Gender Differences in Scratch Programs of Primary School Children

Alexandra Funke
Technical University of Munich
TUM School of Education
Arcisstr. 21, 80333 Munich, Germany
alexandra.funke@tum.de

Katharina Geldreich
Technical University of Munich
TUM School of Education
Arcisstr. 21, 80333 Munich, Germany
katharina.geldreich@tum.de

ABSTRACT

It is commonly believed that the attitudes towards programming are strongly dependent on gender among adolescents or adults. Regarding younger children, these differences in the attitudes might be less relevant. To investigate this, we have analyzed the learning outcomes of an introductory programming course for primary school children. The three-day course was designed specifically for 4th-grade students (ages 9 - 10) and taught four times up to now. Fifty-eight children (26 girls and 32 boys) have participated from May to August 2016. During these courses, the children have developed 127 Scratch projects. We found that boys and girls had used different block types to develop their programs. It also showed that girls and boys created very different types of programs. This paper presents the course design; the methodology of the qualitative analysis and the results in detail.

CCS CONCEPTS

•**Social and professional topics → Computer science education; Gender;** *Children;*

KEYWORDS

Computer science education; primary school; primary education; scratch; programming; gender

ACM Reference format:
Alexandra Funke and Katharina Geldreich. 2017. Gender Differences in Scratch Programs of Primary School Children. In *Proceedings of WiPSCE '17, Nijmegen, Netherlands, November 8–10, 2017,* 9 pages.
DOI: 10.1145/3137065.3137067

1 INTRODUCTION

Computer Science has to overcome several challenges. Universities and schools are confronted with different misconceptions and prejudices towards CS [11] which manifest themselves from an early age [20]. In order to prevent students from developing negative attitudes, one approach is to introduce Computer Science concepts like programming already in primary school to provide opportunities for making experiences with technology and CS. Children should learn that they can use computers not only as users but also as creators [2]. This role change combined with a fun experience of programming could increase their self-confidence towards CS in particular and technology in general [22]. At the same time, the discussion about the necessity of computer science and programming in childhood education is growing steadily [22]. While several countries have already introduced CS in their primary school curricula (e.g. the UK [7] and Australia [10]), Germany has not yet developed mandatory guidelines for how to deal with the new topics. To find out which teaching methods and content would be appropriate for German primary schools, we designed an introductory programming course for fourth graders. Although a predominant goal of the course was to change the students' attitudes towards CS, we also wanted to gain insight into how profoundly they can learn several programming concepts. Further, we want to investigate the relationship between the students' gender and the quality of the program they created. According to this, our research question was

- *Which differences can be detected between the programming results of boy and girls?*

This work is structured as follows: First, we discuss some theoretical background about gender differences in CS and related work regarding Computer Science courses for primary school students, especially programming courses. In addition, we provide a short overview of the design of our programming course. This is followed by a description of the methodology and analysis. In order to illustrate the qualitative analysis of the programming results, we present three project examples from our courses. Afterward, the results are presented with a discussion of the findings. The paper closes with a summary of the findings and an outlook for further work.

2 RELATED WORK

2.1 CS Courses for Primary Schools

In recent years, a variety of courses has been developed to expose children to computer science concepts. Some of these courses set their focus on programming. In [23] the authors implemented an in-school computer science course for 5th-grade students. They co-designed this with a primary school teacher who had prior knowledge in technology but no plain CS background. The authors offered the course as a 30-hour course during the regular school year. To analyze the effectiveness of this collaboratively developed curriculum, the researchers collected data with interviews before and after the course, videos of the working students, screen recordings and student-created material like short essays and storyboards. During the class, the students completed two programming projects with Scratch and worked almost all in pairs (18 pairs in total). They found that female-female pairs worked very well together and were creative. However, their products missed many requirements. One male-male pair had a keen interest and enthusiasm for programming. They tried out many things on their own and had a similar line of thinking. Nevertheless, the product of the boys got low scores for usability and consistency. In summary, the authors listed critical observations for future courses. One key finding from this research is the usefulness of supportive, collaborative work. Programming is also an important part of the CS courses in [9], where Duncan and Bell described an example computer science course for primary schools students. The authors implemented a CS class, which took place an hour a week during a school year. More than 600 students aged between 5 and 12 were taught during the study. The primary goals of the course were that the students engage with the presented content and enjoy the classes. Furthermore, rather than learning to use a specific programming language, the students should become familiar with the basic principles of programming. During these courses, the students worked with the programming language Scratch and had to solve two main programming quizzes at the end. After analyzing the results, the researchers found no statistically significant difference between the average achievement of the participating girls and boys. However, girls were not as good as boys at predicting their ability. Despite doing as well as the boys, they did not seem to be aware of this. Girls needed more encouragement to try out new activities while the boys often followed a more experimental approach and tried out more on their own. In summary, most of the kids enjoyed using Scratch and wanted to continue learning about CS and programming.

