Programming Unplugged: An Evaluation of Game-Based Methods for Teaching Computational Thinking in Primary School

Luzia Leifheit[1,2,3], Julian Jabs[2], Manuel Ninaus[3,1], Korbinian Moeller[3,4,1], & Klaus Ostermann[1,2]

[1] LEAD Graduate School and Research Network, University of Tübingen, Tübingen, Germany

[2] Department of Computer Science, University of Tübingen, Tübingen, Germany

[3] Leibniz Institut für Wissensmedien, Tübingen, Germany

[4] Department of Psychology, University of Tübingen, Tübingen, Germany

luzia.leifheit@uni-tuebingen.de

julian.jabs@uni-tuebingen.de

m.ninaus@iwm-tuebingen.de

k.moeller@iwm-tuebingen.de

klaus.ostermann@uni-tuebingen.de

Abstract: Game-based approaches can be a motivating and engaging way of teaching and learning, in particular for younger students. To evaluate the suitability of game-based approaches for teaching programming in primary school, we conducted a field study on Computational Thinking (CT). CT can be characterized as the ability to understand, formulate, and systematically solve complex problems, which typically requires abstraction, generalization, parameterization, algorithmization, and partitioning as processes of CT, which are also vital to programming. The employed CT course focused on fostering students' conceptual understanding of computational thinking independent of specific technological applications and was based on course material from code.org. Lessons primarily addressed algorithms as a core CT concept and used game-based learning material to increase students' understanding of algorithmic CT concepts such as sequences, loops, branches, and events. These concepts were first introduced through unplugged game-based activities using tangible everyday objects (e.g. pencils, playing cards, etc.) instead of abstract code. In more advanced lessons, students' conceptual understanding was applied and deepened through plugged-in programming exercises. The 18 sessions (45 minutes each) of the course were taught to 33 3rd and 4th grade primary school students. At the end of unplugged lessons, students' understanding of newly introduced CT concepts was assessed by short tests. Students' interest in and motivation for programming education was measured with pre- and post-course self-assessment questionnaires. Results indicated benefits of the unplugged game-based approach for teaching CT concepts. In particular, we observed (a) for all but one of the CT concepts, students reached on average 82% of the learning objectives or more, and (b) students rated their learning experience in the course positively and reported high levels of interest in learning more about computation-related topics. In addition, qualitative analyses indicated part of the curriculum was very complex for the target group (e.g. nested conditionals). This finding is of particular interest for the development and evaluation of future programming courses for primary school students.

Keywords: computer science education, programming education, computational thinking, game-based learning

1. Introduction

In recent years, computer programming or coding has repeatedly been argued to be a so-called 21st century skill in our increasingly technological world, which is more and more dominated by algorithms (e.g., Willson, 2017). As a consequence, different authors and institutions suggested computer programming should be part of school curricula (e.g., Papert, 2005; Balanskat & Engelhardt, 2015). However, just teaching children one or the other programming language does not seem desirable, as programming languages develop fast. Instead, it seems more promising to introduce children to what has been termed computational thinking (henceforth CT). CT is characterized as the processes involved in understanding, formulating, and solving complex problems in a way that admits a computational solution, using abstraction, generalization, partitioning, and algorithmization (Wing, 2014). What teaching children – or anyone – to think computationally actually means is not to turn them into computer programmers, but to help them develop their ability to understand complex problems and to cultivate strategies for solving them systematically.

1.1 Computational thinking in primary education: Why and how?

Across Europe, the focus on logical thinking, problem solving, and programming skills in education is increasing as more and more countries are integrating or planning to integrate programming-related skills into school curricula for as early as primary education (e.g., Balanskat & Engelhardt, 2015). This development is in line with the notion that CT is a fundamental skill for the 21st century (Yadav et al., 2014).

Already Papert (1972) proposed letting children write interactive programs can be used for fostering their ability to articulate and simultaneously learn about their own thought processes, considering creating such programs requires children to contemplate and anticipate all possible misunderstandings and mistakes the program's users might encounter or make. On this basis, he argued programming can be used to support the fundamental ability of understanding one's own thought processes, and thus suggested integrating simple elements of computer science, such as programming, into primary education (Papert, 2005).

