Designing and comparing two Scratch-based teaching approaches for students aged 10-12 years in Dutch primary education

Master thesis Science Education and Communication 2016/2017

30 ECTS

Written by:	Supervisors:
Nienke van Es	prof. dr. J. Jeuring
3852857	drs. P. Bergervoet

January 24, 2017

This paper is written for computer science education scientists and teachers who are interested in the subject. This paper could fit in, but is not especially written for, a journal like Computers & Education.

Abstract

Programming and computational thinking are becoming more important in primary education, but in research there is little information about effective ways to teach students programming. Also, many teachers do not know how to teach this subject. We designed two teaching approaches to teach students on elementary schools how to program. One approach followed the instructionistic 4C/ID model, the other approach followed constructionism. The learning gains of these two approaches were compared using a pre- and posttest. This provided more insight into the difference between learning gains of these two approaches. In total, 129 students from two different schools participated. A significant difference ($p = .037$, $d = .59$) between the two approaches was found on one of the schools. This difference was in favour of the 4C/ID approach. On the other school and for the total group no significant difference was found. This can be explained by the fact that the students on the different schools have different backgrounds. However, there are a few openings for future research.

For young students, learning how to develop a computer program has more advantages than just gaining the ability to program. The process of learning to program also improves problem solving, logical thinking, and organizational skills (Lee, 2011). These skills are also applicable to real world problems and other courses. With the upcoming "Ons Onderwijs2032", programming and computational thinking are becoming more important in primary education in the Netherlands. Also, the Dutch secretary of education was assigned by the parliament to search for a place for programming in primary education. This interest in programming is not only in the Netherlands. Countries all over Europe started or developed plans to integrate programming in the curriculum for primary schools since 2014 (European Schoolnet, 2014).

A problem with this change in curricula is the lack of information on how to teach programming effectively. There is little quantitative research done to the learning gains of different programming teaching approaches (Jeuring, Corbalan, van Es, Leeuwestein, & van Montfort, 2016). Because of the lack of research, many teachers also do not know how to teach this subject. Teachers are still puzzling and experimenting with programming education. They are possibly unaware of the harm they can cause while doing this. Duncan, Bell, and Tanimoto (2014) showed that confused teachers can harm young students' attitude towards programming.

Therefore, it is important to provide teachers with successful teaching approaches to teach young students programming. This can increase the satisfaction about the learning outcomes of both schools and teachers, while also solving the problem of harming young students' attitudes towards programming. With this research we like to contribute to this problem by designing and comparing two different teaching approaches.

A widely used approach for teaching programming is based on constructionism (Baytack, & Land, 2011; Bruckman, & De Bonte, 1997; Dasgupta et al., 2016; Ngai, Chan, Leong, & Ng, 2013) which makes it interesting to see the effect of this kind of approach. The 4C/ID model is designed to teach complex cognitive skills. Complex cognitive skills are described as skills that are time consuming to acquire (Van Merriënboer, & Dijkstra, 1997). As an example of a complex cognitive skill, Van Merriënboer and Dijkstra (1997) mention computer programming. This model can be used in combination with a design model to create programming lessons. For the above mentioned reasons, we based the two teaching approaches on constructionism and the 4C/ID model.

The first teaching approach is based on constructionism. With this ap-

proach students build their own meaningful product. The other approach is based on an instruction model, namely the 4C/ID model. With this approach, the teacher explains a topic with a frontal instruction. After this, the students will practice these topics with worksheets. The approaches will be further explained in section 1.

The tool we used to teach students programming is Scratch. This tool is often used to study teaching approaches and to teach students programming (Aritajati, Rosson, Pena, Cinque, & Segura, 2015; Dasgupta, Hale, Monroy-Hernández & Hill, 2016; Flannery et al., 2013; Franklin et al., 2015). According to the Scratch website, the tool is used in more than 150 countries and has more than nineteen million shared projects (http://scratch.mit.edu). The reason we chose this tool is because it is suitable for young students. Scratch is designed for novice programmers without any programming experience (Aritajati et al., 2015; Lee, 2011). Also, the interface of Scratch is student-friendly (Franklin et al., 2015).

Scratch has a drag and drop interface which provides visual support using programming blocks. These blocks only fit when the code is syntactically correct. Young programmers can use the blocks to write their own software or games without writing the textual code (Lee, 2011). An example of Scratch code is shown in Figure 1.

The learning gains of the two teaching approaches are compared to get more insight in how these approaches affect the development of programming competencies of students. Lessons for each teaching approach are performed on elementary schools. The schools had students aged 10 to 12 years. On each school both teaching approaches were performed for different classes to teach the students program in Scratch. The aim of this comparative design-based research is to determine which of the teaching approaches resulted in

Figure 1: An example of Scratch code: the cat asks for your name, says something with your name, and then walks in the shape of a square.

higher learning gains.

1 Background

For this study, two teaching approaches are used. In this section, the learning theories and practical implications of each approach are discussed. We will explain how the two teaching approaches, as we developed them, are similar and different from each other in the next section. Please note that the outcomes of this research not only depend on theories, but also on how well we managed to use the characteristics of the theories in the development of the approaches (DiSessa, & Cobb, 2004).

1.1 Constructionism

Constructionism is a learning theory which builds on Piaget's constructivism. It shares the idea of the learner being an active constructor of knowledge. The constructionism theory emphasizes that this knowledge is built when the learner is engaged in making a product in a development

cycle which is shareable, for example building a computer program (Kafai & Resnick, 1996, pp. 176-178). The products that are built by the students have to be meaningful as well, because constructing personally meaningful products has proven to be effective. The similarities with the project based approach enables the teacher to act as a coach instead of a facilitator of knowledge (Kafai & Resnick, 1996, pp. 176-178, 208-214).

Papert and Harel (1991) described constructionism as "learning-by-making". They also connect the word "playful" to constructionism. This gives a summary of what constructionism in practice should be. To be more specific, Kafai and Resnick (1996) describe two relevant characteristics of constructionsm in their book: the first is learning through design, and the second one is learning in communities.

Ngai and colleagues described a characteristic of constructionism as a way to incrementally enlarge and build knowledge into fundemental concepts (Ngai et al., 2013). In addition to this, Bruckman and De Bonte (1997) argue that the process of learning is successful when the students are working on projects which are challenging their skills and give opportunities for creativity. In other words, learning through designing. Another important aspect described by Bruckman and De Bonte (1997) is the community where students can help each other. Within the learning process, the students can either ask their teacher or fellow students for help and construct knowledge together. This is the learning in communities as described by Kafai and Resnick (1996). An example of using Scratch with the constructionist approach is to let students design and program their own games.

1.2 The 4C/ID model

The four component instruction model, 4C/ID in short, is designed as a guideline for instructional design. The model consists of four components which, when combined, support complex learning. The four components are learning tasks, supportive information, just in time information, and subtask practice. Whereas other design models follow the theory that a complex learning task can be achieved by the sum of the parts, this does not hold for the 4C/ID model. The 4C/ID model is based on the theory that the whole is more than the sum of the parts. There is evidence that the theory where the whole is equal to the sum of the parts does not work (Van Merriënboer, Clark & De Croock, 2002).

One of the major theoretical foundations of this model is that complex skills can only be learned by doing. The learner must be confronted with exercises where he can perform the skills which have to be learned (Van Merriënboer & Dijkstra, 1997). With learning by doing, a student will learn procedural knowledge. According to Van Merriënboer and Dijkstra (1997), the 4C/ID model makes a distinction between controlled and automatic information processing for procedural knowledge. Controlled information processing requires effort from the learner, whereas automatic information processing requires hardly any effort because it just happens. It is therefore important to determine the desired behaviour of the learner. With the knowledge of what behaviour can be expected, a test can determine whether a learner managed to learn the skill. More importantly, the teacher can determine if the skill is automated or controlled (Van Merriënboer & Dijkstra, 1997).

To achieve automated processing, rule automation is important. Rule automation is the construction of specific procedures which can be used to

decompose problems into subproblems. The procedures can then be used to solve these subproblems. For effective controlled processing, schema acquisition is important. A schema provides knowledge that can be used to solve particular problems. If a schema is more developed it can provide more generalized knowledge. Rule automation and schema acquisition both occur when learning a cognitive complex skill. With this knowledge, the four components of the 4C/ID model are developed (Van Merriënboer & Dijkstra, 1997).

Learning tasks, supportive information, just in time information, and subtask practice are the four components of the 4C/ID model. Together they provide a basis that can be used in different contexts. The first component, learning tasks, is based on the principle that learners learn by doing meaningful tasks. Knowledge and subskills learned by doing these tasks will be integrated in the existing knowledge of the learner. None of the subskills will be practiced without the context of the whole complex task (Van Merriënboer et al., 2002; Poortman & Sloep, 2006). To get the knowledge needed for the learning tasks, the second component, supportive information, is needed. The just in time information is the procedural knowledge which is needed to perform the task. This information is given just before the learner needs it. The last component is subtask practice, which means that a subtask is practiced as long as it is not automated (Van Merriënboer et al., 2002; Poortman & Sloep, 2006).

The learning tasks for programming in Scratch could be to learn how the avatar can ask your name and then greets you with your name. The supportive information for this task will be about variables. The just in time or procedural knowledge for this task is how the students create an avatar and use the blocks and variables in Scratch. Some subtasks to practice

are making variables and changing or combining them with the help of the blocks.

1.3 Research question

In previous research, Baytak, and Land (2011) used the constructionistic theory to develop a Scratch-based teaching approach for 5th grade students. For this approach, students were asked to design an educational game for 2nd graders in Scratch. The students developed the games in 21 sessions of 45 minutes. With interviews, transcriptions, and the collected games, some conclusions were drawn. The students were motivated to make a game, and all ten students succeeded in making a game. However, a limitation of the study is the lack of pre- and post assessments to identify gains in programming skills (Baytak & Land, 2011).

Kalelioglu and Gülbahar (2014) used the opposite strategy by teaching students programming with Scratch by using frontal instructions. They started simple and made the exercises more complex every week. In the end, five lessons of one hour each were given. The aim of the research was to find a difference in the students' perceptions of problem solving skills and not testing the learning gains of the approach. The main conclusion was that they did not find any significant difference in the students' perceptions of their own problem solving skills (Kalelioglu & Gülbahar, 2014).

None of the above studies aimed to identify gains in programming competencies. However, the teaching approaches were suitable to test this with the right assessment. Therefore, we designed one lesson set based on the constructionistic approach of Baytak and Land's (2011) study. The other set of lessons is designed using the 4C/ID model. The aim of the research is to find out which approach leads to the most gain in programming com-

petencies of students: a constructionistic approach or the 4C/ID approach. The main research question is:

> *Which teaching approach, constructionism or 4C/ID, results in higher learning gains in terms of programming competencies for students aged 10-12 years?*

1.3.1 Hypothesis

Sawyer (2005) argued that the instructionistic approach mostly teaches facts and procedures to students. In todays society students need more than just facts and procedures, namely a deeper understanding of complex concepts. Students also need the skills to use this understanding to create new ideas, knowledge, products, or procedures. Instructionism results in knowledge which is hard to use outside the classroom, because the student has to know how the learned knowledge is applicable in new situations (Sawyer, 2005, pp.1-3).

On the other hand, Kirschner, Sweller, and Clark (2006) argued that a lot of guidance and instruction is more effective than constructivistic approaches. They also argue that less guidance can cause misconceptions. The arguments are based on the knowledge of the human cognitive architecture. In this architecture the main components are the working memory, the long term memory, and the relation between these two kinds of memory. To say that a students effectively learned something, there has to be a change in the long term memory. The working memory has two characteristics when new knowledge is provided, namely the capacity and the duration of how long the memory can hold the information. These limitations will be lifted when the memory can work with familiar knowledge. Minimal guiding approaches, like problem based learning, are causing an overloading of the

working memory and therefore less will be learned.

Brunstein, Betts, and Anderson (2009) do not agree with the above arguments. They stated that there is a difference between no or very little guidance and helping students by giving hints and putting them on the right track when they are making their products. When there is guidance during the learning process, a minimal guiding approach can be effective.

There is no clear answer to the question which approach will result in higher learning gains. Therefore, our hypothesis is that there will be no significant difference in learning gains.

2 Design

Five lessons were developed for each teaching approach. For the 4C/ID model, we used the ten steps to complex learning as a guideline to develop the lessons. These ten steps are based on the 4C/ID model and the Instructional Systems Design (ISD) process (Van Merriënboer, & Kirschner, 2012). In each lesson, one of the main programming themes, abstracted from the Dutch computer science curriculum (Barendsen, & Tolboom, 2016), is covered. The main themes are sequences, conditionals (if-statements), loops, and combined conditions and variables.

At the start of a class a short introduction in the theme was given by working through case studies. This explanatory inductive strategy is chosen because this strategy works well for students with little prior knowledge and is time effective (Van Merriënboer et al., 2002). After the introduction, the students worked individually on an assignment that was provided with hand-outs. The other components of the 4C/ID model are incorporated in the hand-outs. The just-in-time information is represented in red outlined squares. These squares give step by step instructions on how to deal with

practical things in Scratch. The part time practice assignments are represented in purple outlined squares. These squares contain small exercises to get more familiar with important subskills.