2.2 Gender Differences in CS

A lot of research has been conducted that showed the existence of gender differences in Computer Science (CS), for example in Europe or the United States [4]. Still, more boys choose computer science at school or as a professional field than girls. Already in school, frequently boys are performing better in CS compared to girls [17]. Several studies explained this differences by certain stereotypes that are supposed to have a negative impact on the performance of female students, see for example [19]. For example, the role model of the typical male computer scientist is still alive [26]. On the other hand, almost no female role models for girls exist, because the majority of computer scientists and computer science teachers still are male. At home girls and boys often experience mothers that are technologically still inept, which represents a substantial obstacle to female role models in computer science [18]. When choosing their major subject, students often have a diffuse perception of CS. It seems that boys decide based on their interests in computers or programming, while girls choose due to the prospects for future professional life [1]. Boys seem to have more prior knowledge in CS. Girls, in contrast, are less self-confident in this area. They tend to attribute their successes in computer science to luck and their failures to the lack of ability [26]. However, both female and male students can reach the same level of ability in computer science [8]. It turned out that the performance of CS students does not differ between men and women if learning styles are taken into account [8].

3 DESIGN OF THE COURSE

We developed a three-day course for students of the fourth grade of primary school, see also [14]. As context, we chose "Circus" for all tasks and materials, out of three reasons. First, we regarded this as an attractive field of their personal experience. Second, a circus offers a variety of interesting tasks to simulate the actions of animals or human beings. Third, we hoped to attract both girls and boys equally by this metaphor. On each day, we spent four hours with the kids. Day by day, the students are exposed to a more and more detailed picture of programming. At the end, we expected that the children had learned the basic principles of programming, in particular, to work with the programming environment as well as to apply and combine algorithmic control structures.

Day 1 Most of the students did not have any previous knowledge of programming or computer science. Therefore, the goal of the first day is to give them a basic idea of how a computer program works. They were to realize that programs execute a particular task by following precise and clear instructions. Because we did not want to overstrain the students, we decided to introduce the basic algorithmic concepts "unplugged" [3], before any programming.

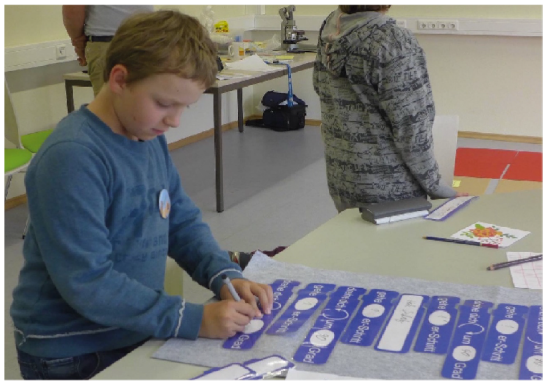


Figure 1: Programming with haptic blocks

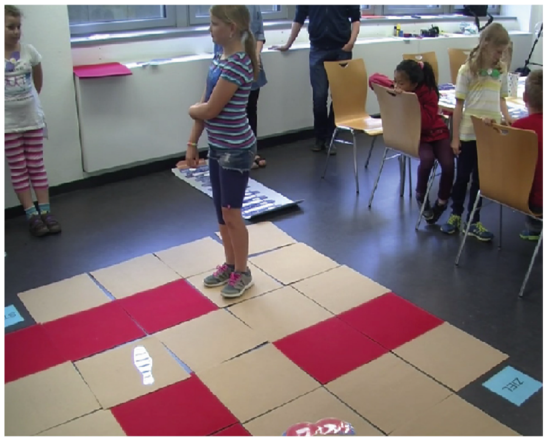


Figure 2: Simulating the execution of a program

Hence we use social activities and group problem-solving at day one, without actually working on computers. In order to learn how to split tasks into smaller parts, we provided a variety of short exercises in which different activities had to be transformed according to unambiguous instructions. Afterward, the groups had to work together to solve a more complex task. To take up the circus theme, we let the students program each other, solving tasks like searching for missing items or animals in a circus tent. To represent the solutions, we use haptic (printed and shrink-wrapped) Scratch-like programming blocks to prepare "real" computer programming on day two (Figure 1 and 2).