Primary school seems an appropriate time for introducing students to programming because the concrete operational stage (according to Piaget, 1972) is typically reached between the age of 7 to 11 years. This stage, according to Neo-Piagetian models of cognitive development with regard to learning programming, is "the first stage where students show a purposeful approach to writing code" (Lister, 2016) and "can write small programs from well-defined specifications" (Teague et al., 2012). For early computer science education, Prottsman (2014) suggested a CT approach making use of unplugged exercises (using concrete materials such as pencils, cards, etc.) to assist the progress of learning abstract computational concepts students will encounter when they begin to learn programming (Kumar, 2014).

Accordingly, we used unplugged games in our study to facilitate children's learning. In particular, we incorporated various forms of embodiment, which was argued to facilitate implicit learning (Barsalou, 2008) and was shown to positively influence comprehension and memory through increasing the organization of conceptualization (Noice & Noice, 2001). Moreover, the use of game elements in a working memory task was found to have the potential for improving learners' performance (Ninaus et al., 2015). In a review of empirical studies on gamification, Hamari, Koivisto, and Sarsa (2014) concluded gamification increases motivation and engagement in the learning task. Therefore, we expected our game-based approach on learning to program to be motivating and engaging for students and to yield significant learning effects. In the following, we will briefly describe the course curriculum employed before reporting the results of the study.

1.2 Course curriculum

The game-based CT teaching approach evaluated in this article was part of a CT course that was taught at two primary schools in Germany as part of an exploratory pilot project in 2016 and 2017. The course followed a concept-based curriculum modeled closely on the existing Course 2 (Code.org, Course 2) by Code.org (Code.org, About US), which is available under Creative Commons Attribution 4.0 International License and has been successfully evaluated in programs for primary school students across the United States (Code.org, Evaluation Summary Report). Code.org's Course 2 is aimed at children who are already able to read and write, but who have not yet taken any previous courses on computer science, programming, or CT. It follows a CT approach by placing its focus on conceptual understanding rather than specific implementations and makes use of many 'unplugged' exercises and teaching units. Exercises being 'unplugged' means they abstain from using technology and instead are carried out using pencils, paper, scissors, glue, playing cards, or even sports-like games.

The course started by introducing students to a key computer science concept – the *algorithm*, a sequence of commands that need to be followed to solve a problem. The following teaching units explored ways in which algorithms can be structured, improved, and the flow of algorithmic instructions can be controlled. In addition to algorithms, the fundamental concepts around which the curriculum is structured include *sequences*, *loops*, *conditional branching*, and *events*. Furthermore, students are intended to learn how to debug algorithms. To complement the 16 lessons we selected from Code.org's Course 2 and to relate them to the existing primary school curriculum, we added 2 unplugged lessons prompting children to transfer their acquired understanding of CT skills and concepts to solve mathematical problems.

Throughout the course, the focus was on helping students develop an understanding of these concepts. The course applied this conceptual approach to help students build a long-lasting and sustainable understanding of CT. Specific technological applications including programming languages are developing fast, but it is argued that the underlying computational concepts are what is fundamental to computer science (Wing, 2006) and that they are applicable more universally (Wing, 2010). Any technology used in the course was only utilized as means to increasing conceptual understanding rather than being an end in itself. For this reason, some of the teaching

units contain programming exercises closely interlinked with the respective unit's conceptual content. Overall, 10 lessons included unplugged activities, while the remaining 8 lessons included programming exercises using tablet devices and desktop computers. Over the course of the curriculum, each CT concept was first introduced in one or more unplugged lessons and consequently applied in a 'plugged-in' lesson.

2. Methods

2.1 Course framework

2.1.1 Lesson structure and course set-up

All of the lessons evaluated in this article followed the same structure. At the beginning of the respective lesson, students were asked to recall the previous lesson's activities and were encouraged to ask questions they have come up with in the aftermath of the previous lesson. Then, the terms and vocabulary for any new CT concepts introduced were presented and discussed with students. The central part of the lesson always consisted of the respective game activity. After the game, the teacher prompted them to discuss what they learned from this activity and how this may be implemented when working with an algorithm. At the end of the lesson, students were handed the assessment sheet on the CT concept introduced in the lesson and were given ten minutes to solve it. Each end-of-lesson assessment tested assigned learning objectives (Code.org, Course 2) and students' results were assessed based on which percentage of the respective learning objectives they reached.

The course was taught by four computer science undergraduate and graduate students who received formal training in teaching the CT course in the format of a weekly two-hour (120 minutes) course over a timespan of three months. Students taught the course in teams of two, which were assigned to the two schools respectively.

