms. At first, most made mistakes related to direction, however, one could clearly observe how they made use of their body when correcting their code. This is a clear gain from doing this unit outdoors where there was enough space for students to move and turn and thus experience the movements on a deeper level. The same unit inside using a pencil and piece of squared paper might not have had the same effect. The proposed lesson seems appropriate for young children to make first experiences with CT. An outdoor unplugged lesson like the one proposed can thus serve as a playful and motivating incentive for students to engage with CT already at a younger age without the need for computers or other expensive equipment, opposed to other approaches where students are engaged in coding outside using technology (Hitron et al, 2017; Offer et al., 2018). As we only conducted the lesson with six students, however, it is necessary to do a second test run in the future with a whole class of grade five or six students and to review whether results especially regarding the knowledge gain can be reproduced.

In future work, it could also be interesting to further focus on the outdoor aspect and how it can help foster the core CT skills. In this regard, it might be interesting to compare the effects on students' CT skills of an outdoor lesson like the one we proposed and a similar lesson conducted inside. This would however require a reliable tool to measure students'

skills in CT. A possible variation of our proposed lesson to stronger emphasize the aspect of outdoor learning could also be to ask students to find interesting shapes outside such as paving pattern or special wall structures and then to design a program with which a Turtle could reproduce these shapes.

References

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer science unplugged: School students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, 13(1), 20-29.

Benton, L., Kalas, I., Saunders, P., Hoyles, C. & Noss, R. (2018). Beyond jam sandwiches and cups of tea: An exploration of primary pupils' algorithm-evaluation strategies. *Wiley Journal of Computer Assisted Learning*, 34, 590-601.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A. & Engelhardt, K. (2016). *Developing Computational Thinking in Compulsory Education*. Seville: JRC Science Hub.

Hitron, T., Apelblat, I., Wald, I., Moriano, E., Grishko, A., David, I., Bar, A. & Zuckerman, O. (2017). Scratch Nodes: Coding Outdoor Play Experiences to enhance Social-Physical Interaction. In *Proceedings of the 2017 Conference on Interaction Design and Children* (pp. 601-607).

Hoppe, H.U. & Werneburg, S. (2019). Computational Thinking—More Than a Variant of Scientific Inquiry! In S.-C. Kong & H. Abelson (Eds.), *Computational Thinking Education* (pp. 13-30). Singapore: Springer.

International Society for Technology in Education [ISTE] (2016). *ISTE Standards for Students*. Retrieved from https://www.iste.org/standards/for-students [02.03.2020].

Maloney, J., Resnick, M., Rusk, N., Silverman, B. & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, 10(4), 1-15.

Ofer, N., Erel, H., David, I., Hitron, T., & Zuckerman, O. (2018). A little bit of coding goes a long way: effects of coding on outdoor play. In *Proceedings of the 17th ACM Conference on Interaction Design and Children* (pp. 599-604).

Pappert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas*. New York: Basic Books Inc.

Tall, D. (2003). Using technology to support an embodied approach to learning concepts in mathematics. *Historia e tecnologia no Ensino da Matemática*, 1, 1-28.

Wing, J. (2006). Computational Thinking. *Communications of the ACM*, 49(3), 33-35.