For the constructionist approach, the students made their own game. To make sure the students had some prior knowledge before making a game, they were taught the basics of Scratch and programming during the first lesson. The students explored the programming blocks by using the Scratch cards from the Scratch website, which encourage the students to learn by exploring (Wilson, Hainey, & Connolly, 2012). After this first lesson, the students started with making their own game. They had three and a half hour to complete their game. The last half hour was to play the other games and ask questions about them. The main sources for programming knowledge were the Scratch cards and the help they could request from the researcher.

While guiding the students, the researcher tried to stimulate cooperation between classmates by passing questions to students who asked the same thing. This kind of cooperation is strongly suggested for constructionism (Sawyer, 2005, pp. 39-40; Baytak, & Land, 2011). When helping the students, it was important to tell the students just enough so they could go on by themselves. It was not the intention to program the game for the students (Blaho, & Salanci, 2011). For students who did not know how to start, a design form was handed out. This form contained some questions to guide the students in the process of designing a game. For example, the students were asked to make a sketch of the layout of the game, and to write down some rules.

The main differences between the two approaches are summarized in Table 1. However, there are also some similarities between the approaches

4C/ID	**Constructism**
Frontal instruction at the start of each class about the main theme	No frontal instruction, except for explaining the assignment
Information on how to use Scratch and how to program	Students explore Scratch and programming by themselves with a little help from the Scratch cards
Small assignments to help with the bigger assignment	One assignment to learn all concepts by themselves
Assignments to make one concept or theme clear	

Table 1: Differences between the two teaching approaches

as well. Students were working on real world tasks with both approaches. These tasks or games were something a real programmer could also make. There was also some guidance at the start of the game design assignment for students who needed this. The students who needed help at the start were guided by using a top-down approach, while the other students had also the option to work bottom-up.

To test if there is a learning gain after performing the five lessons, we developed a pre- and posttest. The two tests are covering the three main concepts we test, namely sequences, conditions, and loops. These are derived from the renewed Dutch curriculum for computer science for secondary schools. In this curriculum, the basic programming constructions are sequencing, repetition (or loops), conditions, and variables (Barendsen, & Tolboom, 2016. pp. 16-17). Barendsen and Tolboom (2016) also mentioned some constructions to support abstraction by programming in the curriculum, but we felt that these topics were not suitable for primary school. There were no specific questions about variables, because this topic is implicitly covered in the other questions. The questions were grouped by topic. For the posttest we used the same kind of questions as for the pretest, but we changed the context a bit. We also changed the order of the questions in

Figure 2: Question inspired by Lewis (2010)

the posttest.

There is little information for this age group about what they know and what can be asked. Lewis (2010) had some questions described in her test for the same target audience. We used these examples to make two similar questions for the tests. In Figure 2 there is a sample question from the pretest. Students were asked how many beats the program lasts in total, and how many beats note 60 is played.

The other questions are based on the description of learning goals in the Dutch curriculum for computer science for secondary education, namely developing code, adapting code, and explaining code (Barendsen, & Tolboom, 2016, pp. 36).

One part of the questions is about sequencing. In these questions it is asked to put the programming blocks in the right order. This tests the ability of the students to develop a program. In some of the questions, we asked the students to give the right order if something in the program changes. These kind of questions correspond with the learning goal that students are able to adapt a program when the requirements change (Barendsen, & Tolboom, 2016, pp. 36). An example of programming blocks which had to be put in the right order is shown in Figure 3. Students were asked to write down the

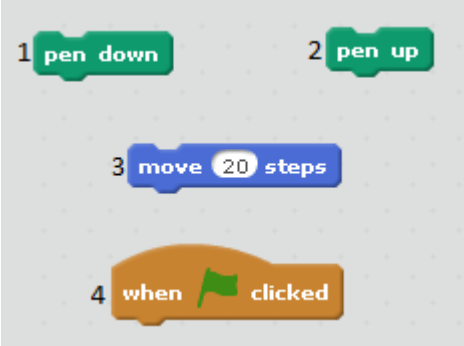


Figure 3: Question where students had to give the right order of programming blocks

Figure 4: Question where students were asked to explain what happened

sequence of blocks in such a way that draw a line of 20 steps.

The last kind of questions are the questions where students explain a piece of code. To test students for explaining the structure and working of a program (Barendsen, & Tolboom, 2016, pp. 36) a question as shown in Figure 4 is used. This also corresponds with a part of the review questions in Calder's book (2011). Students had to explain what the program does when executed.

3 Methods

The general approach for this research is to perform a comparative design-based study. To answer the stated question, we designed two sets of lessons. Both of these sets are based on a different approach, namely construction-

ism or the 4C/ID model. The researcher taught the programming lessons in Scratch on one school for one hour each week. The researcher organised one hour sessions twice a week for three weeks at the other school. The experiment took three to five weeks, and took place in October and November 2016.

3.1 Participants

The lessons were performed on two Dutch primary schools with 5th and 6th grade students, aged nine to twelve years. The schools where the experiments took place are located in Utrecht and Wageningen. The schools participated voluntarily. By distributing information about the project in a newsletter for elementary schools, schools could contact the researcher to participate.

The students from the school in Utrecht were distributed over three classes, one 5th grade, one 6th grade and one combined 5th and 6th grade class. In total 87 students between 9 and 12 years old participated. The 5th grade students (18 female, 13 male) were on average 10,1 years old in the range from 9 to 11. Eight out of thirty-one students were familiar with Scratch. The 6th grade students (11 female, 15 male) were on average 11,1 years old in the range from 10 to 12. In this class, twelve students were familiar with Scratch. The combined 5th and 6th grade class students (14 female, 15 male) were on average 10,4 years old in the range from 9 to 12. Out of the 29 students, 10 students said they knew Scratch. The school is located in the Vogelenbuurt just outside the centre of Utrecht. Furthermore, the school is a public school, which means that the school is accessible for students with all cultural backgrounds and religions.

The students of the school in Wageningen are distributed over two com-

bined 5th and 6th grade classes. In total, this school had 57 students between 9 and 12 years old that participated. The first class had 28 students (17 female, 11 male) with an average age of 10,5 years in the range from 9 to 12. In this class, only five students knew Scratch before the project started. The second class contained 29 students (13 female, 16 male) with an average age of 10,6 in the range from 9 to 12. Eleven students were familiar with Scratch at the start of the project. This school is a Catholic school.

To prevent any major difference between students of the different schools from influencing the research, both teaching approaches were tested on both schools. The distribution of students over the teaching approaches is shown in Tables 2 and 3. For this research, we assumed that similar groups within the schools are equally diverse. This means that we assumed that the different groups within the schools have an equal spread of ethnic backgrounds. Also, we assumed the students had approximately the same learning history, and the same spread of learning levels. So we assumed that by using the fixed groups in the schools, these groups to be comparable within the schools.

	Grade 5	Grade 6	Combined grade 5-6
Constructionism	31 students	27 students	
4C/ID			29 students

Table 2: Distribution of the approaches for the school in Utrecht

	Combined grade 5-6a	Combined grade 5-6b
Constructionism	29 students	
4C/ID		28 students

Table 3: Distribution of the approaches for the school in Wageningen

3.2 Instruments

As explained above, the two teaching approaches which were used differ in how students learned to program in Scratch. Due to the lack of computers in Wageningen, the students learned to program with an app called Pyonkee. This Scratch-based application works on iPads and contains the same programming blocks as Scratch. According to the Scratch website, Pyonkee is based on version 1.4 of Scratch for iPads (https://wiki.scratch.mit.edu/wiki/Pyonkee). The worksheets and tests were not changed for these students. The explanation was also demonstrated with Scratch.

To determine the internal consistency of the tests, Cronbach's alpha (Field, 2009, pp. 673-676) will be calculated. This value has to be above 0.7 for the test to be accepted as reliable. Other test quality measures are the p value, rit, and rir of the questions. The p value tells something about how well students made the question. The rit and rir give an impression of the distinctiveness of the question, and is computed to relate the score of the single question to the total score. By the rir, in contrary to the rit, the total score minus the score of the question for which you calcultate the rir will be used. The rit and rit are considered good from 0.35 and up. For the pretest, Cronbach's alpha was 0.83. The other quality measures are presented in Table 4. The Cronbach's alpha of the posttest was 0.81. Table 5 presents the other quality measures.

	1	2a	2b	3	4	5	6	7	8	9a	9b
p value	.44	.14	.13	.39	.40	.60	.31	.35	.27	.20	.27
variance	.81	.39	.41	.74	.91	1.12	1.73	.91	2.30	.49	.20
rit	.54	.54	.52	.72	.65	.63	.72	.62	.72	.59	.51
rir	.43	.47	.44	.64	.54	.51	.59	.51	.56	.51	.46

Table 4: Statistics of the pretest

	1	2a	2b	3	4	5	6	7a	7b	8	9
p value	.62	.50	.87	.50	.37	.74	.52	.22	.18	.64	.55
variance	.66	.56	.13	.75	1.05	.75	3.00	.55	.54	1.73	.77
rit	.55	.63	.47	.56	.51	.64	.75	.57	.62	.69	.56
rir	.454	.55	.42	.45	.38	.55	.59	.48	.54	.56	.46

Table 5: Statistics of posttest

All tests were graded by the researcher. A randomly picked sample of 30 out of the 129 tests was independently graded by a second grader. This is the case for bot the pre- and posttest. The researcher first looked if there were major differences in scores. For both the pre- and posttest the researcher started a conversation with the second correctors to discuss the differences in how questions were scored. After this, the researcher adapted the correction model where she found it was needed and corrected all the pre- and posttests again. Then, Cohen's Kappa for inter-rater reliability was calculated. The value of Cohen's Kappa has to be above 0.6 to be accepted (McHugh, 2012). For the pretest, Cohen's Kappa was 0.79 and for the posttest 0.68. Therefore we assumed there was enough agreement between the graders.

3.3 Data collection and analysis

Pre- and posttest data on programming competencies of all participants is collected and graded. Data from students who either did not attend the pre- or posttest was removed from the final results. In total, we have data of 129 students. The percentage of correctly answered questions was used to analyse the data. Unless indicated otherwise, the analysis was performed with SPSS.

First, we tested whether the pre- and posttest had a significant difference

	Shapiro-Wilk
Wageningen 4C/ID	$p = .584$
Wageningen constructionism	$p = .020$
Utrecht 4C/ID 5th grade	$p = .252$
Utrecht 4C/ID 6th grade	$p = .888$
Utrecht constructionism	$p = .470$

Table 6: Results of the Shapiro-Wilk normality test of the learning gains

for each teaching approach and school. Therefore, a paired samples t-test is performed (Field, 2009, pp. 326-330). However, the data of the learning gains should be normally distributed to perform a paired samples t-test, and therefore a Shapiro Wilk normality test is performed (Field, 2009, pp. 144-148). If the learning gains are not normally distributed, a condition for performing the t-test is not met. In that case, we have to use a similar test where there is no condition for normally distributed data. This group of tests is called non-parametric tests. Non-parametric tests have a disadvantage in contrast to the t-test: they cannot use the information of the underlying distribution of the data, and are therefore less informative.

The results of the normality tests can be found in Table 6. For the Wageningen construction group, the p value is smaller than 0.05 which means the data is not normally distributed according to this test. However, when we looked at the histogram and Q-Q plot, there was no reason to assume the data is not normally distributed. Therefore, we did not perform non-parametric tests. All other groups have p values that are larger than 0.05 which means the data is normally distributed and the t-tests can be performed. When there is a significant difference between the pre- and posttest, Cohen's d will be calculated in Excel to measure the effect of the intervention.

To compare the two teaching approaches per school, we calculated the learning gain by subtracting the pretest score from the posttest score. We used a two tailed t-test to determine if there is a significant difference between the two teaching approaches (Field, 2009, pp. 334-339). If there is a significant difference, we use the means of the scores to determine which teaching approach was more successful on that school. We also tested if there is a difference between the schools. Therefore, we used a two tailed t-test for both approaches on both schools. Again, if there is a significant difference, the means are used to determine which approach on what school had the higher learning gain.

We also tested if there was a significant difference between the two teaching approach for all the participants together. We used a two tailed t-test to determine if there is a difference (Field, 2009, pp. 334-339). We also performed a two tailed t-test on some variances of the data. For any significant difference, the means will be used to determine which approach had the higher learning gains.

Lastly, we measured the experiences of the students with the programming lessons. These questions were on the front page of the posttest, and consisted of seven questions. Six of these questions should be answered on a five point Likert scale. The last question asked how many hours the students worked with Scratch beside the lessons during the intervention. All the questions gave an overall impression on how students experienced the programming lessons.

4 Results

In this section the main results are presented. First, we want to know if the students learned something during the five lessons they attended. In

	Pretest		Posttest				
	Mean	**SD**	**Mean**	**SD**	***t* value**	***p* value**	**Cohen's *d***
Wageningen 4C/ID	32.23	23.69	62.77	18.65	$t(25) = 7.581$	$p < .001$	1.41
Wageningen constructionism	30.96	24.69	50.52	29.86	$t(26) = 6.129$	$p < .001$	0.71
Utrecht 4C/ID 5th grade	36.44	24.15	46.44	21.40	$t(26) = 2.402$	$p = .024$	0.44
Utrecht 4C/ID 6th grade	23.64	27.61	42.18	26.44	$t(21) = 4.284$	$p < .001$	0.69
Utrecht constructionism	37.93	25.39	50.43	24.59	$t(27) = 4.174$	$p < .001$	0.50

Table 7: Results of the paired sample t-tests for the pre- and posttest for each class

Table 7, the results of the paired sample t-test are shown. All classes did significantly improve between the pre- and posttest ($p < .05$). The effect sizes (Cohen's d) of the school in Wageningen are the highest effects of all classes. The effect measured for the 5th grade students in Utrecht is the lowest effect of all classes. The other classes in Utrecht are in between the lowest and highest effect sizes.