The course at school A (n = 22 students; 10 female; mean age = 8.76 years; SD = 0.75) was taught in lessons of 90 minutes each on a weekly basis over the course of 9 weeks. In contrast, the course at school B (n = 11 students; 5 female; mean age = 9.00 years; SD = 0.67) was taught in lessons of 45 minutes each on a weekly basis over the course of 18 weeks. Written informed consent of parents or legal guardians was required for the students to sign up for the courses, which were offered during the schools' afternoon programme. Both schools combined, the total number of participants was 33 3rd and 4th grade students between eight and ten years old.

2.1.3 Student background

Before the start of the course, children filled out self-assessment questionnaires. These questionnaires asked students to provide background information on, for instance, their previous programming experience or whether their parents were working in IT-oriented jobs.

Out of 33 students participating in the course, 28 filled out the questionnaires – the remaining 5 students had not been present when the questionnaires were administered. Evaluation of questionnaire data yielded the following results: 14 students (6 female) stated at least one of their parents' jobs had something to do with information technology. 6 students (2 female) stated they had some previous experience in programming. 2 of these students, and 3 others who had no programming experience (no female), stated they knew what the term 'computer science' means. 5 out of the 9 students who had previous programming experience or knew what the term 'computer science' means had at least one parent with an IT-related job.

2.2 Pre- and post-course questionnaires

At the start and at the end of the course, students filled out questionnaires to rate their perception of the course in general as well as their interest in computation-related topics and their self-perceived learning progress.

In both the pre- and post-course questionnaire, students were asked to indicate their answer to the question "Could you imagine working in an IT job?" on a 5-point Likert scale ranging from "not at all" (1) to "absolutely" (5) (with only the endpoints labelled accordingly). In the post-course questionnaire only, they were also asked to indicate on the same Likert scale (i) whether they liked the course in general, (ii) whether they learned something new in the course, and (iii) whether they were interested in learning more about computer science.

2.3 The unplugged games

The following three games each made up the central part of one of the lessons evaluated in this article.

2.3.1 "Relay Programming" – a debugging game

Debugging is the action of systematically searching for and consequently resolving problems or errors in an algorithm or program. The lesson's educational objective was for students to be able to find and fix problems in existing programs. This ability was introduced and practised using an unplugged game.

For this game, students were divided into teams of three to five players to compete in a race against the other teams and against time. For each team, the goal was to create a program – command by command – and to debug this program in case any errors occurred. The game's learning objective was for the students to learn to detect errors quickly and debug their program systematically. The time pressure brought about by the competitiveness and fast pace of the game was supposed to provoke students to make mistakes when writing down commands, thus providing the opportunity to learn debugging.

To play the game, each team had to form a queue on one side of the room, just like in a relay race. For each of the teams, the same image is placed on the other side of the room, as well as a sheet of paper on which the teams should write down their program. All images were grids of four times four squares, some of which are black and some are white. One corner of the grid was marked as the starting point for the program. See Figure 1 for an example picture and the commands that can be used to write the program.

When the game starts, the first student in line for each team had to run to the other side of the room, look at the picture, and write down the first command for the program that would reproduce that picture. Then the student had to run back to their team, clap the hand of the next student in line, and go to the end of the queue. The next student in line, who just got a handclap from the previous player, then had to run to the other side of the room, look at the picture, and either debug the team's current program by crossing out an incorrect command or add a new command. Then the student had to run back to their team, clap the next teammate's hand, and the cycle started over again. Importantly, on each turn, each player can only perform one action – either debug a previous command or add a new command. This cycle was repeated until one of the teams completed their program and thereby won the game. The game was played several times in a row with increasing difficulty, determined by the complexity of the respective picture.

Due to limited space in the course classroom at school A, it was impossible to ensure large enough spatial separation between the currently active player and the rest of the team during the "Relay Programming" game phase of the lesson. As a consequence, the active player sometimes received help from their teammates.

At the end-of-lesson assessment, students had to fill out a short test comprising four exercises. Each exercise consisted of one picture just like in the "Relay Programming" game and a sequence of commands (see Figure 1). When the sequence is executed, it was supposed to draw the respective picture. However, there is one faulty command in each sequence. It was the students' task to find each mistake, circle it, and write a new, correct sequence of commands for drawing the picture. This assessment tested how well students were able to detect a bug within a sequential algorithm and replace it with an error-free sequence of commands, which they had learned to do through playing the "Relay Programming" game.

