

Teaching Programming Skills in Primary School Mathematics Classes: An Evaluation using Game Programming

Emmy-Charlotte Förster
Hastily Assembled Games, Germany
emmy@hastilyassembled.com

Klaus-Tycho Förster
University of Vienna, Austria
klaus-tycho.foerster@univie.ac.at

Thomas Löwe
Hastily Assembled Games, Germany
thomas@hastilyassembled.com

Abstract—The integration of programming into the school curriculum has become increasingly important, especially in places and class levels where computer science is not yet available as a subject of its own. In this paper we investigate the performance of a class of sixth grade students who were trained in programming as part of their regular mathematics curriculum following the method of Förster [ACM SIGITE’16], which uses programming as a teaching tool for geometry skills. As a final project the students were tasked to program a computer game in Scratch, by which we gauge the students programming skills using the methodology proposed by Funke et al. [IEEE EDUCON’17], as well as the automatic quality assessment tool Dr. Scratch. We compare our results with the results reported by Funke et al. from over 50 students, and with the automatic quality assessment scores of a data set of 250K Scratch programs published by Aivaloglou et al. [MSR’17]. Our pilot study shows that introductory programming skills taught as part of mathematics classes, aiming at the improvement of geometry skills, also satisfy the computer science requirements of an introductory programming course.

I. INTRODUCTION

Programming is a staple skill not only in computer science, but also a fundamental building block of today’s and tomorrow’s education from which all children can benefit at an early age [1]. Programming classes are already compulsory in primary school in Estonia and Cyprus [2]. Many other countries, however, only introduce computer science in secondary schools, if at all—see the snapshot by Hubwieser et al. [3]. In Germany, for example, there are no mandatory computer science requirements for primary schools [4].

Until computer science enters the curricula as a subject on its own, programming has to be either taught via extra-curricular classes/summer schools [4], [5] or integrated into existing school subjects [6]. While both methodologies yield well-documented positive outcomes, they usually suffer from the same structural downside: limited time. When computer science competes with further extra-curricular activities or the curricula of other school subjects, it becomes difficult to introduce all children to even basic programming concepts in primary schools. Even though there seems to be a consensus that “*All of Europe’s citizens need to be educated in both digital literacy and informatics*” [7], the introduction of early computer science seems to be a problem at the political level [8], which is beyond the scope of this article.

The integration of programming into a non-computer science subject is a promising approach to solving this presented dilemma, if programming simultaneously aids the foreign subject itself in its current curricular goals. Mathematics is a classical choice for such a “partner”-subject, and often responsible for bringing programming into schools [9]. Early studies have reported the benefits of using programming languages as a conceptual framework for teaching mathematics [10].

In this article we use programming as a tool to support the learning of mathematics in the 6th grade, as proposed by Förster [11]. Our motivation for this is two-fold: Firstly, it introduces the students to the programming language Scratch, which is widely popular for introductory courses in computer science nowadays [12]. Secondly, it is directly integrated into the current mathematics curriculum and was positively evaluated for “traditional” geometry goals [11].

Our goal is to evaluate whether using programming as a mathematical toolkit also satisfies the requirements of an introductory programming course in a computer science setting. Towards the end of the mathematics course, the students were tasked with programming a small computer game of their own design. We analyze these programs in order to gauge the programming skill acquired as part of the mathematics course. To this end, we follow the methodology suggested by Funke et al. [4], which is in part derived from the game-based approach of Wilson et al. [13]. We deliberately chose game programming for this task, because as Utesch points out, game creation allows the students to “*organize their own understanding of the topic and become intrinsically motivated*” [14]. Moreover, games are a type of software that all of the students were very familiar with already.

We begin by discussing related work in Section II, focusing on a K-12 education context. In Section III, we then describe our course design, which combines the teaching of mathematics with Scratch programming, and a capstone game programming task. In Section IV, we evaluate the created games and assess the children’s programming skills: first by comparison with a computer science oriented course, followed by an automated evaluation via an independent toolkit. We find that the approach satisfies the requirements of an introductory programming course. We conclude in Section V, along with an outlook on future work.

II. RELATED WORK AND BACKGROUND

Programming is seen as a fundamental 21st-century skill, and can even be considered a form of literacy [1]. For decades there have been efforts to introduce children to programming. These can be traced back as far as the late 1960's and the programming language Logo, to which Seymour Papert added the idea of Turtle graphics [15] to support drawing operations. The connection to teaching mathematics was also made early on [10], [16], [17], as was the extension to real-world drawing by a Logo-controlled robot. Even today, both Logo and Turtle graphics are still used to teach programming [18]—even to children in primary schools [5] and in kindergarten [19]. As was pointed out recently: “*The good old idea of Turtle graphics still has an enormous potential*” [20].

The modern programming language Scratch [21]¹ can be seen as a continuation of Logo's ideals [22]. Scratch employs a block-based approach, leaving behind common problems of textual programming languages such as brackets being out of sync [23]. Furthermore, Scratch is appropriate for the entire K-12 spectrum of educational children's programming and beyond—see [11, Section 3] for further details. Using concepts of Turtle graphics seems to be a common theme in introductory Scratch courses. After all, every new Scratch program already starts with a turtle object (in the form of a cat, see Figure 3).

Even though Scratch was widely released as recently as 2007, it has since become one of the staple languages used to introduce children to programming concepts. It is not only well suited as a stepping-stone into more advanced “real” programming languages [24], but also as a means to teach computer science concepts in general [25].

To this end, Maloney et al. [26] concluded: “*The Scratch programming environment and language work together to create a system that is exceptionally quick to learn—users can be programming within fifteen minutes—yet with enough depth and variety to keep users engaged for years.*”

Perhaps inspired by the constructionist goals of the Logo philosophy [27], a commonly studied theme for introductory courses is the design and programming of computer games [28], [29]. A review of various success factors when teaching with a playful approach was compiled by Heininger et al. in [30]. Various studies focus on learning programming (solely) through the making of computer games—we refer to the survey of Kafai and Burke for an overview [28].

It has been shown that game programming, in an introductory context for children, raises motivation and is additionally beneficial for female students [31]. Ouahbi et al. [32] also remark on the high motivation of students in this context.