The results of the t-test are presented in Table 8. There is a significant difference between the learning gains for the two teaching approaches in Wageningen ($p = .037$). By looking at the means, the difference is in favour of the 4C/ID approach (M = 30.54) There is no significant difference between the learning gains of the two teaching approaches for Utrecht or the schools combined ($p > .05$). However, when we take a closer look at the data we find that students scored lower on one question in the posttest in comparison with the pretest. This question was about conditionals. When designing the posttest we assumed the students would learn something about coordinates in both approaches and would therefore be able to answer the question. In the pretest we assumed the students did not know this and we therefore used more readable programming blocks for this question. Since most students

	4C/ID			Constructionism					
	Mean	**SD**	**N**	**Mean**	**SD**	**N**	***t* value**	***p* value**	**Cohen's *d***
Wageningen	30.54	20.54	26	19.56	16.58	27	$t(51) = 2.146$	$p = .037$	0.59
Utrecht	13.84	21.27	49	12.50	15.85	28	$t(75) = .289$	$p = .773$	0.07
All*	19.63	22.36	75	15.96	16.45	55	$t(128) = 1.028$	$p = .284$	0.18

Table 8: Results of comparing the two teaching approaches
* equal variances not assumed (F = 5.782, $p = .018$)

		4C/ID			Constructionism					
	Shapiro-Wilk	**Mean**	**SD**	**N**	**Mean**	**SD**	**N**	***t* value**	***p* value**	**Cohen's *d***
Wageningen	$p = .140$	38.46	22.39	26	25.25	17.25	27	$t(51) = 2.411$	$p = .020$	0.66
Utrecht	$p = .403$	18.65	23.00	49	17.69	16.52	28	$t(75) = .132$	$p = .848$	0.05
All*	$p = .156$	25.52	24.55	75	21.41	17.16	55	$t(128) = 1.123$	$p = .263$	.019

Table 9: Results of comparing the two teaching approaches without the excluded question
* equal variances not assumed (F = 9.679, $p = .002$)

failed to answer the question in the posttest, we excluded the question from both the pre- and posttest and performed the t-test once more. The results of these tests are presented in Table 9.

There is no difference in the main results with the excluded question. For the school is Wageningen, there is still a significant difference ($p = .020$). For the other groups, there is no significant difference. This was not the only case we looked at. Due to the small effect of the 5th grade in Utrecht, we were curious what happened if we excluded this class from our original results. The results of this exclusion can be found in Table 10. The main difference is that there is a significant difference between the two teaching approaches for the two schools combined ($p = .018$). However, there is no significant difference for the school in Utrecht.

Lastly, the results of how the students experienced the programming

		4C/ID			Constructionism					
	Shapiro-Wilk	**Mean**	**SD**	**N**	**Mean**	**SD**	**N**	***t* value**	***p* value**	**Cohen's *d***
Utrecht	$p = .818$	18.55	20.30	22	12.50	15.85	28	$t(48) = 1.183$	$p = .243$	0.34
All*	$p = .142$	25.04	21.10	48	15.96	16.45	55	$t(88) = 2.410$	$p = .018$	0.48

Table 10: Results of comparing the two teaching approaches without the 5th grade of Utrecht
* equal variances not assumed (F = 5.095, $p = .026$)

	Did you like constructing programs on a computer?	Did you like working with Scratch?	Was it hard to make programs on the computer?	Was it hard to work with Scratch?	Did you like the programming classes?	How much do you think you have learned?	Hours programmed outside programming classes
Wageningen 4C/ID	4.19	4	3.19	2.88	4.04	3.73	1
Wageningen constructionism	4.04	3.74	2.94	2.89	3.96	3.41	1.06
Utrecht 4C/ID 5th grade	4.46	4.46	3.11	2.64	4.32	3.86	1.07
Utrecht 4C/ID 6th grade	4.23	4.32	2.84	2.52	3.98	3.80	5.18*
Utrecht constructionism	4.38	4.14	2.96	2.88	4.32	3.79	1.75

Table 11: The averages of how the students experienced the programming lessons. For liking 1 was did not enjoy, and 5 liked it a lot. For how hard something was 1 was very easy, and 5 very hard. For learning 1 was nothing, and 5 learned a lot.
* One of the students entered a very high amount of hours, namely 105

lessons are presented in Table 11. Overall, the students scored the programming lessons around 4 out of 5. They scored the difficulty of making programs between 2.80 and 3.20. The students also scored themselves on how much they think they have learned. These scores are between 3.40 and 3.90.

5 Discussion

The aim of this study is to determine which of the two developed teaching approaches resulted in higher learning gains. Because there was not enough evidence to support one theory over the other, our hypothesis is that there is no significant difference between the learning gains of the two approaches. The corresponding research question is:

> *Which teaching approach, constructionism or 4C/ID, results in higher learning gains in terms of programming competencies for students aged 10-12 years?*

If we look at the results, a significant learning gain was achieved by the students between the pre- and posttest. If we then look at the effect, the effect in Wageningen is bigger than in Utrecht. In Wageningen, the effect of the 4C/ID approach is almost 2 times bigger than the effect of the constructionistic approach. Interesting is that in Utrecht the effect between pre- and posttest for both teaching approaches are smaller. For the 4C/ID approach, the 5th grade had a small effect and the 6th grade a medium effect (Cohen, 1992). The constructionistic approach in Utrecht also had a medium effect (Cohen, 1992).

For the t-test that are performed to determine if there was a significant difference between approaches, we can see a significant difference in

Wageningen. The effect of this difference is a medium effect ($d = 0.59$). However, this difference is not measured for Utrecht or for all the students combined. The effects for these comparisons are not even small, which raises some questions about why this difference is only measured for Wageningen. What is also remarkable is that the average learning gains in Wageningen (4C/ID: M=30.54, constructionism: M=19.56) are way higher than in Utrecht (4C/ID: M=13.84, constructionism: M=12.50).

5.1 Why is there only one significant difference measured?

We will first discuss some of the circumstances which might have caused some differences between the schools. The first thing to notice is that no additional time was booked for the posttest in Utrecht. In Wageningen however, extra time was booked. This means that the Utrecht students made the posttest during the last lesson after only half an hour of the final lesson. The students of Utrecht thus had thirty minutes less to learn programming. Additionally the transition between working on computers to focussing on a test may have influenced the results.

In addition to this, the differences between Wageningen and Utrecht might be explained by the planning of the lessons. The students in Wageningen had programming lessons twice a week for three weeks. In Utrecht the students had one programming lesson for five consecutive weeks. The programming lessons in Wageningen were closer to each other, which might have helped the students remember the prior knowledge. By remembering the prior knowledge, the working memory can more efficiently process new information (Kirschner et al., 2006). This could have created an advantage for the Wageningen students.

One other reason which might explain that the difference is only present

in Wageningen is that the school in Utrecht mentioned during the planning of the lessons that the students were quite familiar with a learning by doing approach. It could be that the students in Utrecht are more skilled with constructionistic approaches than the students in Wageningen. Therefore, the researcher asked the school in Wageningen if the constructionistic approach she used was new for the students or if the students did something familiar in the past. The school answered that the students were familiar with this kind of approaches. Still, the students in Wageningen could have had a preference for the 4C/ID approach.

Sun, Pan, and Wang (2010) argued that researchers always have to put the effect sizes in context of the subject and research field. If we take all the above mentioned considerations into account, the planning of the lessons and posttest, we still can say that the effect between pre- and posttest for the 4C/ID approach in Wageningen is large. Because if we look at the other effect sizes between pre- and posttest, then the 4C/ID approach in Wageningen is still two times larger than the highest effect size of the others. The difference between the 4C/ID and constructionistic approach in Wageningen had an effect of 0.59. Because there was no effect between the two approaches in Utrecht and with the two schools combined, the effect size from Wageningen also indicates that the 4C/ID approach was more successful in Wageningen.

The main learning theory which underpins the effect of the 4C/ID approach is the human cognitive architecture (Kirschner et al., 2006). The students get the information in small pieces so the working memory can process them with the prior knowledge. This will eventually result in change in the long term memory (Kirschner et al., 2006), which is needed to develop learning schemes (Van Merriënboer, & Dijkstra, 1997). In Wageningen, the

students had less time between classes. Students might had an advantage here, because they could have remembered the prior knowledge better.

5.2 Further analysis

After looking at the main data, we also looked at two separate cases. The first change that was made to the data was the exclusion of a question of which the validity was doubtful. We saw that the effect size in Wageningen between 4C/ID and constructionism slightly rose to 0.66. In Utrecht and for the two schools combined, there was no difference.

We excluded the 5th grade in Utrecht, because this was the group with the smallest effect. Also, this group needed the most guidance when they made the exercises. When we exclude the youngest group of students which also had the smallest effect size between pre- and posttest, a significant difference for the total group occured. The effect size was 0.48. This difference did still not occur in Utrecht.

With these cases, we can say that for the school in Utrecht there is no proof that there is a significant difference between the two approaches. The doubtful question had little effect on this school. Also the 5th grade students did not cause noise for Utrecht. On the total group, with the exclusion of the 5th grade there was a significant difference with a small effect. We think this difference is mostly explained by the effect of Wageningen on the total group, because there are few students without the 5th grade.

6 Conclusion

The 4C/ID approach had a significant difference in learning gains in comparison with the constructionism for the school in Wageningen. We can thus reject our hypothesis for the school in Wageningen. However, we cannot re-

ject our hypothesis for the school in Utrecht or the total group. The main explanation for the difference between the schools is that the two schools are different. Students might have a preference for one approach over another, and students from different schools have different backgrounds. All students seemed to like the programming lessons, no matter what approach was used.

For future research it could be interesting to see what happens when the lessons are performed with the same time between the lessons on both schools. In addition to this, an adapted posttest so the question which is excluded can be tested is desirable. It would also be interesting to see what happens when the research is performed with more equal classes and more students. This means all 5th and 6th grade classes or all combined 5th and 6th grade classes. With the separate 5th and 6th grade classes there might be a difference in age groups.

It might also be interesting to take the different characteristics of the schools into account. It would furthermore be interesting to perform an adapted version of the 4C/ID lessons followed by the constructionistic approach. This would be a longer intervention than the current experiment. This could be interesting because students said, after taking the 4C/ID classes, that they could now make their own games.

References

Aritajati, C., Rosson, M. B., Pena, J., Cinque, D., & Segura, A. (2015). A socio-cognitive analysis of summer camp outcomes and experiences. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 581-586.

Barendsen, E., & Tolboom, J. (2016). Advies examensprogramma infor-

matie havo/vwo. From: http://www.slo.nl/organisatie/recentepublicaties/adviesinformatica/, Enschede: SLO.

Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computers games about environmental science in a fifth grade classroom. *Educational Technology Research and Development, 59(6)*, 765-782.

Blaho, A., & Salanci, L. U. (2011). Informatics in primary school: principles and experience. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, Springer Berlin Heidelberg, 129-142.

Bruckman, A., & De Bonte, A. (1997). MOOSE goes to school: A comparison of three classrooms using a CSCL environment. In *Proceedings of the 2nd international conference on Computer support for collaborative learning, International Society of the Learning Sciences*, 20-27.

Brunstein, A., Betts, S., & Anderson, J. R. (2009). Practice enables successful learning under minimal guidance. *Journal of Educational Psychology, 101(4)*, 790-802.

Calder, T. (2011). Learning to Scrath a beginners guide to computer programming for kids. Prince Rupert, ISBN: 978-0-9811587-1-6.

Cohen, J. (1992). A power primer. *Psychological bulletin, 112(1)*, 155.

Dasgupta, S., Hale, W., Monroy-Hernández, A., & Hill, B. M. (2016) Remixing as a pathway to computational thinking.

DiSessa, A. A., & Cobb, P. (2004). Ontological innovation and the role of theory in design experiments. *The journal of the learning sciences, 13(1)*, 77-103. doi: 10.1207/s15327809jls1301_4

Duncan, C., Bell, T., & Tanimoto, S. (2014, November). Should your 8-year-old learn coding?. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, ACM, 60-69.

European Schoolnet (2014). Computing our future: Computer programming and coding- priorities, school curricula and initiatives across Europe. From: http://www.eun.org/publications/detail?publicationID=481

Field, A. (2009). *Discovering statistics using SPSS*. Sage publications. ISBN: 978-1-84787-906-6.

Flannery, L. P., Silverman, B., Kazakoff, E. R., Bers, M. U., Bontá, P., & Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. In *Proceedings of the 12th International Conference on Interaction Design and Children*, ACM, 1-10.