Day 2 The goal of the second day is to enable the students to create simple Scratch programs that produce Multimedia output. To provide a child-friendly programming environment and to spare the students any unnecessary syntactical overhead, we decided to use the block-based language Scratch [16]. We created a learning circle with increasingly difficult stations, which introduces the core elements of Scratch one by one, leading from simple sequences to control structures like loops and conditional statements. In each part of the circle, the students have to solve tasks that are presented on handed-out instruction sheets. To support the students' expected variety in knowledge and learning pace, we prepared additional tasks as well as helpful tips.

Day 3 On the third day, we wanted to find out what the students had learned and if they could solve more open tasks. In addition, we wanted to stimulate the children to work creatively and in a self-directed fashion. To compare the outcomes, we set the following "mandatory" requirements for the students' projects. The programs should a) work on more than one sprite b) move the sprites during execution c) comprise at least one iteration and d) include at least one conditional statement. After meeting these requirements, the students should continue their programming work without any further guidelines. They were free to experiment with Scratch, to invent their circus stories and implement these. At the end of day three, all programs were presented in front of the course, and the students had the opportunity to comment their project.

Table 1: Subcategories for *A. Requirements*

category/code	description
Several Sprites	Counting all sprites to figure out if more than one sprite is used.
Sprite Motion	Sprites should move during program execution. For this, all blocks of the type Motion were counted to match them with this category.
Iteration	An iteration is included, when the students used *forever* or *repeat* blocks.
Conditional Statement	If a program contains at least one *if _ then* or *if _ then else* block to check for conditions, we code this as a conditional statement.

Table 2: Subcategories for *B. Programming Concepts*

category/code	description
Sequence	A correct sequence is present when the program runs in a systematic order. For example, some students did not pay attention to how the program would be executed. When they created a conversation, sometimes all scripts ran at the same time by mistake.
Variables	Variables can be created within Scratch and then be used within programs. To code this category, the variable blocks of *Data* are counted and matched.
Lists	Similar to Variables, the list blocks of *Data* are counted and matched with this category.
Event Handling	Responding to events triggered by either the user or another script. All blocks of the type *Event* were counted to match them with this category.
Threads	If two or more scripts were going to execute at the same time, we coded this as existent threads.
Coordination and Synchronization	Checking the program for all *wait _ secs*, *wait until*, *when I receive*, *broadcast* and *broadcast _ and wait blocks*, for coordinating the actions of multiple sprites.
Keyboard Input	Includes the program an *ask _ and wait block*, and it provides a keyboard input for users.
Random Numbers	A random number exists when a *pick random to* block could be found in the project.
Boolean Logic	*And*, *or* and *not* were coded as boolean logic.
Dynamic Interaction	The usage of *mouse x*, *mouse y* or *loudness* is used for dynamic interaction.

Table 3: Subcategories for *C. Code Organization*

category/code	description
Extraneous Blocks	A block is extraneous if it has no connection to a script with a starting event.
Sprite Names	This category codes whether the original sprite name is overridden or not.
Variable Names	This category codes whether the name of a variable is meaningful (e.g. fitimerfi) or not.

4 STUDY

4.1 Methodology

The goal of the study is to investigate the relationship between the students' gender and the characteristics of the program they created. For this, our methodological approach consists of two steps: data collection during the courses and the data analysis, consisting of a qualitative analysis of the records, and especially in this case the children's programming results.

All sessions were entirely videotaped to analyze the interactions of the students. At the beginning and end of each course day, we hold group interviews with the classes. To get an idea of the students' prior knowledge, as well as their mental image of programming, we used a variety of interviewing and reflection methods. Further, we captured the screens of the student computers during the programming exercises on days two and three. This is important to get an image of the students' working methods. To collect the Scratch programs of the students, we saved the projects after the course days. One goal of the study is to examine the quality of the programs the children created. Further, we want to find which programming concepts they use and in which way they do so.

For rating the projects' quality, we developed a category system. We distinguish between four main categories: *A. Requirements* (Table 1), *B. Programming Concepts* (Table 2), *C. Code Organization* (Table 3) and *D. Operability* (Table 4). Finally, we analyzed the projects for the *Level of Understanding* (Table 5) which based on the SOLO taxonomy [5]. The code systems based on the analysis of existing category systems. The development of them is described in detail in [13].