However, it is important to note that programming games is orthogonal to playing games [33]. Game programming is an educational tool that is used in early education [34] or specifically the mathematics curriculum [35], though it is related to the more specific task of programming agents in a game [36], [37]. The latter also has the benefit that debugging can be performed “*by observing the agents' actions in a*

virtual [computer game] world” [38], analogous to coding and debugging Turtle graphics.

In general, there is a vast corpus of work regarding the teaching of programming, games programming, and the interplay of programming and mathematics. In the following, we will therefore focus on related work that—like the approach investigated here—uses the programming language Scratch.

Among the most important topics is the assessment of Scratch programs created by students. Moreno-León and Robles [39] propose automated evaluation via a tool they call Dr. Scratch². Funke et al. [4], on the other hand, advocate to manually evaluate the student programs via the SOLO taxonomy [40], [41]. We present both evaluation concepts in greater detail in Section IV, where we use them for evaluation and comparison purposes.

Funke et al.'s approach is embedded into a larger project idea, first proposed in [42], in which the goal is to develop and run an efficient extra-curricular introductory programming course for 3rd and 4th grade students. The project concept and execution is described in [43]: over the course of three days, the students are first exposed to programming via the concepts of CS unplugged [44], then to programming at a computer in the Scratch environment (also introducing loops and if-conditions) and are finally tasked to program personalized projects. In [4], Funke et al. evaluate over 50 projects of 4th-graders. They group these projects into three genres: stories, animations, and games—where games are found to be the most advanced projects overall. They also investigate gender differences between the Scratch programs [45].

We follow their (capstone) method for evaluation in our own course design, but specifically task students to program personalized computer games for their projects.

Scratch has also been used in the context of further K-12 subjects [6], and particularly in investigating the connection between programming and mathematics skilly. Lewis and Shah [46] showed that for 5th graders, scores on Scratch programming were predicted by their previous performances in standardized mathematics tests. For 6th graders, Calao et al. [47] demonstrated that training in Scratch increased their mathematical thinking skills in a significant way. Experiments regarding the learning of probabilities through game creation in Scratch were performed in [48].

Previously, Förster investigated the integration of Scratch into geometry lessons for students in grade 6 and 7, showing that the mathematical knowledge objectives can be reached in an improved way in comparison to control groups using traditional (non-programming) methods [11]. Beyond this, we are not aware of further works that 1) replace part of mathematics classes with specifically tailored programming in Scratch, and 2) obtain enhanced results in mathematics objectives in comparison to traditional teaching methods. In this paper we evaluate this method—limited to grade 6—with a particular emphasis on determining the students' (implicitly) obtained programming skills.

¹ <http://scratch.mit.edu>

² <http://www.drscratch.org/>



Fig. 1: Classroom setting during the course.

III. COURSE DESIGN AND EXECUTION

In this section, we describe the arrangement of our course. We begin by outlining our conceptual choices and the classroom setting, followed by the design and execution of the course, and finally our evaluation methods.

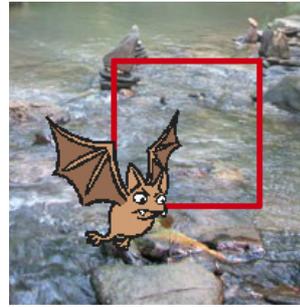
The aim of our course was to integrate an introductory programming course into a mathematics curriculum. To this end, we follow the approach of Förster [11], which was developed to integrate programming and algorithms into the mathematics curriculum of grades 6 and 7. In order to evaluate whether this mathematics course also fulfilled the goals of an introductory programming course, the students were tasked to program a computer game of their own design as a capstone project. These games were then analyzed to gauge the students' programming skills, by applying the evaluation methodology suggested by Funke et al. [4], which is in part derived from the game-based approach of Wilson et al. [13].

The course was held at a German school by a female teacher experienced in both mathematics and computer science. The class consisted of 22 6th graders (14 girls, 8 boys, all aged 11–12 years). The school did not offer computer science as a subject prior to the 6th grade, such that nearly all students had no prior programming experience. Programming was mostly performed in pairs, due to limitations of the facilities, as can be seen in Figure 1. While both the instructions and the programming interface were originally in German, the Scratch interface was changed to English in all screenshots and code snippets, for the purpose of this article.

A. Programming Integrated into the Mathematics Curriculum

The approach proposed by Förster [11] integrates programming into the mathematics curriculum of the 6th grade, specifically into geometry lessons. Note that this approach is not in itself an introductory programming course, but rather uses programming as a mere tool to reach mathematics objectives.

The approach is structured into four lectures of 45 minutes each, taught over two subsequent days, and encompasses the mathematics topics of *polygons* and *tessellations*. In the first two lectures, the students were first briefly introduced to the Scratch programming environment via basic sprite movements and were tasked with the construction of triangles and regular polygons in Turtle graphics [15]. Note that variables were not introduced, as can be seen in the example in Figure 2.



```

when clicked
  pen down
  set pen color to red
  change pen size by 3
  repeat 4
    move 100 steps
    turn 180 - 90 degrees
  
```

Fig. 2: Example of a program generating a square in Turtle graphics. Many children chose to personalize their programs. In this case, the female coder chose to let a bat draw the square in a wilderness environment.

In the following two lectures, the students generated tessellations consisting of triangles, squares, and hexagons. Analogous to the observations made by Förster [11], the students were able to follow the course requirements, though the tessellations were more difficult than the earlier coding tasks and required more iterations. An example is shown in Figure 3.

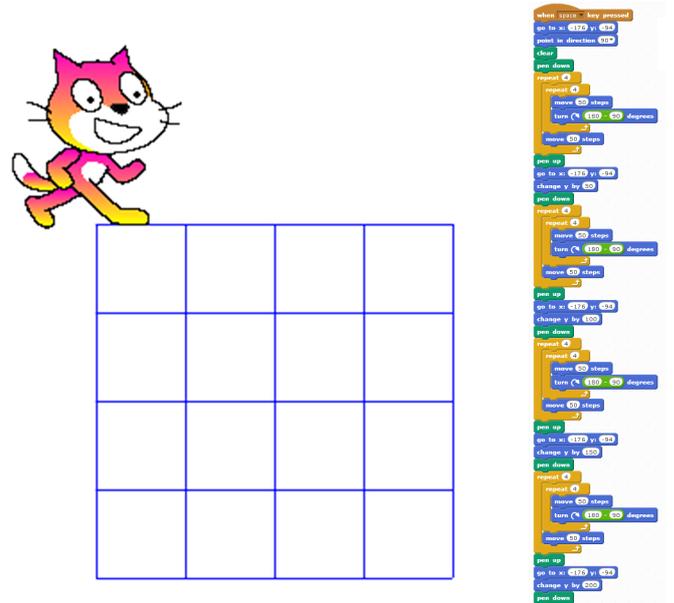


Fig. 3: Iteration example of a square tessellation. The code generating four squares next to each other was pasted identically four times. As a next step, the code can be simplified.