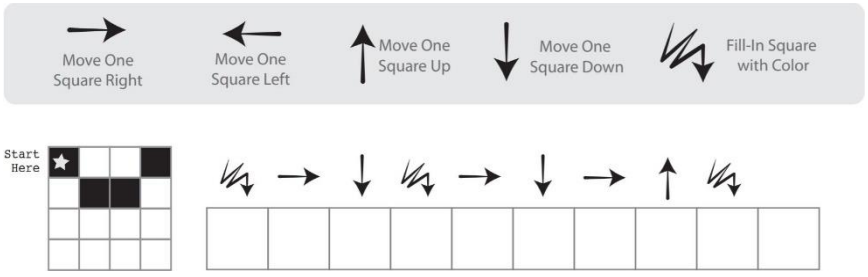

Figure 1: One of the four tasks referring to the debugging lesson's assessment sheet (Code.org, Assessment 9)

2.3.2 "Conditionals with Cards" – a conditional branching game

Conditionals are expressions used for control flow branching within an algorithm, depending on whether the condition is met, meaning whether the expression is true or false. Students had to learn about how conditionals work by playing an unplugged card game with two levels of difficulty, the rules of which were based on awarding points depending on whether a specific condition was met. To play the game, students were split into two teams. Both teams had to keep track of the current score for their own team as well as for their opponent.

For the first difficulty level, teams took turns in each drawing one card from a standard poker deck. The game's rules applied simple conditionals as shown in Figure 2.

```
If (CARD is RED)
    Award YOUR team 1 point
Else
    Award OTHER team 1 point
```

```
If (CARD is RED)
    Award YOUR team 1 point
Else
    If (CARD is higher than 9)
        Award OTHER team 1 point
    Else
        Award YOUR team the same
        number of points on the card
```

Figure 2: Example of simple (Panel A) and nested (Panel B) conditionals in "Conditionals with Cards"

When it was clear all students understood the rules, they moved on to the second level and were given a new set of rules, this time containing nested conditionals as shown in Figure 2.

For the respective end-of-lesson assessment, students filled out a short test instructing them to imagine they were watching a game of cards being played by two teams. Then participants were asked to write down the teams' scores after each turn. The game they "watched" was very similar to the "Conditionals with Cards" game they had just played and its rules also made use of nested conditionals (see Figure 4). This assessment was supposed to test how well students could trace conditional branching in an algorithm and award points to two teams accordingly, which they had learned to do by playing the "Conditionals with Cards" game.

```
If (CARD is lower than 5)
    If (CARD is BLACK)
        Award YOUR team the same
        number of points on the card
    Else
        Award OTHER team 1 point
Else
    If (CARD is HEARTS)
        Award YOUR team 1 point
```

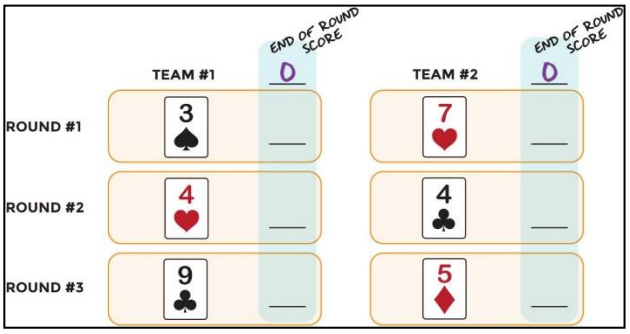

Figure 4: Exemplary assessment task on the conditional branching lesson's assessment sheet (Code.org, Assessment 12)

In case students follow the algorithm and evaluate all conditionals correctly, they keep track of the score as follows:

1st round, 1st turn: Team 1 draws the 3 of spades (lower than 5, black), awards 3 points to team 1.

1st round, 2nd turn: Team 2 draws the 7 of hearts (not lower than 5, hearts), awards 1 point to team 2.

2nd round, 1st turn: Team 1 draws the 4 of hearts (lower than 5, not black), awards 1 point to team 2.

2nd round, 2nd turn: Team 2 draws the 4 of clubs (lower than 4, black), awards 4 points to team 2.

3rd round, 1st turn: Team 1 draws the 9 of clubs (not lower than 5, not hearts), awards 0 points to either team.

3rd round, 2nd turn: Team 2 draws the 5 of diamonds (not lower than 5, not hearts), awards 0 points to either team.

Thus, the final score after the last turn is 3 points for team 1 and 6 points for team 2, which is the winning team.

2.3.3 "The Big Event" – an events game

Events are actions an algorithm is designed to recognize and react to. Typical events handled by computer programs include user actions such as mouse clicks, which can be used to make a program interactive.