4.2 Application

All courses were taught by a female trained primary school teacher who is also a researcher in CS education. Additionally, a male computer scientist assisted during the courses. In May 2016, a pilot study was carried out at a student research center, which is led by our University. After following improvements of the tasks, we conducted the course with classes of 4th grade in June and July. They took place in rooms of our faculty, where we were able to provide a constant setting and stable technical equipment. A further course followed in August at the student research center.

Table 4: Subcategories for *D. Operability*

category/code	description
Functionality	Whether or not the project performs correctly when it is started is coded as functionality.
Sprite Customization	A sprite is customized, if a predefined sprite is adjusted (e.g. removing an arm) or if the student created their own sprite (e.g. drawing it).
Stage Customization	A stage is customized, if a predefined stage is adjusted (e.g. add a drawing) or if the student created their own sprite (e.g. drawing it).
Interactivity	A highly interactive program provides the user with opportunities to interact with it (e.g. key control). In contrast, a program with no interactivity is, for example, a sequence of actions.
Usability	The project is intuitive, if the user understands how the program runs with little or no information. Sometimes programs use unorthodox keys for controlling the program, like fiGfi. This is only partly intuitive for users.
Project Type	We coded the project types *Animation*, *Game*, *Interactive Art*, *Music and Dance*, *Story* and *Video Sensing* according to the project designation on the Scratch web page . For example, a *Story* is a sequence of actions without any user input or control possibilities.

Further, we want to find which programming concepts they use and in which way. In our case, the sample size is 127 programs. 74 programs of 26 girls and 53 programs of 28 boys. Two boys deleted their programs before we could save them and two other boys worked together in a team. For this reason, we were only able to use the projects of 28 male students. For the rating of the projects' quality, we first developed a category system. For this purpose, we analyzed already existing category systems [23] [21] [25] and adjusted them based on our course material and given requirements for the students.

With using these category systems, two researchers rated 27 projects (21 %) of the primary school students of our courses to assess intercoder agreement and reliability. Both raters did not know which student created which project and even not the gender of the student. To get an idea of the accord, the coefficient of Brennan [6] and the raw agreement were calculated, which resulted in an almost

Table 5: Categories for *Level of Understanding*

category/code	description
Prestructural (L1)	The script contains only a few blocks. The student does not understand how to extend the script to a meaningful program.
Unistructural (L2)	The script contains sequences of actions in a simple way. Control structures are not contained or they are unrelated.
Multistructural (L3)	The script fulfills all given requirements and includes a variety of different block types. The code may have been reorganized to make a more integrated solution.
Relational (L4)	The script provides a well-structured program that removes all redundancy and has a clear, logical structure.
Extended Abstract (L5)	The script uses concepts and blocks beyond those required in the exercise to provide an improved solution.

perfect agreement according to Landis [15]. The common factors of Cohen and Krippendorff are not applicable, as they require a normal distribution of the codes [24]. After the coding process, we clustered the projects by the courses and gender. For each group, we studied the results and differences between them, which are presented in Section 5. Furthermore, we tried to find connections between individual categories.

4.3 Example Projects

4.3.1 A Girl's Project. Figure 3 shows a story about a circus manager, wizard, bear and muscle man with twelve sprites. Because the theme of the courses was *Programming Circus*, this is a typical storyline in the projects. The student used eleven different blocks, a total of 50 blocks. As shown in the code of Fig. 4, the most commonly used block type was *Look* (28 blocks). Thirteen of these blocks were *say _ for _ secs*. In order to start the scripts she used *when green flag clicked* (3), and *when I received* (3). Each sprite was assigned exactly one *Event* block. To structure the conversion of the story, the student used 10 *wait _ secs* and three messages.

4.3.2 A Boy's Project. An example program of a boy is shown in Figures 5 and 6. A total of 36 blocks were used (14 different blocks). Out of them, 12 belong to *Motion* (six times *move _ steps*). Even though he only used two sprites to develop his small game, the program reacted to nine *Events* (seven times *when _ key pressed*). With two *if _ then* blocks, the student included conditional statements. In addition, three iterations are included. The user can activate a new game round by pressing the key "q". When the sprite touches the turquoise lines, the user loses the game. Both events broadcast a message that causes that the sprite to move to the start.

5 RESULTS

Overall the students used 3,200 blocks in their Scratch projects. During the second course day, we introduced 28 different blocks to the students. Three-fourths of the used blocks were part of the introduced blocks. Boys used 70 different blocks, whereas girls used 40 different blocks.