B. Game-based Evaluation of Programming Skills

We delayed the capstone evaluation by roughly six weeks to test for longer lasting impact. Moreover, all grades were already finalized by that point—the students were thus aware that their performance would have no impact on their grades. During that delay, there were no additional programming elements in the students' school curriculum. For their final capstone projects, the students had a little over one hour to design and program their game in Scratch.

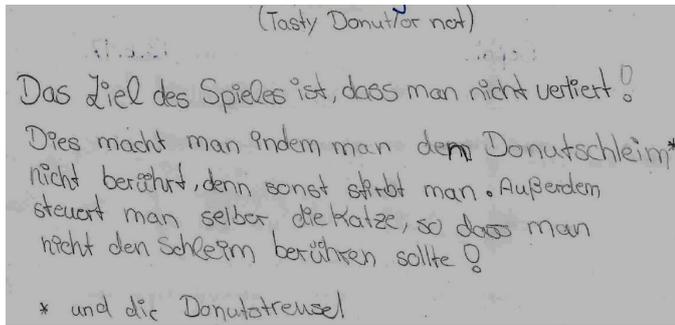


Fig. 4: Instruction manual for the game *Tasty Donut/or not*. Translation: “The goal of the game is not to lose! This is accomplished by avoiding the Donut-slime*, because otherwise you will die. In addition, you control the cat [the avatar of their game] yourself, so you should not touch the slime! *and the Donut-sprinkles”

As described in the work of Funke et al. [4], the students were given simple instructions listing the mandatory requirements for their programming projects. These consisted of the following four items:

- the project must contain multiple sprites
- every sprite must move at least once or be controllable
- the project must contain at least one loop
- the project must contain at least one if-condition

Additionally, the students were tasked with writing a brief instruction manual for their game. Note that there were no further requirements restricting the type or genre of the game. As there was a long pause between the course and the capstone project, an independent minimum working example was provided for each of the four requirements (each ranging from two to four blocks), e. g. *when the space key is pressed* (block 1), *change pen color by 10* (block 2).

All students were able to finish their games in the allotted time, and also composed written instructions on how to play their games, see Figure 4 for an example. In the following section, we provide a detailed evaluation of these game projects.

IV. RESULTS

We evaluated 12 game projects, each created by a group of 1–3 students in 6th grade. While the students were free to choose their own partners, each group ended up consisting of either all female or all male students.

We deliberately chose game programming for this exercise, since computer games are a type of software that all of the students were very familiar with already, which allowed us to forgo complex task descriptions. Instead, the complexity and theme of the game were left up to the students, which allowed them to “organize their own understanding of the topic” [14]—within the requirements discussed in Section III-B.

Despite a lack of formal education in game creation, all student groups assessed their programming skills reasonably well and created small arcade-like games of varying complexity. All games were functional and playable.

We analyze the structure and quality of these game projects using a category system presented by Funke et al. [4], which is based on previous category systems [13], [49]. The system is designed to measure computational thinking skills by coding each project for the presence of 24 program elements, subdivided into four main categories:

1. *Requirements*:
several sprites, sprite motion, iteration, conditional statement
2. *Programming Concepts*:
sequence, variables, lists, event handling, threads, condition and synchronization, keyboard input, random numbers, Boolean logic, dynamic interaction
3. *Code Organization*:
extraneous blocks, sprite names, variable names
4. *Operability*:
functionality, sprite customization, stage customization, interactivity, usability, project type

Moreover, a *level of understanding* is determined for each project using a modification of the SOLO taxonomy [40], [41]: prestructural (L1), unistructural (L2), multistructural (L3), relational (L4), extended abstract (L5).

In addition to this category system, we use the automated quality assessment tool Dr. Scratch [50], [51] to statistically inspect the source code of each game project. The tool assigns point scores from 0 to 3 points on seven dimensions of computational thinking: 1) *abstraction and problem decomposition*, 2) *logical thinking*, 3) *synchronization*, 4) *parallelism*, 5) *algorithmic notions of flow control*, 6) *user interactivity*, and 7) *data representation*. The sum of these seven scores make up an overall score, which ranges from 0 to 21 points.

A. Project Examples

While the students were not given any particular themes for their projects, the finished games can be broadly classified among the following four game genres: *breakout*, *racing*, *shell game*, and *collect/evade*. We note that there were gender disparities present in the genres chosen, see Table I. All of the games categorized in the *collect/evade* genre were created by groups of female students, while all of the games categorized in the *racing* genre were created by groups of male students. This is particularly striking, as these two genres are also among the most popular overall.

TABLE I: Games classified by genre

Genre	Total	% Female	% Male
Collect/Evade	5	100%	0%
Racing	3	0%	100%
Breakout	3	33.3%	66.6%
Shell Game	1	100%	0%

In the following, we give a detailed view on two of the games, in order to provide a sense for the type and code complexity of the created projects.