A sports game was used to help students develop a basic understanding of how events work and what they can be used for. For this game, students were introduced to five geometric symbols, each of which was linked to one

specific action such as standing on one leg. Then, students were instructed to randomly run across the schoolyard until the teacher would hold up a cardboard sign showing one of the symbols. Upon the event of being shown a symbol, students had to react by performing the corresponding action. When the sign was taken down again, they resumed running until another symbol was shown.

To make the activity more challenging and to introduce chains of events, students were split into three teams for a second iteration of the game. Now each team only had to react to some of the symbols but not to all of them. Additionally, however, students did not only have to react to the event of a symbol being shown, but also to the event of another team performing a specific action. For example, when team X observes another team Y doing "jumping jacks", then team X has to react by clapping their hands three times.

At the respective end-of-lesson assessment, students had to fill out a short test with four short tasks. For each task, students had to find the correct reactions to a sequence of events. The events were represented by geometric shapes, just like in "The Big Event" game. Each event causes a picture of a specific animal to be displayed (see Figure 5). This assessment tested how well students were able to respond to individual events in a chain of events by linking them to the respective assigned reaction, which they had learned to do through playing the "Big Event" game.

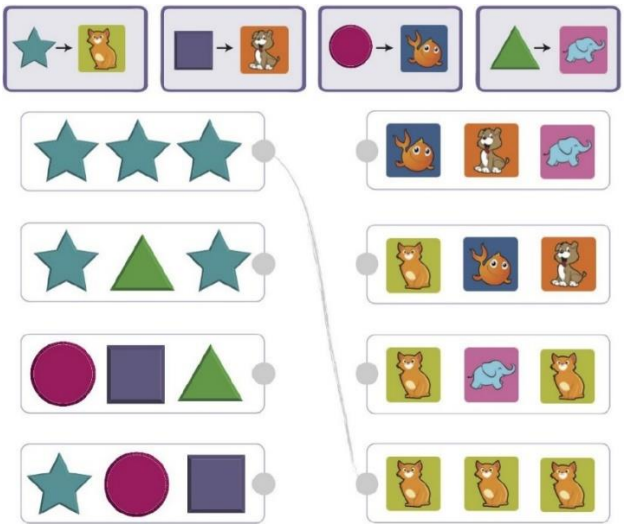

Figure 5: Exemplary task for the event lesson's assessment

3. Results

3.1 Pre- and post-course questionnaires

Out of 33 students who attended the course, 22 completed both the pre- and the post-course questionnaires. In both questionnaires, students were asked whether they could imagine working in an IT job, on a 5-point Likert scale. On average, interest in such a job decreased from M=3.27 (SD=1.39) to 2.82 (SD=1.27).

Three of the 22 students who answered both the pre- and post-course questionnaire rated their pre-course vocational interest as "not at all", and six of them as "absolutely". In both school A and school B there was one student who reported previous experience in programming. After the course, of the 22 students, 4 rated their vocational interest as "not at all", and two as "absolutely".

The post-course questionnaires were completed by 28 students. On the 5-point Likert scale, the students rated how they liked the course overall with on average M=4.36 (SD=0.89). For their self-assessed learning progress made in the course, they gave an average rating of M=4.39 (SD=0.98). The interest in learning more about computer science was rated on average with M=3.36 (SD=1.34).

3.2 The unplugged games

3.2.1 "Relay Programming" – a debugging game

Figure 6 provides an overview of how many of the students were able to solve which percentage of the assessment sheet correctly. Out of 25 students who completed the assessment at both schools combined, 16

were able to solve all tasks of the assessment correctly. 20 students' solutions were in the highest quartile of correctness. On average, students solved 89% of the assessment correctly (SD = 0.2).

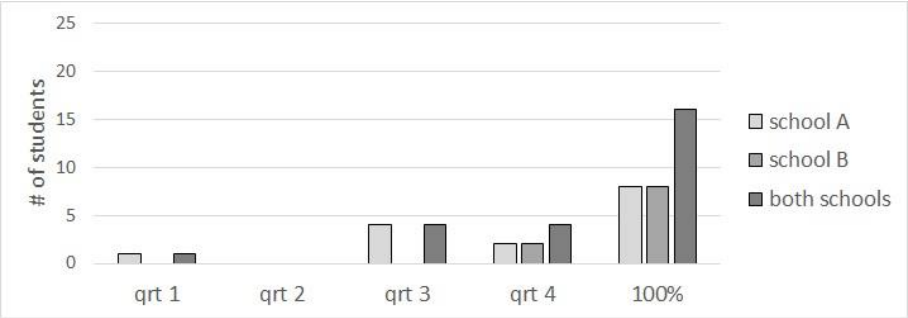

Figure 6: Results of the end-of-lesson assessment – how many students reached the debugging lesson's educational objective