Franklin, D., Hill, C., Dwyer, H., Iveland, A., Killian, A., & Harlow, D. (2015). Getting started in teaching and researching computer science in the elementary classroom. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ACM, 552-557.

Jeuring, J., Corbalan, G., van Es, N., Leeuwestein, H., van montfort, J. (2016). Leren programmeren in het PO - een literatuurreview. NRO.

From: https://www.nro.nl/kennisrotondevragenopeenrij/effecten-programmeeronderwijs-op-programmeervaardigheden/

Kafai, Y. B., & Resnick, M. (1996). Constructionism in practice: Designing, thinking, and learning in a digital world. New Jersey: Lawrence Erlbaum Associates.

Kalelioglu, F., & Gülbahar, Y. (2014). The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education, 13(1)*, 33.

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist, 41(2)*, 75-86.

Lee, Y. J. (2011). Scratch: Multimedia programming environment for young gifted learners. *Gifted Child Today, 34(2)*, 26-31.

Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education*, ACM, 346-350.

McHugh, M. L. (2012). Interrater reliability: the kappa statistic. *Biochemia Medica, 22(3)*, 276282.

Ngai, G., Chan, S. C., Leong, H. V., & Ng, V. T. (2013). Designing i* CATch: A multipurpose, education-friendly construction kit for physical and wearable computing. *ACM Transactions on Computing Education (TOCE), 13(2)*.

Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism*, Westport: Ablex Publishing, 1-11.

Poortman, S., & Sloep, P. (2006). Education models.

Sawyer, R. K. (2005). The Cambridge handbook of the learning sciences. Cambridge University Press.

Sun, S., Pan, W., & Wang, L. L. (2010). A comprehensive review of effect size reporting and interpreting practices in academic journals in education and psychology. *Journal of Educational Psychology, 102(4)*, 989 - 1004.

Van Merriënboer, J. J., Clark, R. E., & De Croock, M. B. (2002). Blueprints for complex learning: The 4C/ID-model. *Educational technology research and development, 50(2)*, 39-61, doi: 10.1007/BF02504993.

Van Merriënboer, J.J.G., & Dijkstra, S. (1997). The four-component instructional design model for training complex cognitive skills. *R.D. Tennyson, F. Schott, N.Seel & S. Dijkstra (Eds.),Instructional design: International perspectives. Theory, Research, and Models (Vol. 1).* Mahwah, New Jersey: Lawrence Erlbaum Associates, 427-445.

Van Merriënboer, J. J., & Kirschner, P. A. (2012). Ten steps to complex learning: A systematic approach to four-component instructional design. New York and London: Routledge.

Wilson, A., Hainey, T., & Connolly, T. (2012). Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In *European Conference on Games Based Learning*. Academic Conferences International Limited.

A 4C/ID materials

A.1 Teacher material

Docentenhandleiding programmeerlessen -instructie-

Nienke van Es,

Universiteit Utrecht

Begeleiding door:

Prof. Dr. J.T. Jeuring

Dit lesmateriaal is onderdeel van een vergelijkend onderzoek naar lesmethoden voor het aanleren van programmeren in het basisonderwijs.

Universiteit Utrecht

INHOUDSOPGAVE

Inleiding 3

Vier componenten 4

Taakklassen & leertaken 4

Instructie 5

Les 1: Sequentie 5

Case 1: Meow! 5

Les 2: if-statements 6

Case 1: Tegen een muur lopen 7

Case 2: Onzichtbaar 8

Les 3: Herhalingen 9

Case 1: Van het scherm lopen 9

Case 2: Lopen tot… 9

Case 3: Vierkant tekenen 10

Les 4: Functies en variabelen 11

Case 1: Groeten 11

Case 2: Kleiner of groter? 11

Begeleiding 12

Literatuur 13

INLEIDING

In deze lessenserie van vijf lessen maken de leerlingen kennis met de basisprincipes van het programmeren. De lessenserie is bedoeld voor leerlingen van 10 tot 12 jaar (groep 7 en 8). Het is de bedoeling dat de leerlingen alle opgaven alleen maken, maar ze mogen elkaar wel helpen. Dit programma is ontworpen volgens mijn interpretatie van het 4C/ID model, gebaseerd op de tien stappen zoals omgeschreven door van Merriënboer en Kirschner (2012). Hieronder staat kort per les welke programmeerbegrippen er voornamelijk aan bod komen en een globale tijdsplanning. Bij elke les staat ook een link naar een voorbeeld van de grotere opdrachten. Deze kan gebruikt worden als voorbeeld voor de leerlingen en als voorbeeld uitwerking.

Les 1: Kennismaken met Scratch + sequentie
https://scratch.mit.edu/projects/122910833/
- Instructie (10 min)
- Vierkant tekenen (15 min)
- Gesprek programmeren (30 min)
- Afsluiten en scratch dagboek (5 min)

Les 2: If-statements
https://scratch.mit.edu/projects/122920381/
- Instructie (10 min)
- Voer de muis (45 min)
- Afsluiten en scratch dagboek (5 min)

Les 3: Herhaling
https://scratch.mit.edu/projects/123301313/
- Instructie (10 min)
- Appeltjes vangen (45 min)
- Afsluiten en scratch dagboek (5 min)

Les 4: Functies + variabelen
https://scratch.mit.edu/projects/123313408/
- Instructie (10 min)
- Snake (45 min)
- Afsluiten en scratch dagboek (5 min)

Les 5: Spelletje maken
https://scratch.mit.edu/projects/123488928/
- Race spelletje (55 min)
- Afsluiten en scratch dagboek (5 min)

VIER COMPONENTEN

Het 4C/ID model staat voor “four component instructional design model” en bestaat uit vier componenten. Deze vier componenten zijn de leertaken, ondersteunende informatie, just-in-time informatie en deeltaak oefeningen. De leertaken en ondersteunende informatie zullen later in meer detail besproken worden.In deze lessenserie is de ondersteunende informatie terug te vinden in de instructie voor de leerlingen met de opdrachten beginnen.

De just-in-time informatie is de informatie over *hoe* je bepaalde dingen doet. In Scratch is dit bijvoorbeeld leren hoe je een programma in elkaar kan slepen en de achtergrond kan veranderen. Deze informatie gaat over de *procedurele kennis*. In deze lessenserie staat al deze informatie vermeld in de rood omlijnde blokken.

Just-in-time informatie

De deeltaak oefeningen moeten de leerlingen voorbereiden op de grotere oefeningen, de zogenoemde gehele taken (taken die vergelijkbaar zijn met situaties die ze in het echt tegen zouden kunnen komen). Het is de bedoeling dat leerlingen bekend raken met belangrijke onderdelen om de grotere opdracht te kunnen maken. Echter, mag de leerling zelf kiezen of hij/zij alle deeltaak opdrachten maakt. Wanneer een leerling de stof al beheerst kan deze gewoon verder. In de lessenserie zijn alle deeltaak oefeningen te vinden in de paars omlijnde rechthoeken.

Deeltaak oefeningen

TAAKKLASSEN & LEERTAKEN

Het 4C/ID model maakt gebruik van taakklassen. Dit zijn groepen van leertaken die hoofdzakelijk bij één thema horen, de taakklasse. De instructie die gegeven moet worden aan het begin van elke les gaat over de taakklasse. Hieronder staan de taakklassen en leertaken per les.

- Sequentie
    - Blokken slepen in Scratch
    - Achtergrond veranderen in Scratch
    - Werken met meerderen sprites in Scratch
    - Vierkant tekenen
    - Sprites aansturen
        - bewegen
        - praten
        - …

- If-statements
    - Zelf achtergrond tekenen in Scratch
    - Bewegen met pijltjes toetsen
    - als.. dan.. constructies
- Herhaling
    - Dynamisch bewegen (sprites wisselen)
    - Willekeurige posities
    - als… dan.. constructies
    - Herhalingen
- Functies + variabelen
    - Variabelen aanmaken
    - Variabelen gebruiken
    - Samengestelde condities (d.m.v. functies)
    - Groter dan, kleiner dan, gelijk aan

INSTRUCTIE

Zoals eerder aangegeven, hoort bij elke les een (korte) instructie waarin de verschillende thema's aan de leerlingen uitgelegd worden. Van Merriënboer, Clark en de Croock (2002) beschrijven een aantal strategieën om de theorie over te brengen. Deze informatie valt onder het *supportive information* component van het model. In deze lessenserie is gekozen voor een *verklarende inductieve strategie*, omdat deze strategie tijd effectief is en goed werk voor leerlingen met weinig voorkennis (van Merriënboer, Clark, & de Croock, 2002).

Deze strategie houdt in dat elk thema uitgelegd wordt aan de hand van één of meerdere case studies. Een case studie bestaat uit een realistisch probleem, een (of meerdere!) oplossing(en) en een plan van aanpak. Hierbij is het de bedoeling dat de relaties tussen case studies en de theorie door de docent expliciet gemaakt worden. Hieronder staan per les een aantal case studies met uitwerkingen om te gebruiken tijdens de instructie. Let op! Er zijn bij de oplossingen en plan van aanpak meerdere manieren om tot een correct antwoord te komen. Er zijn ook meerdere correcte antwoorden mogelijk.

LES 1: SEQUENTIE

Deze les moet er eerst iets uitgelegd worden over wat Scratch is. Scratch is een programmeeromgeving die werkt met blokken. Deze blokken passen, als een soort puzzel, in elkaar. Door de blokken aan elkaar te slepen, kunnen leerlingen hun eigen programma's maken. Een voorbeeld van een programma is bijvoorbeeld het spelletje snake (wat ze in les 4 zullen gaan maken!). Na deze korte introductie start de uitleg van sequentie, ofwel opeenvolging. Dit gebeurt aan de hand van case studies.

CASE 1: MEOW!

Als programmeur ben je gevraagd om een klein programma te schrijven. In dit programma moet de kat zeggen dat hij geluid maakt als je op hem klikt. Vervolgens moet hij “Meow” zeggen wanneer je op de kat klikt.

OPLOSSING

PLAN VAN AANPAK

Stap 1: Elk programma heeft een *gebeurtenis* nodig om te starten. Bedenk welke je wilt, zodat de kat iets kan zeggen.

Stap 2: De kat moet wat gaan zeggen, kies hiervoor het juiste blok. Je kan ervoor kiezen om dit in één tekst te doen, maar je kan er ook meer gebruiken. Sleep deze blokken onder de gebeurtenis en verander de tekst (en eventueel de tijd).

Stap 3: Kies de gebeurtenis om ervoor te zorgen dat wanneer er op de kat geklikt wordt er iets kan gebeuren.

Stap 4: Zoek het juiste blok om een geluid af te laten spelen.

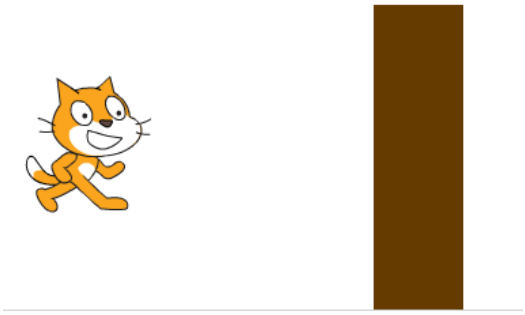
Als het voorbeeld duidelijk is, moet er nog een link gelegd worden met de theorie. De bijbehorende theorie is voor dit stuk sequentie. Dit betekent opeenvolging en gaat over het feit dat elk programmeerblok een instructie is. Deze instructies volgen elkaar op en vormen zo een programma. Dit is de basis van het programmeren, de kunst om de juiste instructies in de juiste volgorde te zetten zodat je programma doet wat je wilt.

LES 2: IF-STATEMENTS

Deze les gaan we het hebben over de als… dan… constructies. In programmeertaal worden dit if-statements genoemd, maar in Scratch zijn de blokken gewoon Nederlands. Dit is een al iets ingewikkelder onderwerp, omdat er hier meer met logica gewerkt wordt. Als een bewering waar is dan wordt een stuk code uitgevoerd. Ook dit wordt aan de hand van case studies geïllustreerd.

CASE 1: TEGEN EEN MUUR LOPEN

Je wordt gevraagd om een programma te schrijven waarbij de kat steeds 20 stappen naar rechts gaat als je de rechter pijltjes toets indrukt. Op de achtergrond moet er aan de rechterkant een muur komen. Wanneer de kat tegen de muur aanloopt moet hij “Auw een muur!” zeggen en een stukje van de muur afgaan.



OPLOSSING

PLAN VAN AANPAK

Stap 1: Zorg ervoor dat er een muur op de achtergrond komt. Dit kan je doen door een bruine balk op de achtergrond te tekenen.

Stap 2: Zoek de juiste gebeurtenis om mee te kunnen beginnen.

Stap 3: Laat de kat eerst 20 stappen lopen als je op het pijltje naar rechts klikt.

Stap 4: Zoek de als… dan… constructie en kijk hoe je kan weten of je de muur raakt. In dit geval is de muur één egale kleur en hebben we hier een blok voor. Verander de kleur van het blok in de juiste kleur.

Stap 5: Laat de kat nu iets zeggen als hij de muur raakt door in de als… dan… constructie verder te werken.

Stap 6: Zorg ervoor dat de kat een aantal stappen terug gaat, zodat hij niet meer tegen de muur aanstaat. Hoeveel stappen er nodig zijn is een kwestie van uitproberen.