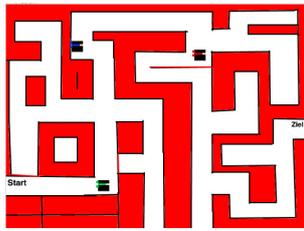
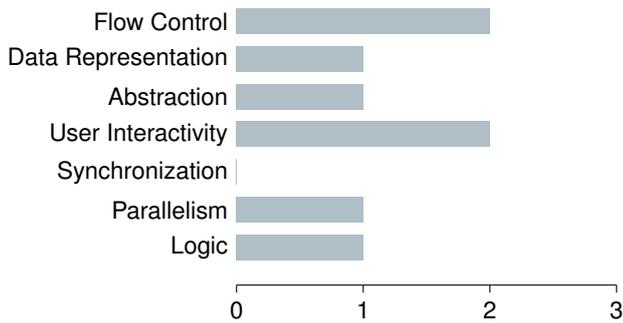
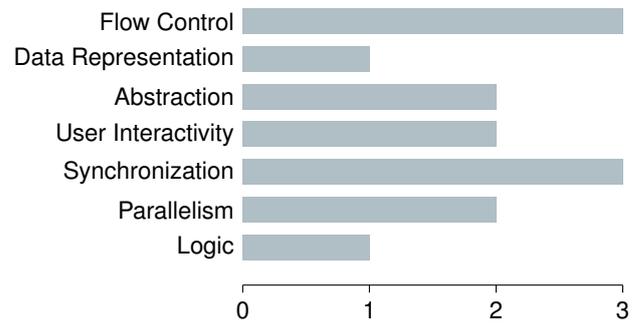




Fig. 5: From top to bottom: Stage screenshot, scratch code and automated Dr. Scratch ratings of “Labyrinth Race”, a multiplayer racing game for up to three players.

Fig. 6: From top to bottom: Stage screenshot, scratch code and automated Dr. Scratch ratings of “Tasty Donut/or not”, a game about evading donuts and sprinkles.

1) *Labyrinth Race*: A multiplayer racing game for up to three players—the first player to reach the exit of the labyrinth wins. If a player collides with a wall of the labyrinth their position is set back to the start. A screenshot and the code of the project can be found in Figure 5. The students used a total of 51 blocks across three sprites (players). The code in each of the three player sprites is identical, except for the input keys—as can be expected in a balanced multiplayer game design.

While the accompanying game description that the students provided mentions a win-state, this has not been implemented in the actual program. However, given that the students knew how to implement players being reset back to the start upon touching a wall (fail-state), it is reasonable to expect that implementing the win-state would have been in their skill set.

This project fulfils all of the task requirements. It uses sequences, threads, keyboard inputs, and dynamic interactions. There are no extraneous blocks and no unnamed sprites. It is fully functional, has user interactions, and its usability is intuitive and documented.

The project scored an overall rating of 8 out of 21 in the automated evaluation using Dr. Scratch (Figure 5). It scored lowest (0 out of 3 points) in Synchronization, as there were no synchronization structures used (wait, when I receive, ...).

2) *Tasty Donut/or not*: A game in which the player controls a cat and has to evade three floating objects (donuts and sprinkles). If the player touches any of the objects, the game stops and a game over message is displayed. A screenshot and the code of the project can be found in Figure 6. The students used a total of 42 blocks across three sprites. The accompanying instructions that the students wrote for the game can be found in Figure 4.

Pressing the space key starts the game—and also restarts it after an object hit the player. Unfortunately, the sprites do not stop when the game does and are not reset upon restart. Additionally, as the students have not learned the concept of random numbers yet, the movement patterns for each moving object are deterministic and hard-coded. The game can thus be easily beaten by moving the player to the lower right of the game stage, as none of the objects will ever move there.

This project fulfils all of the task requirements. It uses sequences, event handling, threads, coordination, keyboard inputs and dynamic interactions. There are no extraneous blocks and no unnamed sprites. It is fully functional, has user interactions and its usability is intuitive and documented.

The project scored an overall rating of 14 out of 21 in the automated evaluation using Dr. Scratch (Figure 6).

B. Code Structure and Quality

We now examine the code structure and quality of the students' game projects as outlined in the beginning of Section IV, in order to assess the student's computational thinking skills. We first evaluate the four main categories regarding the presence of program elements, followed by the level of understanding according to the modified SOLO taxonomy.

1. Requirements

All student projects met the four minimal requirements discussed in Section III-B.

2. Programming Concepts

All student projects used *sequences*, *event handling* with at least 3 events (average: 10), *threads*, *keyboard input*, and *dynamic interaction*. One student project used the concept of *random numbers* and two projects used *Boolean logic* and *variables*. No project used *lists*.

3. Code Organization:

In all projects sprites and variables were named in a meaningful way. 40% of the projects included extraneous blocks.

4. Operability:

All projects (games) were fully functional and included user input. The input schemes were intuitive as they were similar to standard keyboard control schemes in games.

When evaluating the *level of understanding* according to the modified SOLO taxonomy we found that all game projects qualified for at least the third level L3 (*Multistructural*). 25% of all projects were categorized in the fourth level L4 (*Relational*), and 30% of all projects demonstrated qualities of the highest level L5 (*Extended Abstract*).

Overall, the student projects of our course received at least similar scores to the scores reported in Funke et al. [4], indicating that the students in our course had learned programming on at least the same level. We note that our students were in the 6th grade as opposed to the 4th grade, but on the other hand, the 6th grade students were instructed in 4×45 minutes plus the final game project six weeks later, as opposed to a more comprehensive 3×4 hours course. It is, however, not our goal to rank both courses: We are merely interested whether the success of Funke et al. can be replicated in the context of

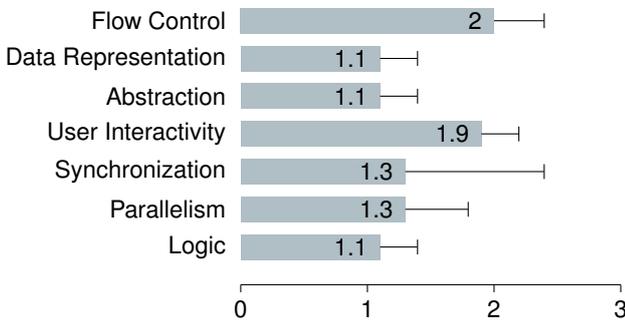


Fig. 7: Average ratings of the student projects according to the automated Dr. Scratch evaluation tool. Each dimension is individually ranked in a range of 0 to 3 points.

the mathematics curriculum at a later stage for primary school children. We believe that the course design of Funke et al. is exemplary in providing an introduction to programming for primary school children.