3.2.2 "Conditionals with Cards" – a conditional branching game

Figure 7 provides an overview of how many of the students were able to solve which percentage of the assessment sheet correctly. Out of 27 students who completed the assessment at both schools combined, none were able to solve all tasks of the assessment correctly. 24 students' solutions were in the lowest quartile of correctness. On average, students solved 6% of the assessment correctly (SD=0.16).

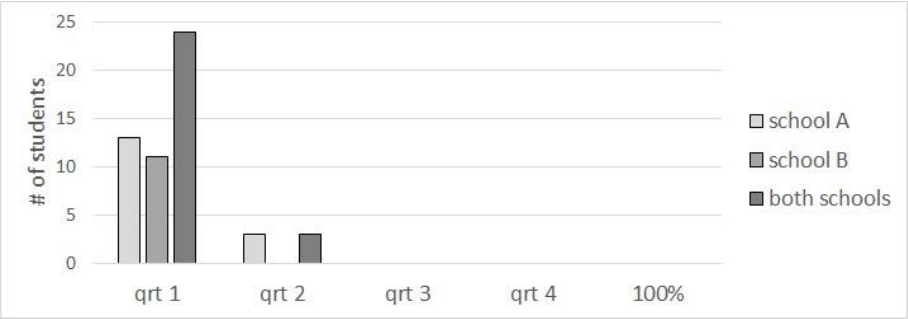

Figure 7: Results of the end-of-lesson assessment – how many students reached the conditional branching lesson's educational objective

These results conflict with our experience from the conditional branching lesson, in which we observed students were well able to understand the rules and play the "Conditionals with Cards" game. Some students had at first been confused by nested conditionals. The idea that it was not obvious at first glance which case should be followed was new to students, but by the end of the game, all students were able to follow the rules and correctly award points to the respective team.

To better understand this inconsistency between our observations from the lesson and the results of the end-of-lesson assessment, we analysed students' solutions of the tasks for types of mistakes on the level of individual turns and identified four solution types: For each turn, students either gave a solution that was (a) correct, (b) partially correct – the score was incorrectly added up due to a previous mistake –, (c) incorrect – the mistake came from following an instruction from the wrong block –, or (d) incorrect – the mistake was not in accordance with any of the instructions. We found for each respective turn, either the majority of students found the correct solution, or one specific type of mistake was dominant. There was no relevant difference in the distribution of correct solutions and types of mistakes between schools, which is why we discuss these mistakes for both schools combined.

For both turns of the *first round*, all or almost all students gave the correct solution. For the first turn of the *second round*, the dominant type of mistake was students coming to an incorrect solution as a result of following an instruction from the wrong block. In the second turn of the second round, the majority of students calculated an incorrect score as a consequence of the previous mistake. In both turns of the *last round* of the game, the prevalent type of mistakes was students specifying an incorrect solution as a result of following an instruction from the wrong block.

Possible reasons for the occurrence of these specific types of mistakes will be discussed in section 4.2 (Interpretation of the end-of-lesson assessment results).

3.2.3 "The Big Event" – an events game

Figure 8 provides an overview of how many of the students were able to solve which percentage of the respective assessment sheet correctly. Out of 21 students who completed the assessment at both schools combined, 19 were able to solve all tasks of the assessment correctly. 19 students' solutions were in the highest quartile of correctness. On average, students solved 90% of the assessment correctly (SD = 0.29).

Figure 8: Results of the end-of-lesson assessment – how many students reached the event lesson's educational objective

4. Discussion

4.1 Interpretation of questionnaire data

It is a long-term goal to get more children and students to be interested in studying computer science and in taking related jobs. Thus, the results for the questionnaire data indicating a descriptive decrease in vocational interest of students after the course seem surprising. However, it needs to be noted the difference was not significant and even more important additional qualitative feedback revealed most children liked the course, indicated they learned something valuable, and were interested in learning more about the topic. We consider this to necessitate further empirical studies to better understand this possible inconsistency. Moreover, we also think it would be desirable for future studies to use a more fine-grained questionnaire for inquiry into that problem.