CASE 2: ONZICHTBAAR

Dit keer moeten we een programma maken waarbij de kat gaat proberen je muis te vangen. Dit gebeurd wanneer je op spatiebalk drukt. Wanneer je op spatiebalk drukt, verdwijnt de kat en gaat hij naar je muis. Hij wacht een halve seconde, zodat je kan maken dat je wegkomt. Wanneer de kat je muis aanraakt na de halve seconde zegt de kat “Hebbes!”. Als dit niet het geval is moet de kat zeggen dat je weg gekomen bent.

OPLOSSING

PLAN VAN AANPAK

Stap 1: Kies de juiste gebeurtenis om te starten.

Stap 2: Zorg ervoor dat de kat eerst verdwijnt voor hij naar de muisaanwijzer gaat.

Stap 3: Laat de kat een halve seconde wachten.

Stap 4: Gebruik de als... dan… anders... constructie en zoek het juiste blok om te controleren of je de muis aanraakt.

Stap 5: Zorg ervoor dat de kat in allebei de gevallen weer verschijnt.

Stap 6: Programmeer de kat zo dat hij de juiste teksten zegt.

Na deze voorbeelden is het goed om een link te leggen tussen de twee cases. Laat zien wat er hetzelfde is aan de als… dan… constructies. Maak ook duidelijk dat in de tweede case een net wat andere constructie is gebruikt. Leg uit dat dit handig kan zijn als je wilt dat er iets gebeurt wanneer de conditie niet waar is.

LES 3: HERHALINGEN

Het gebruik van herhalingen is erg belangrijk als je bijvoorbeeld een spelletje wilt maken. Je wilt dan namelijk dat je slang in snake blijft bewegen of dat er dingen naar beneden blijven vallen. Er zijn verschillende soorten herhalingen, deze worden geïllustreerd aan de hand van drie korte case studies.

CASE 1: VAN HET SCHERM LOPEN

Voor dit programma moet je ervoor zorgen dat de kat blijft lopen als je op de vlag hebt gedrukt. Hij loopt als het ware van het scherm af.

OPLOSSING

PLAN VAN AANPAK

Stap 1: Kies de juiste gebeurtenis om het programma mee te starten

Stap 2: Bedenk dat het hierbij gaat om één handeling die je achter elkaar wilt doen. Bedenk eerst wat de handeling is, in dit geval een aantal stappen zetten.

Stap 3: Zet de bedachte handeling in de juiste soort herhaling, in dit geval is het een oneindige herhaling.

CASE 2: LOPEN TOT…

Nu willen we niet dat de kat oneindig door blijft lopen, maar stopt wanneer je op de spatiebalk drukt.

OPLOSSING

PLAN VAN AANPAK

Stap 1: Kies de juiste gebeurtenis om het programma mee te starten.

Stap 2: Bedenk dat het hierbij gaat om één handeling die je wilt herhalen tot een bepaalde gebeurtenis. Bedenk eerst wat de handeling is, in dit geval een aantal stappen zetten.

Stap 3: Kies de juiste soort herhaling, in dit geval de herhaal tot. Bedenk nu welk blok nog mist om ervoor te zorgen dat het programma stopt als je op de spatiebalk hebt gedrukt. Zet dit blok op de juiste plek.

CASE 3: VIERKANT TEKENEN

In de eerste les hebben de leerlingen geprobeerd om een vierkant te tekenen door middel van opeenvolging van acties. De opdrachtgever heeft nu gevraagd om de code korter en overzichtelijker te maken, want dit is ook heel belangrijk!

OPLOSSING

PLAN VAN AANPAK

Stap 1: Kies de juiste gebeurtenis om het programma mee te starten

Stap 2: Zorg ervoor dat de pen neergezet wordt.

Stap 3: Bedenk welke instructies er steeds herhaald worden als je een vierkant tekent. Bedenk nu hoe vaak deze instructies herhaald worden.

Stap 4: Kies de juiste soort herhaling en vul in hoe vaak de instructie herhaald moet worden.

Stap 5: Zet de instructie in het herhaalblok.

Stap 6: Haal de pen weer van de achtergrond af.

Leg nu vooral de nadruk op hoe in elke case één of meerdere instructies herhaald werden om het probleem op te lossen. Wanneer leerlingen dus dezelfde instructies vaker willen uitvoeren, is het handig om gebruik te maken van herhaling.

LES 4: FUNCTIES EN VARIABELEN

In Scratch zijn de functies de groene blokken. Deze blokken kunnen gebruikt worden voor condities (groter dan, kleiner dan, gelijk aan), samengestelde condities (en, of, niet), rekenen (plus, min, keer, gedeeld door, modulo) en het samenvoegen van bijvoorbeeld teksten. Dit biedt enorm veel mogelijkheden, waarvan er een paar gebruikt zijn in de case studies.

CASE 1: GROETEN

Vandaag is er aan ons gevraagd of we een programma kunnen maken waarbij de kat vraagt om je naam. Vervolgens voer je een naam in en groet de kat je met de ingevoerde naam.

OPLOSSING

PLAN VAN AANPAK

Stap 1: Kies de juiste gebeurtenis om het programma te starten.

Stap 2: Bedenk welk blok je nodig zou hebben om een vraag te stellen en de gebruiker antwoord te kunnen laten geven. Gebruik dat blok.

Stap 3: Zorg ervoor dat je de groet en ingevoerde naam kan samenvoegen tot één tekst.

Stap 4: Zorg er nu voor dat de groet met naam daadwerkelijk gezegd wordt.

CASE 2: KLEINER OF GROTER?

We hebben een nieuwe uitdaging gekregen, namelijk bepalen of een getal dat de gebruiker invoert groter of kleiner is dan 10. De kat moet kenbaar maken of het ingevoerde getal groter of kleiner is dan 10.

OPLOSSING

PLAN VAN AANPAK

Stap 1: Kies de juiste gebeurtenis om het programma mee te starten.

Stap 2: Bedenk welk blok je nodig zou hebben om een vraag te stellen en de gebruiker antwoord te kunnen laten geven. Gebruik dat blok.

Stap 3: Bedenk welke constructie je nodig hebt om in het geval van een getal kleiner dan 10 een andere reactie te kunnen geven dan wanneer het getal groter is dan 10.

Stap 4: Zorg er nu voor dat je bedenkt wat je nodig hebt om het ingevoerde getal te vergelijken met 10. Gebruik de juiste blokken om de constructie uit stap 3 compleet te maken.

Stap 5: Zet nu nog de juiste blokken in de constructie met de juiste teksten om ervoor te zorgen dat de kat zegt of het getal groter of kleiner is dan 10.

Leg nu de nadruk op het feit dat in het eerste programma het antwoord een tekst was en in het tweede programma een getal. Een variabele kan dus verschillende soorten informatie bevatten! Ook hebben we de groene blokken gebruikt, dit zijn functies. Hiermee kan je dingen met elkaar vergelijken, samenvoegen en nog veel meer.

BEGELEIDING

Voor de begeleiding van leerlingen maken we onderscheid tussen procedurele problemen en semantische problemen. De procedurele problemen zijn gerelateerd aan de programmeeromgeving. Een voorbeeld hiervan is er niet uitkomen hoe je de achtergrond kan veranderen of een sprite kan aanpassen. Hiervoor is de begeleiding eenvoudig. Help de leerling stapsgewijs door het probleem mee en laat zien hoe jij het doet. Probeer, indien nodig, ook te laten zien hoe de leerlingen de resultaten van de fout weer kan herstellen (bijvoorbeeld de achtergrond wissen na mislukte pogingen om een vierkant te tekenen).

Semantische problemen zijn problemen en fouten met de code zelf. De code doet bijvoorbeeld niet wat de leerling had verwacht. Dit zijn lastigere problemen. Probeer hierbij eerst de leerling een hint

of een stapje richting de oplossing te geven. Als dit niet aanslaat, geef dan een suggestie voor een oplossing. Wanneer het dan alsnog niet lukt, loop dan stapsgewijs door de oplossing heen en probeer de leerling zoveel mogelijk zelf te laten doen. Wanneer de leerling een punt bereikt waarop hij of zij weer zelf verder kan, dan laat je de leerling zelf verder gaan.

LITERATUUR

Van Merriënboer, J. J., Clark, R. E., & De Croock, M. B. (2002). Blueprints for complex learning: The 4C/ID-model. *Educational technology research and development*, *50*(2), 39-61.

Van Merriënboer, J. J., & Kirschner, P. A. (2012). *Ten steps to complex learning: A systematic approach to four-component instructional design*. Routledge.

A.2 Student material

Les 1 : Een kort gesprek

Opdrachten

In deze les gaan we kennis maken met Scratch en proberen een kort gesprek te programmeren. Hiervoor maken we gebruik van *opeenvolging*. Laten we hier eerst even mee oefenen.

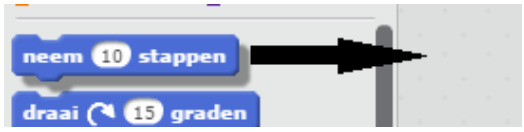
We gaan proberen om de kat een vierkant te laten tekenen. Hieronder zie je een voorbeeld van hoe dit eruit zou kunnen zien. Kijk goed naar de paarse rechthoeken op dit blad als je er niet uitkomt. Ook kun je kijken in de blauwe ovalen op dit blad. Hier staan een paar programmeerblokken in die je misschien nodig hebt. In de rode blokken staat hoe je bepaalde dingen doet in Scratch.

Kun je de kat ook een rechthoek laten tekenen?

Programma maken

1. Kies een categorie:

2. Sleep blokken naar de grijze ruimte



3. Klik op de groene vlag om je programma uit te voeren en op de rode stopknop om het programma te stoppen

Programmeerblokken

Scratch heeft heel veel verschillende programmeerblokken. Bekijk eens de verschillende categorieën en denk goed na welke je nodig kan hebben!

Nienke van Es,
Universiteit Utrecht

Nu we een beetje weten hoe Scratch werkt, gaan we proberen om een gesprek te programmeren. Je mag zelf kiezen wat voor achtergrond en dieren/personen je gebruikt! ☺

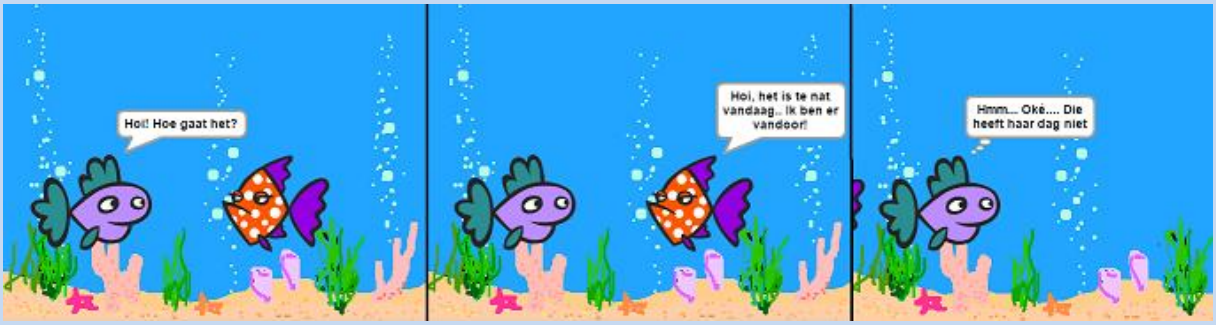
Het is de bedoeling dat het gesprek uit minimaal 2 dieren/personen bestaat en er 3 keer wat gezegd wordt. Hieronder staat een voorbeeld met vissen. Als je klaar bent kun je ook proberen de dieren/personen te laten bewegen!

Achtergrond veranderen

1. Klik dubbel op het icoon van de achtergrond (linksonder)

2. Klik nu aan de rechterkant van het scherm op het volgende icoon:

3. Kies een achtergrond naar keuze



Bewegen in Scratch

Je kan op meerdere manieren bewegen in Scratch. Hieronder staan een aantal opdrachten om je kennis te laten maken met bewegen in Scratch. Bedenk steeds wat voor blokken je nodig hebt!

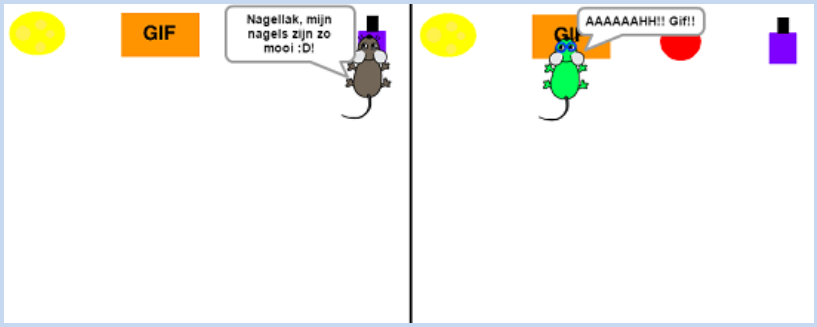
1. Laat de kat 10 stappen zetten wanneer je op de groene vlag drukt.
   a. Wat gebeurd er al je de 10 veranderd door een groter getal?
   b. Wat gebeurd er al je de 10 veranderd door een kleiner getal?
2. Laat de kat naar je muis toekomen wanneer je op de groene vlag drukt.
3. Laat de kat naar positie x: 240 en y: 0 gaan als je op de groene vlag drukt.
   a. Verander het getal bij x een paar keer. Wat gebeurd er?
   b. Verander het getal bij y een paar keer. Wat gebeurd er?
   c. Kun je uitleggen welk getal je moet veranderen als je van links naar rechts wilt bewegen? En wat als je van beneden naar boven wilt bewegen?