As all of the above evaluation methods required manual analysis, we additionally included a comparable automatic rating method. We chose Dr. Scratch, since both the methodology of Funke et al. as well as the automatic evaluation algorithm of Dr. Scratch are at least partly based off of the work of Wilson et al. [13]. The Dr. Scratch tool automatically evaluates a Scratch program and scores it along the seven dimensions of computational thinking. Figure 7 shows the automatically computed mean scores over all student projects.

Each student project achieved a score of at least one in each of the seven categories, except for the Synchronization category. Three games (2 racing games, 1 breakout game) did not reach a score of one in Synchronization. A score of one can for example be achieved by the use of *wait* blocks, which can be found in 50% of the games, see Table II. Higher scores can be achieved with blocks like *Broadcast, when I receive message, stop all, wait until* etc.

Table II shows that 83.3% of the student projects used the *key pressed* block. The use of this block leads to a score of two in User Interactivity. The project which scored one point in his category was the shell game. Here the user interactivity is given by the *touching* block, which makes more sense regarding the game design.

Overall, the automatically generated Dr. Scratch scores are similar to the manually obtained scores in the category system. To put the automated results into further context, we additionally compared the results to a large the data set provided by Aivaloglou et al. [52], which consists of 250K Scratch projects of 100K different authors scraped from the Scratch repository.

TABLE II: Top ten blocks in the students' projects, including the percentage of projects the block occurs in.

Block	Total	% blocks	% projects
	104	12.9%	100%
	92	11.4%	83.3%
	64	7.9%	75%
	60	7.4%	50%
	59	7.3%	50%
	53	6.6%	83.3%
	50	6.2%	75%
	49	6.1%	66.6%
	40	5%	100%
	37	4.6%	25%

This data set also holds programming mastery scores generated by the Dr. Scratch quality assessment tool, as well as average numbers of blocks used.

The average number of blocks used for projects in this data set was 144.3 blocks with a median of 29 blocks. In comparison, the students in our experiment used an average of 67 blocks (median: 54) in their game projects.

The mean score of mastery per project in the data set was 8.9 points (median: 8). Here the students in our experiment achieved a higher mean mastery score at an average of 9.7 points (median: 9), with the lowest score being 8 points and the highest being 14 points. According to the Dr. Scratch documentation, a mastery score of 8 points is considered to show a medium level of computational thinking skills development [52].

Overall, our evaluation of the student projects with the methodology of Funke et al. [4] and Dr. Scratch show that the students developed at least a medium level of computational thinking skills, indicating that the mathematical approach described by Förster [11] also satisfies the requirements of an introductory programming course.

C. Gender Differences

In this section, we investigate the differences regarding gender in the projects. Recall that all games were created by female-only and male-only groups. During our evaluation we did notice certain gender specific differences.

Regarding the Dr. Scratch evaluation, we noticed that the projects of the female groups reached an average computational thinking score of 10.6, whereas the male groups reached an average score of 8.6. The average scores for the seven dimensions of computational thinking separated by gender can be found in Figure 8. Noticeable are the different average scores in the categories Synchronization and Parallelism.

As mentioned above, three games developed by male students (2 racing games, 1 breakout game) did not reach a score of one in Synchronization. The *wait* block, for example, was used in only 20% of the male projects (Table III), as opposed to 71.4% in the female projects (Table IV). However, the *wait* block was mostly used for visual effects, which occurred more often in the female-developed collect/evade games.

All projects scored at least one point in Parallelism, which is directly connected to the multiple usage of the *green flag, key pressed, when this Sprite is clicked, when I receive* blocks on the same sprite. Here, female projects from the collect/evade genre scored the highest, achieving up to two points. We observed that the score disparities seem to be largely influenced by the gender disparities in the self-selected genres, see Table I. For example all racing games received a Dr. Scratch rating of 8, while the collect/evade games had an average score of 11.25 (lowest 9). The differences in the used blocks in male and female projects (see Tables III and IV) seem to stem from the different requirements of the selected genres. Regarding the level of understanding (SOLO), we found that the female projects reached an average level of 4 and the male projects reached an average level of 4.25. However, this difference is

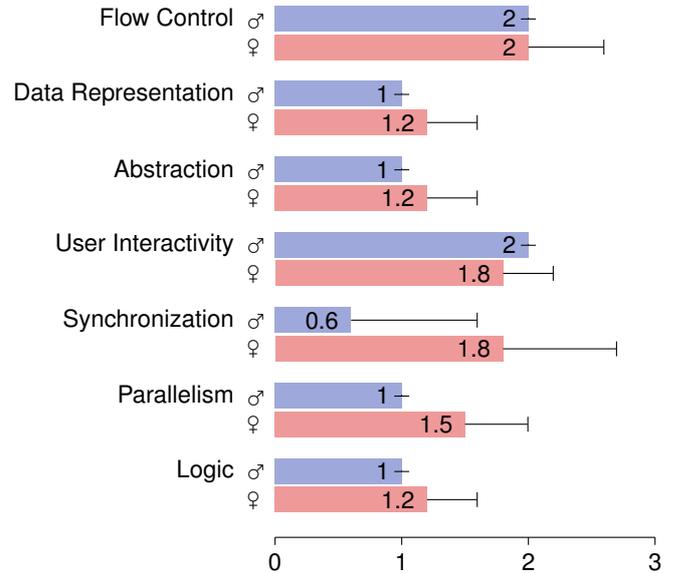


Fig. 8: Gender separated average ratings of the student projects according to the automated Dr. Scratch evaluation tool. Each dimension is individually ranked in a range of 0 to 3 points.

too small to allow for any qualitative distinction between the different genders.

In order to perform a more comprehensive gender-based evaluation of programming skills using Dr. Scratch, it may be necessary to prescribe a game genre for the final project. Given the gender difference in the genre-choices, however, it could be important to investigate whether certain genres may have a negative influence on the motivation of either female or male students.

V. CONCLUSION AND OUTLOOK

The results of our pilot study indicate that an introduction of 6th grade students to programming can be performed in a way that satisfies both:

- 1) the curricular goals of the mathematics classes in which the introduction to programming is performed, and
- 2) the requirements of a computer science introductory course on programming in Scratch.