When looking at the types of blocks, which were used, we observe that male students use twice as often *Motion* blocks. Female students instead use twice as often blocks out of the *Look* section. The usage of *Control* and *Event* blocks are nearly the same for both groups. With 10 % of usage, the most used block of boys is *move _ steps*. However, as one can see in (Table 7, the second (*when _ key pressed*) and third (*when green flag clicked*) most used blocks of them have nearly the same percentage. In the programs of girls are three blocks, which stand out: *wait _ secs*, *when green flag clicked* and *say _ for _ secs* (Table 6). Boys chose more often the *when _ key pressed*-block as a start of their script.

Overall, girls use a mean of 23 blocks (median 16) and six sprites (median 5). Boys instead used 32 blocks on average (median 18) and five sprites (median 3). Important programming concepts are conditional statements and iterations. During the third course day, we gave the instruction that the programs should contain at least one of each concept. 61 % of the girls and 64 % of the boys reached the requirement to use at least one conditional statement. However, iterations were included in the projects of 84 % of girls and 78 % of boys.

The developed programs of the students differ in their complexity. Almost all students arranged the blocks in their projects in a systematic order (90 %) and used at least one event (93 %). 102 projects (80 %) include more than two Event blocks. Parallel execution of two independent scripts is supported by 16 % of all projects. Moreover, even 63 % support the parallel launching of more than two scripts. Some subcategories of programming concepts were not included in any project.

Neither girls nor boys have overridden the names of the sprites. Further, only four projects included variables. Only one of them was made by a girl. With 5 % of all produced programs by girls, only a small number include extraneous blocks. For boys, this number is with 21 % much higher.

80 % of the projects were categorized as being completely functional. Only 11 programs (9 of girls) have no function at all. In the most projects, there exist no sprite customization (108 projects, 85 %) and stage customization (112 projects, 88 %). These results are balanced between boys and girls. Nearly two-thirds of the projects do not offer any interactive elements to the user, such as user input or keyboard control to interact with the program. Most of the projects which contain keyboard control were made by boys. As partly or entirely intuitive were categorized over 80 % of the projects, only 23 projects are not intuitive at all. The most often project type which appears is Story with nearly 45 % of all programs. It is followed by Animation (32 %) and Game (16 %). An important observation is that the girls produced much more Story-programs (79 % of these

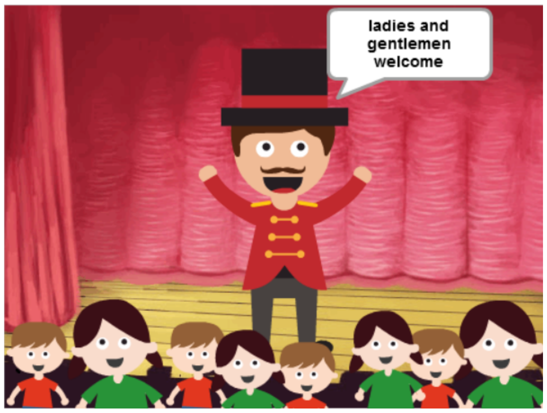
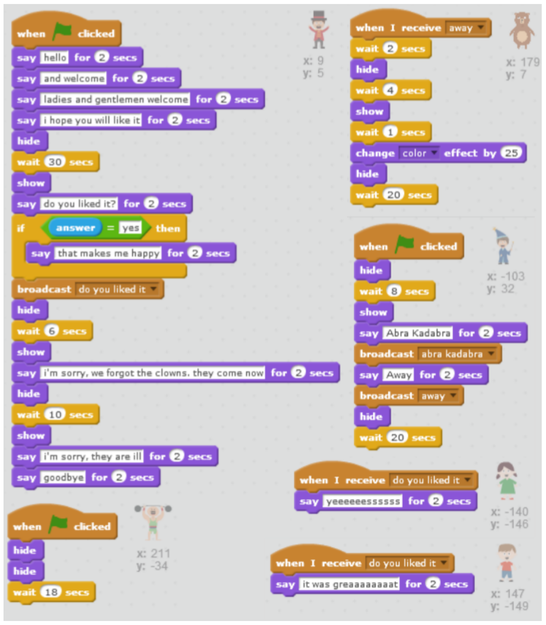