Nienke van Es,
Universiteit Utrecht

Les 2 : Voer de muis!

Opdracht

Vandaag gaan we een muis programmeren. De muis heeft eten nodig, maar kan natuurlijk niet alles eten wat hij tegen komt. Hier gaat de muis op reageren. Dit gaan we doen met behulp van de *als... dan...* constructie. Hieronder zie je een voorbeeld van het programma.



Eerst moet de muis rond kunnen bewegen. Herinner je nog hoe je iets kan laten bewegen in Scratch? Wanneer de muis rond kan bewegen, moet er natuurlijk wel wat te eten liggen om op te eten. Hiervoor ga je zelf wat te eten en gif tekenen. Zorg ervoor dat het eten en gif grotendeels uit één kleur bestaat.

Je geeft de muis te eten door op het eten te gaan staan met de muis en dan op spatiebalk te drukken. Vervolgens reageert de muis op wat je hem gevoerd hebt. Begin simpel met alleen iets wat de muis zegt.

Achtergrond tekenen

1. Klik op de achtergrond (linksonder)

2. Klik op achtergronden (rechts) en dan op de kwast

3. Teken je eigen achtergrond!

Ben je klaar? Zorg ervoor dat de muis van uiterlijk veranderd als hij gif te eten krijgt. Probeer nu ook eens een *als… dan… anders…* constructie in je programma te verwerken. Je mag zelf weten hoe je dit doet! Als dit ook gelukt is, dan mag je het programma zelf nog uitbreiden. Je kan meer verschillende soorten eten neerleggen of de muis nog iets heel anders laten doen. Wees creatief ☺!

Als… dan… anders…

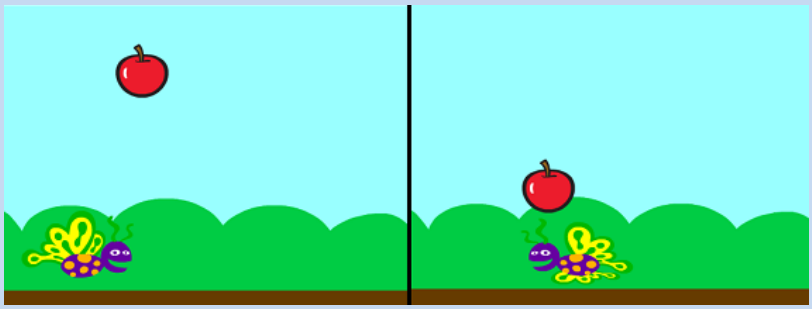
1. Zoek eens het als..dan.. blok op. Welke vorm blokken kun je in dit blok zetten?
2. Maak een programma waarmee als je op de vlag drukt en de spatiebalk ingedrukt houdt de kat zegt “Klik”. Anders zegt de kat “.....”.
3. We gaan nu proberen een iets groter programma te maken.
    a. Teken je eigen achtergrond: maak hier een paar gekleurde cirkels of rechthoeken op.
    b. Wanneer je op de kat klikt moet de kat de kleur roepen van de cirkel of rechthoek waar hij op staat. (Je kan de kat verplaatsen door hem te slepen met je muis!) Gebruik hiervoor de als… dan… blokken.
    c. Wanneer de kat niet op een gekleurde cirkel of rechthoek staat moet hij de kleur van de achtergrond zeggen. Maak dit ook in je programma!

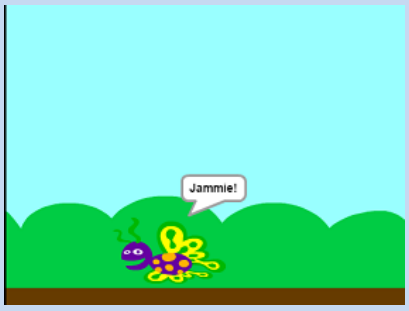
Nienke van Es,
Universiteit Utrecht

Les 3 : Appeltjes vangen

Opdracht

Vandaag gaan we een vlinder laten vliegen. De vlinder gaat namelijk appeltjes vangen die uit de lucht komen vallen! Dit gaan we doen door gebruik te maken van *herhaling*. Het is de bedoeling dat de appel steeds weer bovenin verschijnt als hij is gevangen of op de grond is gevallen. Als de vlinder het appeltje heeft gevangen zegt deze "Jammie!". Hieronder zie je een voorbeeld.

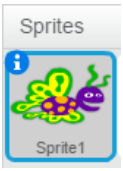




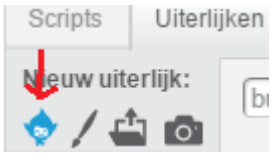
Eerst moet de vlinder gaan bewegen. Denk er aan dat het leuk is als de vlinder haar vleugels echt laat bewegen! Ook moet je het appeltje laten vallen. Je hebt maar één appeltje nodig, deze kun je naar boven verplaatsen als hij gevangen of gevallen is. Kijk goed naar de paarse rechthoek en de blauwe ovaal als je wat hulp nodig hebt!

Beweging bij het lopen

1. Klik op je sprite (linksonder)



2. Klik bij uiterlijken (rechtsboven) op het linker poppetje.



3. Kies het andere uiterlijk van je poppetje.

4. Gebruik het volgende blok elke keer dat je verplaatst:

Universiteit Utrecht

Ben je klaar? Probeer nu je programma uit te breiden, zodat het spel stopt als de appel de grond aanraakt. Kun je nog andere dingen verzinnen om je spel zo leuk mogelijk te maken?

Herhalen, herhalen en herhalen

1. Maak een programma waar de kat je muis volgt zodra je op de groene vlag drukt.
2. Herinner je het tekenen van het vierkant nog uit les 1?
    a. Teken nog eens een vierkant wanneer je op de groene vlag drukt.
    b. Welke blokken worden steeds herhaald?
    c. Maak nu hetzelfde programma, maar nu met een herhaling!
3. Gebruik het herhaal tot... blok om het volgende programma te maken. Je gaat namelijk tikkertje maken! De kat gaat achter de muis aan, totdat hij de muis heeft aangeraakt. Dan zegt de kat: “Hebbes”. Programmeer dit spelletje.

Nienke van Es,
Universiteit Utrecht

Les 4 : Je eigen snake!

Opdracht

Je hebt inmiddels al veel geleerd en bent goed geworden in Scratch! Daarom ga je vandaag je eigen versie van het spelletje snake maken! Hiervoor moet je goed gebruik maken van een aantal *functies en variabelen*. Hieronder zie je een voorbeeld.

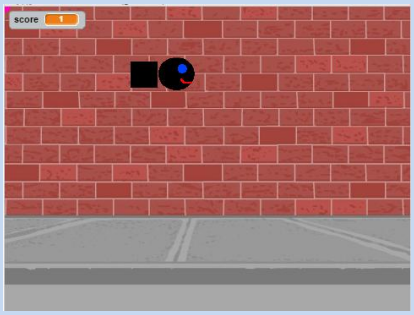
De eerste uitdaging is om je eigen slang te maken. De slang heeft meerderen uiterlijken nodig om steeds te kunnen groeien. In het echte spelletje kan de slang oneindig groeien. In ons spelletje kan de slang maximaal vijf keer langer worden. Na vijf keer stopt het spel. Maak je eigen uiterlijken voor de slang, maar houd het simpel. Je moet hem ook nog programmeren!

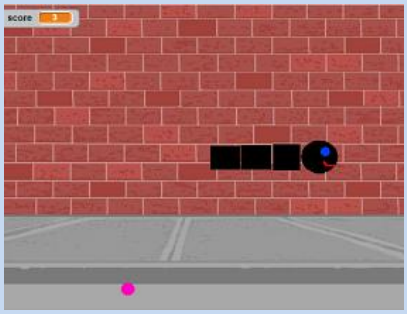
De slang moet ook kunnen bewegen. Dit kan lastig zijn, omdat je slang niet mag stoppen met bewegen als je de pijltjestoets los laat. Denk goed na over hoe je dit wilt doen. Als je er niet uitkomt, dan kun je het beste even oefenen met het paarse rechthoek.

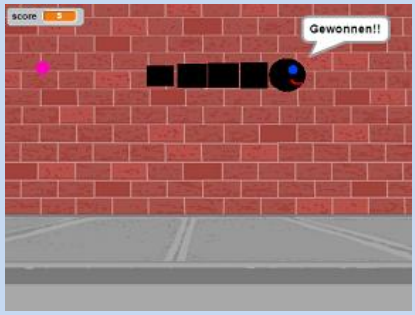
Variabele maken

1. Klik onder het kopje “Data” op “maak een variabele”

2. Geef de variabele een naam en druk op “OK”







Nienke van Es,
Universiteit Utrecht

De volgende uitdaging is om een blokje of rondje op het scherm te laten verschijnen. Dit is het eten voor de slang. Het eten moet steeds op een willekeurige plek in het speelveld verschijnen als het opgegeten is. Als de slang het eten raakt dan wordt de slang ook langer! Als de slang vijf keer eten heeft gehad dan zegt de slang dat je gewonnen hebt en dan is het spel voorbij.

Ben je klaar? Nu kun je het spel proberen wat meer aan te kleden. Zorg ervoor dat de slang aan het begin om je naam vraagt en je groet met je naam voor het spel begint. Ook kun je proberen ervoor te zorgen dat het spel eerder stopt. Bijvoorbeeld als je de randen raakt dan ben je game over.

Variabelen en functies

1. Maak een programma waarbij de kat om je naam vraagt en je vervolgens groet met je naam erin.
2. Kijk bij de categorie “Data” en maak een variabele aan (bijvoorbeeld “score”). Probeer een paar dingen uit je variabele en beantwoord de volgende vragen:
    a. Wat is het verschil tussen “maak score 1” en “verander score met 1”?
    b. Noem een paar programmeerblokken waar je variabele blokje in past.
3. Maak een programma waarbij je een variabele elke keer ophoogt met 1 als je op de kat drukt. Als je 5 keer op de kat hebt geklikt dan zegt de kat “AUW!”.
4. Geef voor de volgende opdrachten aan wat de uitkomst is: *waar* of *niet waar*.
    a. 

    b. 

    c. 

    d. 

    e. 

Nienke van Es,
Universiteit Utrecht

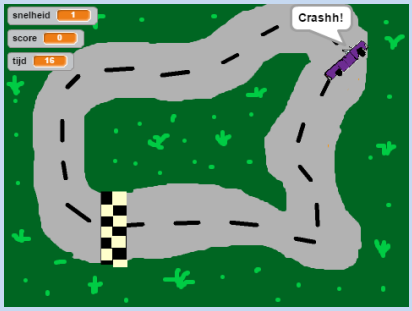
Les 5 : Race naar het eind

Opdracht

Vandaag gaan we alles wat jullie hebben geleerd in Scratch gebruiken om een race spelletje te maken! Om dit voor elkaar te krijgen moet je gebruik maken van *variabelen, als... dan... constructies, herhalingen en functies.* Een voorbeeld van het spel vind je hieronder.

Allereerst heb je voor een race spel natuurlijk een auto nodig en geen kat. Kies een nieuwe sprite die past bij je race spel. Begin eerst met het draaien van de auto. Hierna moet je ervoor zorgen dat zodra je op de groene vlag drukt, de auto blijft rijden.

Nu de auto kan rijden heb je natuurlijk wel een racebaan nodig. Deze mag je zelf ontwerpen! Teken je eigen racebaan, maar zorg ervoor dat de randen van de racebaan één kleur hebben. Dit is straks handig, want dan kun je makkelijker zien of je van de baan af rijd.



De volgende stap is om de auto in de baan te houden. Wanneer je de rand van de baan raakt, dan crasht de auto. Zorg ervoor dat de auto dan terug wordt gezet naar het begin en je opnieuw moet beginnen. Het is ook leuk als de auto zegt dat hij gecrasht is.

Werkt het allemaal? Goed zo! Dit is de basis van je spel. Nu kun je nog een aantal dingen doen. Maak een finish (nieuwe sprite!) en houd een score bij van hoe vaak je over de finish bent gegaan. Ook kun je er nog voor zorgen dat de auto sneller gaat als je op het pijltje omhoog drukt en langzamer als je op pijltje naar beneden drukt. Tot slot kun je ook nog een tijdslimiet maken. Laat de tijd aflopen en als de tijd op 0 staat dan stopt het spel. Natuurlijk mag je zelf ook nog dingen erbij verzinnen ☺!

Nienke van Es,
Universiteit Utrecht

B Constructionism materials

B.1 Teacher material

Docentenhandleiding programmeerlessen -constructionisme-

Nienke van Es,

Universiteit Utrecht

Begeleiding door:

Prof. Dr. J.T. Jeuring

Dit lesmateriaal is onderdeel van een vergelijkend onderzoek naar lesmethoden voor het aanleren van programmeren in het basisonderwijs.