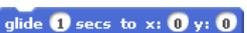
We have provided a successful course example, which demonstrates how 6th grade students can be introduced to computer science in school without having to sacrifice scarce extra-curricular class hours. We believe the latter to be especially important, as it allows all school children to obtain programming skills, even if computer science is not available as a school subject. Furthermore, the game-oriented capstone approach enabled the children to express their programming skills in an intuitive and meaningful way.

As a next step, we plan to expand this approach to additional groups of children and other parts of the K-12 curriculum. We also plan to investigate integrating programming into non-mathematics subjects, as well as incorporating further game-development aspects [28], [30].

TABLE III: Top ten blocks in the male students' projects, including the percentage of projects the block occurs in. The sum of all used blocks is 254. with an average of 50.8 blocks per project.

Block	Total	% blocks	% projects
	55	21.6%	100%%
	53	20.8%	100%%
	32	12.5%	60%
	32	9.4%	60%
	21	8.3%	40%
	19	7.5%	100%
	15	5.9%	100%%
	15	5.9%	100%
	3	1.2%	40%
	2	0.8%	20%

TABLE IV: Top ten blocks in the female students' projects, including the percentage of projects the block occurs in. The sum of all used blocks is 550 with an average of 78.5 blocks per project.

Block	Total	% blocks	% projects
	58	10.5%	71.4%
	51	9.3%	100%
	40	7.3%	85.7%
	38	6.9%	74.1%
	37	6.7%	71.4%
	35	6.4%	57.1
	35	6.4%	28.6
	28	5.1%	3.6%
	27	4.9%	42.9%
	21	3.8%	100%

REFERENCES

- [1] Y. B. Kafai, Q. Burke, and M. Resnick, *Connected Code: Why Children Need to Learn Programming*. MIT Press, 2014. [Online]. Available: <http://www.jstor.org/stable/j.ctt9qf8rk>
- [2] C. Duncan and T. Bell, "A pilot computer science and programming course for primary school students," in *Proceedings of the Workshop in Primary and Secondary Computing Education, WiPSCE 2015, London, United Kingdom, November 9-11, 2015*, J. Gal-Ezer, S. Sentance, and J. Vahrenhold, Eds. ACM, 2015, pp. 39–48. [Online]. Available: <http://doi.acm.org/10.1145/2818314.2818328>
- [3] P. Hubwieser, M. N. Giannakos, M. Berges, T. Brinda, I. Diethelm, J. Magenheimer, Y. Pal, J. Jacková, and E. Jasute, "A global snapshot of computer science education in K-12 schools," in *Proceedings of the 2015 ITiCSE Working Group Reports, ITiCSE-WGR 2015, Vilnius, Lithuania, July 4-8, 2015*, N. Ragonis and P. Kinnunen, Eds. ACM, 2015, pp. 65–83. [Online]. Available: <http://doi.acm.org/10.1145/2858796.2858799>
- [4] A. Funke, K. Geldreich, and P. Hubwieser, "Analysis of scratch projects of an introductory programming course for primary school students," in *2017 IEEE Global Engineering Education Conference, EDUCON 2017, Athens, Greece, April 25-28, 2017*. IEEE, 2017, pp. 1229–1236. [Online]. Available: <https://doi.org/10.1109/EDUCON.2017.7943005>
- [5] G. Serafini, "Teaching programming at primary schools: Visions, experiences, and long-term research prospects," in *Informatics in Schools. Contributing to 21st Century Education - 5th International Conference on Informatics in Schools: Situation, Evolution and Perspectives, ISSEP 2011, Bratislava, Slovakia, October 26-29, 2011*. Proceedings, ser. Lecture Notes in Computer Science, I. Kalas and R. T. Mittermeir, Eds., vol. 7013. Springer, 2011, pp. 143–154. [Online]. Available: https://doi.org/10.1007/978-3-642-24722-4_13
- [6] J. Moreno-León and G. Robles, "Code to learn with scratch? A systematic literature review," in *2016 IEEE Global Engineering Education Conference, EDUCON 2016, Abu Dhabi, United Arab Emirates, April 10-13, 2016*. IEEE, 2016, pp. 150–156. [Online]. Available: <https://doi.org/10.1109/EDUCON.2016.7474546>
- [7] W. Gander, A. Petit, G. Berry, B. Demo, J. Vahrenhold, A. McGettrick, R. Boyle, A. Mendelson, C. Stephenson, C. Ghezzi *et al.*, "Informatics education: Europe cannot afford to miss the boat," *ACM Europe: Informatics education report*, 2013. [Online]. Available: <http://europe.acm.org/iereport/ie.html>
- [8] I. Livingstone, "A new year challenge on programming for politicians, schools and universities," <https://www.theguardian.com/education/2012/jan/10/a-new-year-challenge-on-programming-for-politicians-schools-and-universities>, 10 January 2012, the Guardian.
- [9] J. Ziegenbalg, "Informatik-affine themen in der didaktik der mathematik," *Mitteilungen der Gesellschaft fuer Didaktik der Mathematik*, vol. 40, no. 96, pp. 7–14, 2014. [Online]. Available: <https://ojs.didaktik-der-mathematik.de/index.php/mgdm/article/view/363>
- [10] W. Feurzeig, S. Papert, M. Bloom, R. Grant, and C. Solomon, "Programming-languages as a conceptual framework for teaching mathematics," *SIGCUE Outlook*, vol. 4, no. 2, pp. 13–17, Apr. 1970. [Online]. Available: <http://doi.acm.org/10.1145/965754.965757>
- [11] K.-T. Foerster, "Integrating Programming into the Mathematics Curriculum: Combining Scratch and Geometry in Grades 6 and 7," in *Proceedings of the 17th Annual Conference on Information Technology Education*, ser. SIGITE '16, D. Boisvert and S. J. Zilora, Eds. New York, NY, USA: ACM, 2016, pp. 91–96. [Online]. Available: <http://doi.acm.org/10.1145/2978192.2978222>
- [12] S. Y. Lye and J. H. L. Koh, "Review on teaching and learning of computational thinking through programming: What is next for k-12?" *Computers in Human Behavior*, vol. 41, pp. 51–61, 2014. [Online]. Available: <https://doi.org/10.1016/j.chb.2014.09.012>
- [13] A. Wilson, T. Hainey, and T. M. Connolly, "Using scratch with primary school children: An evaluation of games constructed to gauge understanding of programming concepts," *IJGBL*, vol. 3, no. 1, pp. 93–109, 2013. [Online]. Available: <https://doi.org/10.4018/ijgb.2013010107>
- [14] M. C. Utesch, "The special track 'games engineering' at EDUCON 2017," in *2017 IEEE Global Engineering Education Conference, EDUCON 2017, Athens, Greece, April 25-28, 2017*. IEEE, 2017, pp. 1883–1884. [Online]. Available: <https://doi.org/10.1109/EDUCON.2017.7943110>
- [15] H. Abelson and A. A. DiSessa, *Turtle geometry : the computer as a medium for exploring mathematics*, ser. The MIT Press series in artificial intelligence. Cambridge, Mass. MIT Press, 1981. [Online]. Available: <https://mitpress.mit.edu/books/turtle-geometry>
- [16] J. Howe, F. Plane, and T. O'Shea, "Teaching mathematics through logo programming : an evaluation study," University of Edinburgh (Edinburgh, GB), Tech. Rep. DAI-RP-115, 1979.
- [17] J. Howe, P. Ross, K. Johnson, F. Plane, and R. Inglis, "Teaching mathematics through programming in the classroom," *Computers & Education*, vol. 6, no. 1, pp. 85 – 91, 1982. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/0360131582900161>