More importantly, however, we actually observed another of our goals was clearly reached: Providing school children with information to be able to judge whether IT jobs might be interesting for them in a more differentiated way. As such, the descriptive mean decrease reflects a decrease in strong and possibly unsubstantiated opinions (i.e., extreme ratings on the Likert scale). In fact, 9 students (or 40% of participants) indicated strong interest or disinterest before the start of the course. Even if we assume the 2 students with previous programming experience gave their positive response on purpose, this still leaves us with 7 students (over 30%) with a strong but not possibly substantiated stance.

Finally, this data led us to be concerned that such strong opinions seem to be formed very early on. We think this again underlines the importance of starting in primary school with introducing children to CT and thus to concepts crucial for computer programming.

4.2 Interpretation of the end-of-lesson assessment results

Results of the end-of-lesson assessments on debugging and events revealed students on average reached 89% and, respectively, 90% of the lessons' learning objectives. This indicates the lessons and their central activities, the "Relay Programming" game and the "Big Event" game, were suitable for helping students understand and apply the CT concepts of debugging and events respectively.

Interestingly, at school A, only 10 out of 15 students scored in the highest quartile of the end-of-lesson assessment on debugging. It is possible the spatial limitations and the resulting aberrations from the intended gameplay at school A resulted in a smaller learning effect for students who received help and thus did not have the opportunity to actively practise applying the concept themselves.

Additionally, we observed students had specific problems with grasping the concept of conditional branching. The problems the students faced in some of the turns on the end-of-lesson assessment on conditional branching might be related to programming novices' difficulties to understand 'else' cases (Guzdial 2008) and to trace code linearly (Kaczmarczyk et al., 2010). This would explain why the students tended to follow instructions within the wrong outer branch if an inner conditional was true for the card that had been drawn. This is in line with the literature indicating conditional branching is indeed a complex concept to grasp for any computation novice (e.g., Guzdial, 2008) and requires to be dealt with in depth. Therefore, our first suggestion is to extend the teaching unit on branches from one lesson to two lessons.

During the first lesson, the concept of conditional branching might be introduced and students be familiarized with simple conditionals using the "Conditionals with Cards" game without any nested conditionals. In the second lesson then, the contents of the first lesson should first be recapitulated before introducing the concept of nested conditionals. The latter may then be practised in depth with the help of the card game from the first lesson augmented with nested conditionals. Teachers should also explain what happens when none of the conditions is met and there is no else statement, which students had difficulties to figure out themselves.

Our second suggestion would be to restructure the assessment worksheet to avoid dual-task interference – after all, the goal is to assess students' ability to trace an algorithm containing conditional branches, not their ability to perform more than one task at the same time. A restructured version of the end-of-lesson assessment on conditional branching avoiding dual-task interference and allows students to keep track of the score turn by turn rather than round by round might be more suitable.

These changes would also make it easier for teachers to evaluate students' performance on the assessment, as it would be observable at first glance how many points students awarded to which team for each card. This is not as clearly visible in the current layout of the task, which complicated assessing and interpreting the types of mistakes students made.

4.3 Limitations and future perspectives

There are several limiting factors to these interpretations of the results. It cannot be determined whether the difference in performance results might be explained by potential differences in students' cognitive abilities or socioeconomic status between the groups or by the differing temporal structures of the courses between both schools. The appropriateness of the evaluation of questionnaire data regarding students' interest in working in an IT-related job might be impaired due to CT and programming skills – which were central in the course – not being required for, and therefore not representative of, the entire IT sector. In addition, while the students who taught the children received a weekly two hours of teaching training over the course of three months prior to teaching the course, a difference in teaching quality compared to professional teachers might be expected.

The results of this pilot project – as well as its limitations – have proved helpful in informing the planning and design of a new study on teaching CT. Building on the experience gathered in this pilot project, we are currently conducting a study evaluating a game-based CT training program we developed (Tsarava et. al, 2017). To overcome such limitations as the ones described above, we are conducting a randomized controlled field trial with a pre- and post-test design using standardized and validated questionnaires and testing instruments. The CT training program is taught by professional teachers.

In conclusion, 3rd and 4th grade students seemed able to understand the respective CT concepts across all lessons and apply them while playing the unplugged games. This was reflected in the results of the post-course self-assessment questionnaire, in which students rated their course experience very positively and reported high levels of interest in learning more about computer science. Furthermore, considering students on average reached at least 82% of the learning objectives except for conditional branching, we are confident the unplugged game-based approach was motivating and beneficial for fostering the understanding of CT concepts in primary school students.