Universiteit Utrecht

INLEIDING

In deze lessenserie van vijf lessen maken de leerlingen kennis met de basisprincipes van het programmeren. Deze lessenserie is bedoeld voor leerlingen van 10 tot 12 jaar (groep 7 en 8). Het is de bedoeling dat de leerlingen elk hun eigen spel gaan ontwerpen, maar ze worden hierbij wel aangemoedigd om elkaar te helpen. Deze lessenserie is ontworpen volgens mijn interpretatie van het constructionisme (Sawyer, 2005, pp. 35-46). Kennis moet hierbij niet overgedragen worden, maar door de leerling zelf geconstrueerd worden. Een belangrijk onderdeel van het constructionisme is dat leerlingen de kennis zelf construeren door middel van betekenisvolle en publieke artefacten. In dit geval wordt het artefact een spel wat de leerling zelf bedacht en geprogrammeerd heeft in Scratch.

Dit kan natuurlijk niet zonder enige kennis van Scratch. Daarom zullen de leerlingen gedurende de eerste les Scratch zelf ontdekken aan de hand van de Scratch kaarten (Wilson, Hainey & Connolly, 2012). De Scratch kaarten zijn te vinden op https://scratch.mit.edu/info/cards. Leerlingen krijgen (alleen of in tweetallen) een Scratch kaart en proberen deze op te lossen. Als ze klaar zijn met hun eigen kaart, kunnen de leerlingen onderling ruilen en zo alle kaarten afwerken.

In de lessen die hierop volgen beginnen de leerlingen met het maken van hun eigen spel. Voordat ze hiermee beginnen kan de docent eerst een voorbeeld laten zien van een spel. Een voorbeeld is te vinden op https://scratch.mit.edu/projects/125028896/. Het spel dat de leerlingen gaan maken mogen ze helemaal zelf verzinnen en vervolgens bouwen. Ze mogen er ook voor kiezen om het doolhof uit het voorbeeld na te maken en uit te breiden. Om de leerlingen die niet goed weten hoe ze moeten beginnen een beetje te helpen, kunnen deze leerlingen het stappenplan uit de opdracht eerst volgen. Het invullen hiervan is niet verplicht, dit mogen de leerlingen zelf besluiten. Leerlingen kunnen namelijk verschillende strategieën hebben om hun spel te maken en dat is ook prima!

Als de leerlingen aan het werk zijn met hun spel, heeft de leerkracht de rol gekregen van een coach. De leerkracht springt bij om leerlingen te helpen die vast zitten, hierover later meer. Een belangrijk aspect bij het ontwerpen en programmeren van het spel is dat de leerlingen gemotiveerd worden om door het lokaal te lopen en elkaar te helpen of inspiratie op te doen. Het is leuk om aan het einde van de lessenserie een half uurtje over te laten om de leerlingen elkaars spel uit te laten proberen.

De indeling van de vijf lessen met de duur van één uur per les ziet er als volgt uit:

Les	**Activiteit**
1	Kennismaking met Scratch d.m.v. de Scratch kaarten
2 t/m 5a	Werken aan het spel!

5b (30 min)	Leerlingen bekijken en spelen elkaars spelletjes en kunnen op het werkblad invullen wat ze ervan vinden.

BEGELEIDING VAN LEERLINGEN

De docent heeft in deze lesmethode een coachende rol, in plaats van voor de klas te staan en kennis over te dragen en te testen. Dit betekent dat de leerlingen zelf aan de slag gaan en tegen veel verschillende dingen aan kunnen lopen. Voor de docent betekent dit soms een snelle schakeling tussen verschillende delen van de stof. De ene leerling kan bijvoorbeeld een vraag stellen over welk programmeerblok er gebruikt moet worden, terwijl de andere leerling een groter stuk code heeft wat niet naar behoren werkt. Er zijn een aantal richtlijnen hoe de docent om kan gaan met de verschillende problemen. Deze zullen hieronder toegelicht worden.

HULP BIEDEN

Als een leerling uitleg nodig heeft om verder te komen, bijvoorbeeld over het concept herhalingen in Scratch, dan zijn er twee manieren om dit aan te pakken. De eerste manier is om gebruik te maken van de andere leerlingen in de klas. Wat voor de ene leerling een heel lastig probleem is, kan voor een andere leerling juist heel eenvoudig zijn. In het constructionisme wordt onderlinge communicatie sterk aangeraden. Om dit te stimuleren kan de docent aan de klas vragen of iemand de leerling met het probleem kan helpen het op te lossen (Sawyer, 2005, pp. 39-40; Baytak, & Land, 2011). Deze aanpak vraagt van de docent wel dat hij/zij ongeveer weet waar de andere leerlingen staan ten opzichte van de stof.

Een andere optie is om zelf de uitleg te geven aan de leerling. Probeer hierbij wel om alleen het noodzakelijke uit te leggen en niet te veel voor te zeggen waar de leerling ook zelf uit kan komen (Blaho, & Salanci). Hierbij moet de docent goed opletten dat hij/zij niet te weinig begeleiding geeft, waardoor de leerling alsnog niet verder kan en gefrustreerd raakt. Bij de uitleg zelf is het verstandig om alle denkstappen die je moet maken om verder te komen met de leerling te bespreken. Stel veel vragen, want op die manier is het makkelijker om in te schatten of de leerling weer zelf verder kan.

ADVISEREN EN REFLECTEREN

Wanneer leerlingen bezig zijn met het maken van een spel, kan het ook zijn dat ze geen vragen hebben over de code maar over bijvoorbeeld het idee van het spel. Nu is de rol van de docent om te adviseren en de leerling te laten reflecteren op hun eigen werk. Ook hierbij kan de hulp van medeleerlingen ingeschakeld worden. De docent kan zelf advies of suggesties geven ter verbetering, maar kan ook een andere leerling het spel laten spelen en die vragen wat hij of zij ervan vindt. Op deze manier stimuleert de docent het gesprek.

LITERATUUR

Baytak, A., & Land, S. M. (2011). An investigation of the artifacts and process of constructing computers games about environmental science in a fifth grade classroom. *Educational Technology Research and Development, 59(6),* pp. 765-782.

Blaho, A., & Salanci, L. U. (2011). Informatics in primary school: principles and experience. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, Springer Berlin Heidelberg, pp. 129-142.

Sawyer, R. K. (2005). *The Cambridge handbook of the learning sciences*. Cambridge University Press.

Wilson, A., Hainey, T., & Connolly, T. (2012). Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In *European Conference on Games Based Learning* (p. 549). Academic Conferences International Limited.

B.2 Student material

Ontwerp je eigen spel!

Nu je een beetje kennis hebt gemaakt met Scratch ga je zelf een spel bedenken en maken! Het is goed om voor je begint goed na te denken over wat je wilt gaan maken. Daarom vind je hieronder een aantal opdrachten die je kunnen helpen, maar je hoeft deze niet te maken als je dat niet wilt.

Start! Dit spel is van:

Mijn spel heet:

Het doel van het spel is om (voorbeeld: prinses te redden)

Teken hierin hoe je spel er uit gaat zien. Denk ook aan de achtergrond!

Spelregels:

Je hebt nu een idee van hoe je spel eruit gaat zien, heel goed! Nu moet je het nog gaan programmeren. Begin heel simpel met je spelletje. Je kan bijvoorbeeld eerst de achtergrond maken en daarna een poppetje erop zetten als je die gaat gebruiken. Hierna kun je bedenken hoe je kan bewegen of wat er moet gebeuren als je ergens op klikt. Test je spel heel vaak! Laat je klasgenoten je spel ook eens testen.

Klaar? Het mooiste zou zijn als je een aantal programmeerblokken in je spel kan gebruiken, omdat je dan Scratch heel goed snapt! Hieronder staat een lijstje met programmeerblokken. Kijk eens naar de code van je spel en vink de programmeerblokken die je hebt gebruikt af. Als je ze nog niet allemaal hebt, kun je iets verzinnen om deze erin te zetten?

Checklist

- ☐ 

of 

- ☐ 

of 

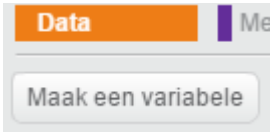
- ☐ 

of 

- ☐ 

of 

of 

- ☐ Een variabele: 

Wat vind ik van mijn spel?

Wat vindt de juf/meester van mijn spel?

Wat vindt mijn klasgenoot van mijn spel?

C Pre- and posttest

C.1 Pretest

Wat weet jij al over programmeren?

Je gaat straks een toets maken om te kijken wat je al weet van programmeren. Het is niet erg als je het antwoord op een vraag niet weet! Probeer de vragen zo goed mogelijk te beantwoorden.

Naam:
Leeftijd:
Geslacht: jongen/meisje

Heb je al eens eerder van Scratch gehoord? Zo ja, waar?

Heb je al eerder met een of meerdere van de onderstaande programma's gewerkt? Je mag hier meer dan één hokje aanvinken.

- ❑ Scratch
- ❑ Logo
- ❑ Lego programmeerbare robots (Mindstorms, etc.)
- ❑ Andere programmeerbare robots
- ❑ Code.org
- ❑ Codecedemy
- ❑ Minecraft
- ❑ GameMaker
- ❑ App Inventor
- ❑ Anders, namelijk...
- ❑ Geen van de bovenstaande

Zo ja, hoeveel uur denk je in totaal te hebben besteed aan deze programma('s)?

Vraag 1. Kijk naar het plaatje:

Hoeveel tellen wordt de noot in totaal afgespeeld? Leg uit.

Vraag 2. Kijk naar het plaatje:

a. Hoeveel tellen duurt dit programma in totaal? Leg uit.

b. Hoeveel tellen wordt noot 60 in totaal afgespeeld? Leg uit.

Vraag 3. Leg uit wat er gebeurt als je dit stukje programma uitvoert:

Vraag 4. Leg uit wat er gebeurt als je dit stukje programma uitvoert:

Vraag 5. Bekijk het volgende plaatje (als je twijfelt over de kleuren, dan mag je ook het nummer wat in het vakje staat opschrijven als antwoord):

a. Welke kleur heeft de pen als je de muis indrukt?

b. Welke kleur heeft de pen als je op het pijltje omlaag drukt?

c. Welke kleur heeft de pen als je op het pijltje omhoog drukt?

d. Welke kleur heeft de pen als je de spatiebalk indrukt?

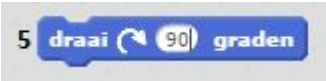
e. Welke kleur heeft de pen als je op het pijltje naar rechts drukt?

f. Welke kleur heeft de pen als je geen toets indrukt?

Vraag 6. Leg uit wat het volgende stukje code doet.

Vraag 7. Je wilt graag een lijn van 20 stappen tekenen. Zet de onderstaande blokken in de goede volgorde zetten om dat te doen. Schrijf alleen de volgorde van de nummers op.

Vraag 8. Je krijgt het volgende programmeerblok erbij:



Je kan nu samen met de blokken uit opdracht 7 een vierkant maken. Je mag de blokken meerdere keren gebruiken. Zet de blokken in de goede volgorde om een vierkant te tekenen. Schrijf alleen de volgorde van de nummers op.

Vraag 9. Bekijk het volgende stukje code:

a. Leg uit wat het stukje code doet.

b. Wat wordt de volgorde van de programmeerblokken als je wilt dat het verplaatsen naar een willekeurige positie gebeurd voordat er gevraagd wordt wat je naam is? Schrijf de nummers op.

C.2 Posttest

Wat heb je geleerd? - Programmeertoets Scratch

Je hebt nu een aantal lessen gehad over programmeren met Scratch! Om te kijken of je dit leuk vond en of je veel geleerd hebt, krijg je nu een toets. Probeer overal zo goed mogelijk antwoord op te geven. Ook als je het antwoord niet weet!

Naam:
Leeftijd:
Geslacht: jongen/meisje

Hoe vond je het om een programma te bouwen met de computer?

Heel stom	Een beetje stom	Neutraal	Leuk	Heel leuk
○	○	○	○	○

Hoe vond je het om met Scratch te werken?

Heel stom	Een beetje stom	Neutraal	Leuk	Heel leuk
○	○	○	○	○

Vond je het maken van programma's moeilijk?

Heel moeilijk	Een beetje moeilijk	Neutraal	Een beetje makkelijk	Heel makkelijk
○	○	○	○	○

Vond je het werken met Scratch moeilijk?

Heel moeilijk	Een beetje moeilijk	Neutraal	Een beetje makkelijk	Heel makkelijk
○	○	○	○	○

Hoe vond je de programmeerlessen in het algemeen?

Heel stom	Een beetje stom	Neutraal	Leuk	Heel leuk
○	○	○	○	○

Hoeveel denk je dat je geleerd hebt?

Niks	Een beetje	Neutraal	Veel	Heel veel
○	○	○	○	○

Heb je buiten de Scratch lessen zelf nog met Scratch geprogrammeerd? (bijvoorbeeld thuis of als je op school klaar was met je werk?) Zo ja, hoeveel uur denk je?

Vraag 1 Leg uit wat er gebeurt als je het volgende stukje programma uitvoert:

Vraag 2. Bekijk het volgende stukje code:

a. Leg uit wat het stukje code doet.

b. Wat wordt de volgorde van de programmeerblokken als je wilt dat er eerst een geluid afgespeeld wordt, voor er gevraagd wordt hoe oud je bent? Schrijf alleen de nummers op.