Figure 3: Stage in the girls' project



```
when [flag] clicked
say [hello] for (2) secs
say [and welcome] for (2) secs
say [ladies and gentlemen welcome] for (2) secs
say [i hope you will like it] for (2) secs
hide
wait (30) secs
show
say [do you liked it?] for (2) secs
if <(answer) = [yes]> then
    say [that makes me happy] for (2) secs
broadcast [do you liked it v]
hide
wait (6) secs
show
say [i'm sorry, we forgot the clowns. they come now] for (2) secs
hide
wait (10) secs
show
say [i'm sorry, they are ill] for (2) secs
say [goodbye] for (2) secs

when [flag] clicked
hide
hide
wait (18) secs

when I receive [away v]
wait (2) secs
hide
wait (4) secs
show
wait (1) secs
change [color v] effect by (25)
hide
wait (20) secs

when [flag] clicked
hide
wait (8) secs
show
say [Abra Kadabra] for (2) secs
broadcast [abra kadabra v]
say [Away] for (2) secs
broadcast [away v]
hide
wait (20) secs

when I receive [do you liked it v]
say [yeeeeeessssss] for (2) secs

when I receive [do you liked it v]
say [it was greaaaaaaaat] for (2) secs
```

Figure 4: Code for the girls' project

were made by girls). Against this, male students developed more programs which were categorized as Game (90 % of these were made by boys). Animation programs were made 50 % of boys and 50 % of girls.

44 % of the students reached only the first level of the *Levels of Understanding* (see [13]) *Prestructural*. It means the students do not understand how to animate the sprites. In some projects, there are no blocks used at all. Girls made nearly 80 % of the programs at this level. Level 2 *Unistructural* was reached by one-fourth of the Scratch projects. First animations and logic are observable in them but on a very simple step. These programs were made 50 % of boys and 50 % of girls. This distribution is similar to the next level *Multistructural*. The highest level of understanding *Extended Abstract* was reached only by six projects, which were made by boys.

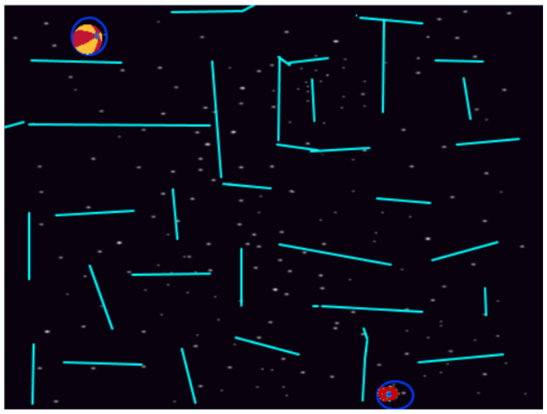
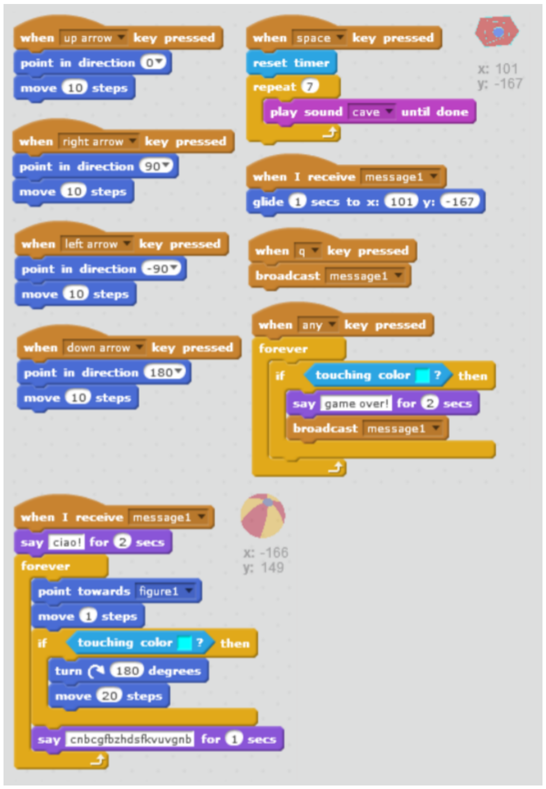


Figure 5: Stage in the boys' project



```
when [up arrow v] key pressed
point in direction (0 v)
move (10) steps

when [right arrow v] key pressed
point in direction (90 v)
move (10) steps

when [left arrow v] key pressed
point in direction (-90 v)
move (10) steps

when [down arrow v] key pressed
point in direction (180 v)
move (10) steps

when [space v] key pressed
reset timer
repeat (7)
    play sound [cave v] until done

when I receive [message1 v]
glide (1) secs to x: (101) y: (-167)

when [q v] key pressed
broadcast [message1 v]

when [any v] key pressed
forever
    if <touching color [ ] ?> then
        say [game over!] for (2) secs
        broadcast [message1 v]

when I receive [message1 v]
say [ciao!] for (2) secs
forever
    point towards [figure1 v]
    move (1) steps
    if <touching color [ ] ?> then
        turn ↻ (180) degrees
        move (20) steps
    say [cnbcgfbzhdsfkvuvgnb] for (1) secs
```