- [18] J. Hromkovic, *Einfuehrung in die Programmierung mit LOGO: Lehrbuch fuer Unterricht und Selbststudium*, 2nd ed. Wiesbaden: Vieweg+Teubner, 2012. [Online]. Available: <https://dx.doi.org/10.1007/978-3-8348-2266-6>
- [19] G. Fessakis, E. Gouli, and E. Mavroudi, "Problem solving by 5-6 years old kindergarten children in a computer programming environment: A case study," *Computers & Education*, vol. 63, pp. 87–97, 2013. [Online]. Available: <https://doi.org/10.1016/j.compedu.2012.11.016>
- [20] R. Oldenburg, M. Rabel, and J. Schuster, "A Turtle's Genetic Path to Object Oriented Programming," in *Proceedings to Constructionism, 2012*. [Online]. Available: <http://constructionism2012.etl.ppp.uoa.gr/-pid=31.htm>
- [21] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009. [Online]. Available: <http://doi.acm.org/10.1145/1592761.1592779>
- [22] R. Romeike, "Constructionist approaches to mathematics education and mathematics teacher education (Konstruktionistische Ansätze fuer Mathematikunterricht und Mathematiklehrausbildung)," in *Beitraege zum Mathematikunterricht 2011, 45. Jahrestagung der Gesellschaft fuer Didaktik der Mathematik vom 21. bis 25. Februar 2011 in Freiburg, 2011*. [Online]. Available: <http://dx.doi.org/10.17877/DE290R-13680>
- [23] I. Utting, S. Cooper, M. Kölling, J. Maloney, and M. Resnick, "Alice, greenfoot, and scratch - A discussion," *TOCE*, vol. 10, no. 4, pp. 17:1–17:11, 2010. [Online]. Available: <http://doi.acm.org/10.1145/1868358.1868364>
- [24] M. Armoni, O. Meerbaum-Salant, and M. Ben-Ari, "From scratch to "real" programming," *TOCE*, vol. 14, no. 4, pp. 25:1–25:15, 2014. [Online]. Available: <http://doi.acm.org/10.1145/2677087>
- [25] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Learning computer science concepts with scratch," *Computer Science Education*, vol. 23, no. 3, pp. 239–264, 2013. [Online]. Available: <https://doi.org/10.1080/08993408.2013.832022>
- [26] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *TOCE*, vol. 10, no. 4, pp. 16:1–16:15, 2010. [Online]. Available: <http://doi.acm.org/10.1145/1868358.1868363>
- [27] S. Papert et al., "Logo philosophy and implementation," *Logo Computer Systems Inc*, 1999. [Online]. Available: <http://www.microworlds.com/company/philosophy.pdf>
- [28] Y. B. Kafai and Q. Burke, "Constructionist gaming: Understanding the benefits of making games for learning," *Educational Psychologist*, vol. 50, no. 4, pp. 313–334, 2015, pMID: 27019536. [Online]. Available: <http://dx.doi.org/10.1080/00461520.2015.1124022>
- [29] Y. B. Kafai, *Minds in Play: Computer Game Design As a Context for Children's Learning*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1995. [Online]. Available: <https://dl.acm.org/citation.cfm?id=527173>
- [30] R. Heininger, L. Prifti, V. Seifert, M. C. Utesch, and H. Krcmar, "Teaching how to program with a playful approach: A review of success factors," in *2017 IEEE Global Engineering Education Conference, EDUCON 2017, Athens, Greece, April 25-28, 2017*. IEEE, 2017, pp. 189–198. [Online]. Available: <https://doi.org/10.1109/EDUCON.2017.7942846>
- [31] W. S. Yue and W. L. Wan, "The effectiveness of digital game for introductory programming concepts," in *10th International Conference for Internet Technology and Secured Transactions, ICITST 2015, London, United Kingdom, December 14-16, 2015*. IEEE, 2015, pp. 421–425. [Online]. Available: <https://doi.org/10.1109/ICITST.2015.7412134>
- [32] I. Ouahbi, F. Kaddari, H. Darhmaoui, A. Elachqar, and S. Lahmine, "Learning basic programming concepts by creating games with scratch programming environment," *Procedia - Social and Behavioral Sciences*, vol. 191, no. Supplement C, pp. 1479 – 1482, 2015, the Proceedings of 6th World Conference on educational Sciences. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1877042815024842>
- [33] T. M. Connolly, E. A. Boyle, E. MacArthur, T. Hainey, and J. M. Boyle, "A systematic literature review of empirical evidence on computer games and serious games," *Computers & Education*, vol. 59, no. 2, pp. 661–686, 2012. [Online]. Available: <https://doi.org/10.1016/j.compedu.2012.03.004>
- [34] T. Hainey, T. M. Connolly, E. A. Boyle, A. Wilson, and A. Razak, "A systematic literature review of games-based learning empirical evidence in primary education," *Computers & Education*, vol. 102, pp. 202–223, 2016. [Online]. Available: <https://doi.org/10.1016/j.compedu.2016.09.001>
- [35] M. Kebritchi, A. Hirumi, and H. Bai, "The effects of modern mathematics computer games on mathematics achievement and class motivation," *Computers & Education*, vol. 55, no. 2, pp. 427–443, 2010. [Online]. Available: <https://doi.org/10.1016/j.compedu.2010.02.007>
- [36] J. E. Laird, "Using a computer game to develop advanced AI," *IEEE Computer*, vol. 34, no. 7, pp. 70–75, 2001. [Online]. Available: <https://doi.org/10.1109/2.933506>