Acknowledgement

This project was supported by the Vector Foundation under the funding ID P2015-0036.

References

Balanskat, A. and Engelhardt, K. (2015) “Computing our future. Computer programming and coding – priorities, school curricula and initiatives across Europe”, *European Schoolnet*, Brussels.

Barsalou, L. W. (2008) “Grounded cognition”, *Annual review of psychology*, 59, pp. 617–645.

Code.org (n.d.). *About Us*. [online] Available at: https://code.org/about.

Code.org (n.d.) *Course 2*. [online] Available at: https://studio.code.org/s/course2.

Code.org (n.d.) “Assessment 9”, *Course 2* [online] Available at: https://code.org/curriculum/course2/9/Assessment9-RelayProgramming.pdf.

Code.org (n.d.) “Assessment 12”, *Course 2* [online] Available at: https://code.org/curriculum/course2/12/Assessment12-Conditionals.pdf.

code.org (n.d) *Evaluation Summary Report 2015 – 2016*. [online] Available at: https://code.org/files/EvaluationReport2015-16.pdf.

Guzdial, M. (2008) “Education: Paving the Way for Computational Thinking”, *Commun. ACM*, Vol. 51, No. 8, pp. 25–27.

Hamari, J., Koivisto, J., and Sarsa, H. (2014) “Does gamification work? – A literature review of empirical studies on gamification”, *Proceedings of the Annual Hawaii International Conference on System Sciences*, pp. 3025–3034.

Kaczmarczyk, L.C., Petrick, E.R., East, J.P., and Herman, G.L. (2010) “Identifying Student Misconceptions of Programming”, *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, pp. 107–111.

Kumar, D. (2014) “Welcome”, *ACM Inroads*, Vol. 5, No. 4, pp 52–53.

Lister, R. (2016) “Toward a developmental epistemology of computer programming”, *Proceedings of the 11th workshop in primary and secondary computing education*, ACM, pp. 5–16.

Ninaus, M., Pereira, G., Stefitz, R., Prada, R., Paiva, A., and Wood, G. (2015) “Game elements improve performance in a working memory training task”, *International Journal of Serious Games*, 2(1), pp. 3–16.

Noice, H. and Noice, T. (2001), “Learning dialogue with and without movement”, *Memory & Cognition*, Vol. 29, No. 6, pp. 820–827.

Papert, S. (1972) “Teaching Children Thinking”, *Programmed Learning and Educational Technology*, Vol. 9, No. 5, pp. 245–255.

Papert, S. (2005) “You can’t think about thinking without thinking about thinking about something”, Contemporary Issues in Technology and Teacher Education, Vol. 5, No. 3, pp. 366–367.

Piaget, J. (1972) “Intellectual evolution from adolescence to adulthood”, *Human development*, Vol. 15, No. 1, pp. 1–12.

Prottsman, K. (2014) “Computer Science for the Elementary Classroom”, *ACM Inroads*, Vol. 5, No. 4, pp 60–63.

Teague, D.M., Corney, M.W., Fidge, C.J., Roggenkamp, M.G., Ahadi, A., & Lister, R. (2012) “Using neo-Piagetian theory, formative in-Class tests and think alouds to better understand student thinking: a preliminary report on computer programming”, *Proceedings of 2012 Australasian Association for Engineering Education (AAEE) Annual Conference*, pp. 772–780.

Tsarava, K., Moeller, K., Pinkwart, N., Butz, M., Trautwein, U., & Ninaus, M. (2017) “Training computational thinking: Game-based unplugged and plugged-in activities in primary school”, *Proceedings of the 11th European Conference on Game Based Learning*, pp. 687-695.

Willson, M. (2017) “Algorithms (and the) everyday”, *Information, Communication & Society*, Vol. 20, No. 1, pp. 137–150.

Wing, J.M. (2006) “Computational Thinking”, *Communications of the ACM*, 49(3), pp. 33–35.

Wing, J.M. (2010) “Computational Thinking: What and Why?”, T*he Link - The Magazine of the Carnegie Mellon University School of Computer Science*, pp. 1–6.

Wing, J.M. (2014) “Computational Thinking Benefits Society”, [online], New York Academic Press, http://socialissues.cs.toronto.edu/index.html%3Fp=279.html

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., and Korb, J.T. (2014) “Computational Thinking in Elementary and Secondary Teacher Education”, *ACM Transactions on Computing Education*, 14(1), pp. 1–16.