Figure 6: Code for the boys' project

6 DISCUSSION

We selected the projects presented in Section 4.3 based on the most frequently created project types for each group. Program 1 is a Story project of a girl. Indications for this are the typical distribution of the blocks used: the most frequently used type is *Look* (especially *say for _ secs*), followed by *Control* (especially *wait for _ secs*). This usage of blocks is not only typical for girls of our courses (see Table (Table 7), but even for the distribution within a *Story*. Furthermore, this program executes as a fixed flow

Table 6: Boys' Top Ten Blocks

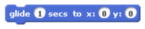
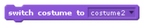
Block	Image	Type	Portion
move _ steps		Motion	10.8 %
when _ key pressed		Events	9.1 %
when green flag clicked		Events	9.0 %
wait _ secs		Control	7.7 %
forever		Control	7.2 %
if _ then		Control	5.3 %
point in direction		Motion	4.2 %
say _ for _ secs		Look	4.1 %
repeat		Control	4.1 %
glide _ secs to x: y:		Motion	3.2 %

Table 7: Girls' Top Ten Blocks

Block	Image	Type	Portion
wait _ secs		Control	13.5 %
when green flag clicked		Events	12.1 %
say _ for _ secs		Look	11.5 %
move _ steps		Motion	6.3 %
repeat		Control	3.8 %
hide		Look	3.7 %
show		Look	3.6 %
play sound _		Sound	2.7 %
switch costume to		Look	2.6 %
rest for _ beats		Sound	2.4 %

of a conversation. The movement of elements is minimal. The student has accurately timed, when each sprite speaks. There is no possibility for the user to interact with the program with e.g. key control. The figures bear, girl and boy are activated with messages within the project. The program is missing a conditional statement, which was a given instruction.

In project 2, the boy used quite a few blocks very well, and the mixture of the different blocks was balanced. One of the indications that this project belongs to the type *Game* is that a game uses fewer sprites as well as fewer blocks compared to other project types. Both characterizations are included in this case. A game, like this one, is mostly controlled by pressing keys and broadcasting messages. There are almost no *Look* blocks and many *Motion* blocks. Although there are only two sprites, the program includes nine event blocks. By using the arrow keys to control the game, it is clear to the user what to do.

As mentioned in section 5, boys used 70 different blocks, whereas girls used only 40 different blocks. Boys are often described as more willing to experiment [12]. That could be another indication for this and should be the focus of following investigations. Further, we found that the most used block types, as well as concrete blocks, differ between the groups. Girls use more often *Look* blocks, boys more often *Motion* blocks. Also, the three most used blocks of girls (*wait _ secs*, *when green flag clicked*, *say _ for _ secs*) might be an indication for creating a program of the type *Story*, because one characterization of *Stories* is a very high number of *Look* blocks [13]. The scripts in this project types start almost every time with a *when green flag clicked*-block. On the other hand, the high usage of *when _ key pressed* and the *Motion* blocks of boys might indicate that they created a *Game* project.

There are many possible reasons for such differences. Firstly, the schools and teachers play a significant role in the development of their students. For example, the students of one class course had a man as the class teacher. In contrast, a woman usually teaches the other class course. Furthermore, the students of three courses lived in a rural area and the others in an urban area. Other reasons could be the educational background of the parents and the migration background of the students. Of all 18 students in one class course, only one girl had no migration background. In addition, we as course teachers influence the students.

7 CONCLUSION

Overall, the results of our pilot courses demonstrated that at least every second child, boys as well as girls, in grade 4 is able to learn basic programming with Scratch in three days. However, there are some differences between girls and boys in the working methods and the used blocks/concepts.

Due to the low number of students, this result might still be caused by some specific luck circumstances of our courses. A lot of work will be needed to provide convincing arguments that all children in primary school are able to learn to program in several days. Of course, we will repeat our programming course with many more classes in the following years. To refine the evaluation, our next step will be to analyze the screen captures we recorded during the courses of all screens in order to find out in which order the students developed their programs. Furthermore, we will try to find connections between the video analysis and the analysis of the programming results.