- [37] K.-T. Foerster, M. Koenig, and R. Wattenhofer, "A Concept for an Introduction to Parallelization in Java: Multithreading with Programmable Robots in Minecraft," in *Proceedings of the 17th Annual Conference on Information Technology Education*, ser. SIGITE '16, D. Boisvert and S. J. Zilora, Eds. New York, NY, USA: ACM, 2016, p. 169. [Online]. Available: <http://doi.acm.org/10.1145/2978192.2978243>
- [38] K.-T. Foerster, "Teaching spatial geometry in a virtual world: Using minecraft in mathematics in grade 5/6," in *2017 IEEE Global Engineering Education Conference, EDUCON 2017, Athens, Greece, April 25-28, 2017*. IEEE, 2017, pp. 1411–1418. [Online]. Available: <https://doi.org/10.1109/EDUCON.2017.7943032>
- [39] J. Moreno-León and G. Robles, "Automatic detection of bad programming habits in scratch: A preliminary study," in *IEEE Frontiers in Education Conference, FIE 2014, Proceedings, Madrid, Spain, October 22-25, 2014*. IEEE, 2014, pp. 1–4. [Online]. Available: <https://doi.org/10.1109/FIE.2014.7044055>
- [40] L. M. Seiter, "Using SOLO to classify the programming responses of primary grade students," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education, Kansas City, MO, USA, March 4-7, 2015*, A. Decker, K. Eiselt, C. Alphonse, and J. Tims, Eds. ACM, 2015, pp. 540–545. [Online]. Available: <http://doi.acm.org/10.1145/2676723.2677244>
- [41] J. B. Biggs and K. F. Collis, *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, 1982. [Online]. Available: <http://www.sciencedirect.com/science/book/9780120975525>
- [42] K. Geldreich, A. Funke, and P. Hubwieser, "A programming circus for primary schools," *ISSEP 2016*, pp. 49–50, 2016. [Online]. Available: http://issep2016.ens-cachan.fr/ISSEP_2016_Proceedings.pdf
- [43] —, "Willkommen im programmierzirkus-ein programmierkurs fuer grundschulen," *Informatische Bildung zum Verstehen und Gestalten der digitalen Welt (INFOS'17)*, pp. 327–334, 2017. [Online]. Available: https://www.uni-oldenburg.de/fileadmin/user_upload/informatik/ag/didaktik/download/infos2017/INFOS2017Proceedings-274.pdf
- [44] T. Bell, I. H. Witten, and M. R. Fellows, *Computer Science Unplugged - an enrichment and extension programme for primary-aged children*. csunplugged.org, 2002. [Online]. Available: <http://csunplugged.org/>
- [45] A. Funke and K. Geldreich, "Gender differences in scratch programs of primary school children," in *Proceedings of the 12th Workshop on Primary and Secondary Computing Education, WiPSCE 2017, Nijmegen, The Netherlands, November 08 - 10, 2017*, E. Barendsen and P. Hubwieser, Eds. ACM, 2017, pp. 57–64. [Online]. Available: <http://doi.acm.org/10.1145/3137065.3137067>
- [46] C. M. Lewis and N. Shah, "Building upon and enriching grade four mathematics standards with programming curriculum," in *Proceedings of the 43rd ACM technical symposium on Computer science education, SIGCSE 2012, Raleigh, NC, USA, February 29 - March 3, 2012*, L. A. S. King, D. R. Musicant, T. Camp, and P. T. Tymann, Eds. ACM, 2012, pp. 57–62. [Online]. Available: <http://doi.acm.org/10.1145/2157136.2157156>
- [47] L. A. Calao, J. Moreno-León, H. E. Correa, and G. Robles, "Developing mathematical thinking with scratch - an experiment with 6th grade students," in *Design for Teaching and Learning in a Networked World - 10th European Conference on Technology Enhanced Learning, EC-TEL 2015, Toledo, Spain, September 15-18, 2015, Proceedings*, ser. Lecture Notes in Computer Science, G. Conole, T. Klobucar, C. Rensing, J. Konert, and E. Lavoué, Eds., vol. 9307. Springer, 2015, pp. 17–27. [Online]. Available: https://doi.org/10.1007/978-3-319-24258-3_2
- [48] Y. Akpınar and mit Aslan, "Supporting childrens learning of probability through video game programming," *Journal of Educational Computing Research*, vol. 53, no. 2, pp. 228–259, 2015. [Online]. Available: <https://doi.org/10.1177/0735633115598492>
- [49] J. Denner, L. L. Werner, and E. Ortiz, "Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts?" *Computers & Education*, vol. 58,

- no. 1, pp. 240–249, 2012. [Online]. Available: <https://doi.org/10.1016/j.compedu.2011.08.006>
- [50] J. Moreno-León and G. Robles, “Analyze your scratch projects with dr. scratch and assess your computational thinking skills,” in *Scratch Conference*, 2015, pp. 12–15. [Online]. Available: <http://jemole.me/replication/2015scratch/InferCT.pdf>
- [51] J. Moreno-León, G. Robles, and M. Román-González, “Dr. scratch: Automatic analysis of scratch projects to assess and foster computational thinking,” *RED. Revista de Educación a Distancia*, no. 46, pp. 1–23, 2015. [Online]. Available: <http://www.um.es/ead/red/46/>
- [52] E. Aivaloglou, F. Hermans, J. Moreno-León, and G. Robles, “A dataset of scratch programs: scraped, shaped and scored,” in *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, J. M. Gonzalez-Barahona, A. Hindle, and L. Tan, Eds. IEEE Computer Society, 2017, pp. 511–514. [Online]. Available: <https://doi.org/10.1109/MSR.2017.45>