Vraag 3. Kijk naar het plaatje:

Hoeveel tellen wordt het slagwerk in totaal afgespeeld? Leg uit.

Vraag 4

Kijk naar het plaatje:

a. Welke kleur is de pen als de muis op punt x = 50 en y = 0 staat?

b. Welke kleur is de pen als de muis op punt x = 0 en y = -5 staat?

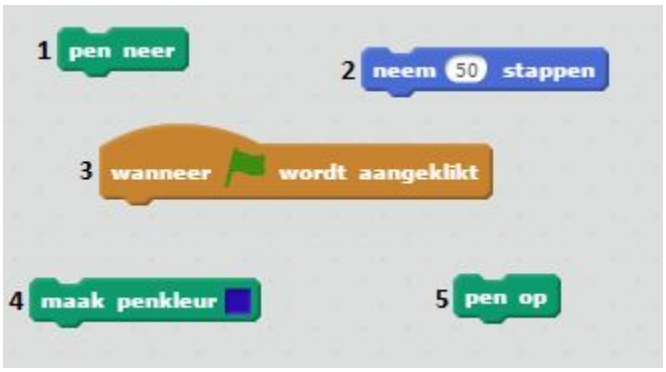
c. Welke kleur is de pen als de muis op punt x = 0 en y = 220 staat?

d. Welke kleur is de pen als de muis op punt x = 10 en y = 0 staat?

e. Welke kleur is de pen als de muis op punt x = -10 en y = 0 staat?

f. Welke kleur is de pen als de muis op punt x = 0 en y = 150 staat?

Vraag 5. Je wilt graag een lijn in de kleur blauw tekenen. Deze lijn is 50 stappen. Zet de onderstaande blokken in de goede volgorde om dit voor elkaar te krijgen. Schrijf alleen de volgorde van de nummers op.



Vraag 6. Je krijgt het volgende programmeer blok erbij:

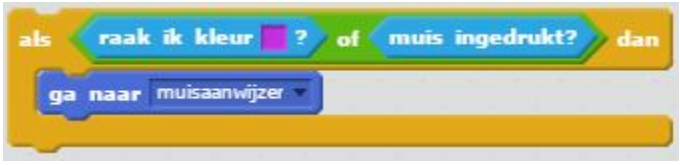
Je kan nu samen met de blokken uit opdracht 5 een blauwe driehoek maken. Je mag de blokken meerdere keren gebruiken. Zet de blokken in de goede volgorde om een driehoek te tekenen. Schrijf alleen de volgorde van de nummers op.

Vraag 7. Kijk naar het plaatje:

a. Hoeveel tellen duurt dit programma in totaal? Leg uit.

b. Hoeveel tellen wordt slagwerk 13 in totaal afgespeeld? Leg uit.

Vraag 8. Leg uit wat het volgende stukje code doet.



Vraag 9. Leg uit wat er gebeurt als je dit stukje programma uitvoert:

C.3 Correction model

Pretest

Vraag 1 (2pt)
Antwoord: 10 tellen, want 10 x 1 tel noot 53.

1pt: 10 tellen (leerling moet hiervoor wel duidelijk het antwoord noemen!)
1pt: uitleg goed
2pt: 10 tellen, want 10 keer 1 tel noot 53 of soortgelijke uitleg.

Geen punten voor de uitleg wanneer 53 wordt gebruikt als berekening.

Vraag 2 (4pt)
Antwoord a: 55 tellen, want 5 x 1 tel + 5 x 10 x 1 tel

1pt: 55 tellen
2pt: 55 tellen (1pt), want 5 x 1 tel + 5 x 10 x 1 tel of soortgelijke uitleg.

Antwoord b: 50 tellen, noot 60 wordt 10 keer herhaalt per ronde. Die ronde wordt nog 5 keer herhaalt (10*5* 1 tel = 50).

1pt: 50 tellen
2pt: 50 tellen, want noot 60 wordt 10 keer herhaalt per ronde. Die ronde wordt nog 5 keer herhaalt (10*5* 1 tel = 50) of soortgelijke uitleg.

Vraag 3 (2pt)
Antwoord: De pop (mag ook aangeduid worden als “iets” of “ding”) verschijnt, zegt Hallo! voor 2 seconden en verdwijnt weer. Hierna gebeurd er 2 seconden niks om vervolgens weer opnieuw te beginnen (verschijnen, zeggen, verdwijnen). Dit herhaalt zich oneindig.

1pt: De pop verschijnt, zegt Hallo! voor 2 seconden en verdwijnt weer.
2pt: De leerling spreekt hier over een herhaling. De pop verschijnt, zegt Hallo! voor 2 seconden en verdwijnt weer voor 2 seconden. Dit blijft zich herhalen/dit gaat zo door/ander soortgelijk antwoord.

Vraag 4 (2pt)

1 pt: Er komt Hallo of Dag voor 2 seconden op het scherm.
2pt: Als je de muis indrukt dan komt er Hallo op het scherm, anders komt er Dag op het scherm.
2 pt: Leerling noemt ook dat je de muis moet indrukken en dan Hallo verschijnt en anders Dag

Vraag 5 (3pt)
Elk goed antwoord is 0.5 pt waard
Antwoord a: paars (3)
Antwoord b: groen (2)
Antwoord c: rood (1)
Antwoord d: blauw (4)
Antwoord e: oranje/geel (5)
Antwoord f: rood (1)

Vraag 6 (5pt)

Antwoord a (3pt): Als de rand aangeraakt wordt EN toets ‘a’ wordt ingedrukt dan verplaatst de pop naar een willekeurige positie.

1pt: Als de rand wordt aangeraakt dan verplaatst de pop.

1pt: Als de toets ‘a’ wordt ingedrukt dan verplaatst de pop.

2pt: Een van bovenstaande antwoorden aangevuld met dat de pop naar een willekeurige positie verplaatst wordt.

2pt: De pop verplaatst (maar er wordt niet genoemd naar een willekeurige positie) als de rand aangeraakt wordt EN toets ‘a’ ingedrukt wordt

3pt: De leerling heeft het inzicht dat als zowel de rand aangeraakt wordt en toets ‘a’ wordt ingedrukt de pop naar een willekeurige positie wordt verplaatst.

Vraag 7 (2pt)

Voor elk afwijkend getal in de reeks -1 pt

Antwoord: 4-1-3-2, wanneer 2 getallen zijn omgedraaid, dan -2 punten.

Vraag 8 (4pt)

Voor elk afwijkend getal in de reeksen hieronder -1 pt. Als 4 aan het einde van de reeks staat is dit -1 punt (niet -2 omdat hij ook aan het begin mist).

Antwoord (4pt): 4-1-3-5-3-5-3-5-3-(5)-2

Antwoord(3pt): 4-1-3-5-(2)-1-3-5-(2)-1-3-5-(2)-1-3-2

Antwoord (2pt): Leerling legt tekstueel het antwoord uit maar mist het deel van pen op/pen neer

Antwoord (2pt): 4-2-3-5-3-5-3-5-3-(5)-1

Vraag 9 (3 pt)

Antwoord a (2pt): Als er op de vlag gedrukt wordt dan wordt er gevraagd hoe je heet. Vervolgens wordt er Hallo plus het antwoord wat je hebt gegeven voor 2 seconden laten zien en verplaatst het poppetje naar een willekeurige positie.

1pt: De leerling noemt dat er gevraagd wordt om de naam, er gewacht wordt en een groet komt (maar niet dat er gegroet wordt met de naam die hebt ingevuld) en noemt dat je naam een willekeurige positie gaat.

1pt: De leerling geeft aan dat er om je naam gevraagd wordt en dat er vervolgens een

groet op het scherm verschijnt.

2pt: De leerling noemt het bovenstaande en noemt ook dat de groet 2 seconden duurt,

de pop naar een willekeurige positie verplaatst en het pas start als je op de vlag drukt.

Antwoord b (1pt): 1-4-2-3

Totaal 2+4+2+2+3+3+2+4+3 = 25 pt

Cijfer = (behaalde punten/25) * 9 + 1

Posttest

Vraag 1 (2pt)
Antwoord: Als de spatiebalk wordt ingedrukt dan neemt de pop 10 stappen, anders dan draait de pop 15 graden linksom.

1pt: Als er op de spatiebalk wordt gedrukt dan neemt de pop (of avatar/de kat/iets soortgelijk) 10 stappen.

2pt: De leerling noemt ook dat anders de pop (15 graden (linksom)) draait.

Vraag 2 (3pt)
Antwoord a: Wanneer er op de vlag wordt geklikt, dan vraagt de pop hoe oud je bent en wacht op het antwoord. Als je het antwoord ingevoerd hebt dan zegt de pop: “Zo hé! Jij bent dus: ” met daarachter de ingevoerde leeftijd voor 2 seconden. Hierna wordt het geluid “meow” afgespeeld.

1pt: De leerling noemt dat er gevraagd wordt hoe oud je bent en dit vervolgens gezegd

wordt (bijv. Zo hé! Jij bent dus: *leeftijd*). Dit punt mag gegeven worden wanneer de leerling begrijpt dat het antwoord dat is ingetypt gebruikt wordt.

1pt: De leerling noemt dat het programma start (als je op de vlag drukt). Ook noemt de leerling dat hierna het geluid “meow” afgespeeld (of gezegd) wordt.

2pt: De twee bovenstaand antwoorden samengevoegd.

Antwoord b (1pt): 1-4-2-3

Vraag 3 (2pt)
Antwoord: 20 tellen (10 x 2)

1pt: 20 tellen

2pt: 20 tellen, want 10 x 2 = 20 (of soortgelijke uitleg)

Vraag 4 (3pt)
Voor elk goed antwoord 0.5 pt.
Antwoord a: groen (2)
Antwoord b: rood (1)
Antwoord c: oranje (5)
Antwoord d: groen (2)
Antwoord e: 3
Antwoord f: blauw (4)

Vraag 5 (2pt)
Voor elk afwijkend getal in de reeks -1 pt
Antwoord: 3-1-4-2-5 of 3-4-1-2-5

Vraag 6 (4pt)
Voor elk afwijkend getal in de volgende reeksen -1 pt, leerling mag ipv 1-4 ook 4-1 hebben. Wanneer een leerling een “2-6” te veel of te weinig is, dan zijn dit -2 punten. Er zijn

meerdere variaties die kloppen. Wanneer de reeks wel klopt, maar minder efficiënt is (zoals het 4pt antwoord), dan mogen er 3 punten gegeven worden.

4pt: 3-1-4-2-6-2-6-2-(6)-5

3pt: 3-1-4-2-(5)-6-1-(4)-2-(5)-6-1-(4)-2-5-(6)

Vraag 7 (4pt)

Antwoord a: 120 tellen, want 10 x 2 (slagwerk 4) + 10 x 5 x 2 (slagwerk 13)

1pt: 120 tellen

2pt: 120 tellen, want 10 x 2 (slagwerk 4) + 10 x 5 x 2 (slagwerk 13) (of soortgelijke uitleg)

Antwoord b: 100 tellen, want 10 x 5 x 2

1pt: 100 tellen

2pt: 100 tellen, want 10 x 5 x 2 (of soortgelijke uitleg)

Vraag 8 (3pt)

Antwoord: Als de pop de kleur aanraakt OF de muis wordt ingedrukt, dan verplaatst de pop naar de muisaanwijzer.

1pt: De pop verplaatst als hij de kleur paars aanraakt.

1pt: De pop verplaatst als de muis ingedrukt wordt.

2pt: Een van bovenstaande antwoorden, aangevuld met dat de pop verplaatst naar de muisaanwijzer.

2pt: De pop verplaatst (maar wordt niet genoemd waarheen) als hij de kleur paars aanraakt OF de muis wordt ingedrukt.

3pt: De pop verplaatst naar de muisaanwijzer (gaat de muis achter na/volgt de muis, etc..) als hij de kleur aanraakt OF als de muis wordt ingedrukt.

Vraag 9 (2pt)

Antwoord: De pen wordt neergezet en er worden 5 stappen gezet (lijn van 5 stappen). Vervolgens wordt de pen omhoog gehaald en worden er weer 5 stappen gezet (tussenruimte van 5 stappen). Dit wordt oneindig herhaalt. (In feite teken je een oneindig doorlopende stippellijn).

1pt: De leerling spreekt over herhaling, maar heeft wat er herhaalt wordt niet helemaal goed.

1pt: De pen gaat neer, er worden 5 stappen gezet. De pen gaat op en er worden 5 stappen gezet.

1pt: Er gaat/je legt iets neer, er worden 5 stappen gezet, je pakt het weer op en neemt 5 stappen (pen wordt niet genoemd). Ook spreekt de leerling over herhaling.

2pt: De leerling spreekt hier over een herhaling. De pen gaat neer, er worden 5 stappen gezet. De pen gaat op en er worden 5 stappen gezet. Dit blijft zich herhalen/dit gaat zo door/ander soortgelijk antwoord.

2pt: De leerling zegt dat je een stippellijn/allemaal streepjes krijgt

Totaal 2+3+2+3+2+4+4+3+2 = 25pt

Cijfer = (behaalde punten/25) * 9 + 1