ACKNOWLEDGMENTS

We thank Johannes Krugel and Ramona Olwitz for supporting us during the course days.

REFERENCES

[1] M. K. Ahuja. Women in the information technology profession: a literature review, synthesis and research agenda. *European Journal of Information Systems*, 11(1):20–34, 2002.

[2] M. Armoni and J. Gal-Ezer. Early computing education. *ACM Inroads*, 5(4):54–59, 2014.

[3] T. Bell, I. H. Witten, and M. Fellows. *CS Unplugged: An enrichment and extension programme for primary-aged students.* 3 edition, 2015.

[4] S. Beyer, K. Rynes, J. Perrault, K. Hay, and S. Haller. Gender Differences in Computer Science Students. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '03, pages 49–53, New York, NY, USA, 2003. ACM.

[5] J. B. Biggs and K. F. Collis. *Evaluating the quality of learning: the SOLO taxonomy (structure of the observed learning outcome).* Academic Press, 1982.

[6] R. L. Brennan and D. J. Prediger. Coefficient Kappa: Some Uses, Misuses, and Alternatives. *Educational and Psychological Measurement*, 41(3):687–699, 1981.

[7] N. C. C. Brown, S. Sentance, T. Crick, and S. Humphreys. Restart: the resurgence of computer science in UK schools. *ACM Transactions on Computing Education*, 14(2):1–22, 2014.

[8] P. Byrne and G. Lyons. The effect of student attributes on success in programming. *ACM SIGCSE Bulletin*, 33(3):49–52, 2001.

[9] C. Duncan and T. Bell. A Pilot Computer Science and Programming Course for Primary School Students. In *the Workshop in Primary and Secondary Computing Education*, pages 39–48, 2016.

[10] K. Falkner, R. Vivian, and N. Falkner. The Australian Digital Technologies Curriculum: Challenge and Opportunity. In *Proceedings of the Sixteenth Australasian Computing Education Conference - Volume 148*, ACE '14, pages 3–12, Darlinghurst, Australia, Australia, 2014. Australian Computer Society, Inc.

[11] A. Funke, M. Berges, and P. Hubwieser. Different Perceptions of Computer Science. In *2016 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*, pages 14–18. IEEE, 2016.

[12] A. Funke, M. Berges, A. Mühling, and P. Hubwieser. Gender Differences in Programming: Research Results and Teachers' Perception. In *the 15th Koli Calling Conference on Computing Education Research*, pages 161–162. IEEE, 2015.

[13] A. Funke, K. Geldreich, and P. Hubwieser. Analysis of Scratch Projects of an Introductory Programming Course for Primary School Students. In *Proceedings of the 2017 IEEE Global Engineering Education Conference (EDUCON)*, pages xx – xx. IEEE, 2017.

[14] K. Geldreich, A. Funke, and P. Hubwieser. A Programming Circus for Primary Schools. In *Proceedings of the 9th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, pages 46–47. 2016.

[15] J. R. Landis and G. G. Koch. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33(1):159–174, 1977.

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

[17] J. Margolis and A. Fisher. *Unlocking the Clubhouse: Women in Computing.* 2002.

[18] P. Moorman and E. Johnson. Still A Stranger Here: Attitudes Among Secondary School Students Towards Computer Science. *ACM SIGCSE Bulletin*, 35(3):193, 2003.

[19] B. Nelson. The data on diversity. *Communications of the ACM*, 57(11):86–95, 2014.

[20] K. Prottsman. Computer science for the elementary classroom. *ACM Inroads*, 5(4):60–63, 2014.

[21] L. Seiter. Using solo to classify the programming responses of primary grade students. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 540–545, New York, NY, USA, 2015. ACM.

[22] H. Topi. Gender imbalance in computing. *ACM Inroads*, 6(4):22–23, 2015.

[23] J. Tsan, K. E. Boyer, and C. F. Lynch. How Early Does the CS Gender Gap Emerge? In *the 47th ACM Technical Symposium*, pages 388–393, 2016.

[24] A. von Eye. An Alternative to Cohen's . *European Psychologist*, 11(1):12–24, 2006.

[25] A. Wilson, T. Hainey, and T. Connolly. Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. *International Journal of Games-based Learning*, (3):93–109, 2013.

[26] B. C. Wilson. A Study of Factors Promoting Success in Computer Science Including Gender Differences. *Computer Science Education*, 12(1-2):141–164, 2010.