Universiteit Leiden
The Netherlands

Opleiding Informatica

Handling of abstraction by children using the Hedy programming language

Michael de Rooij

Supervisors:
Efthimia Aivaloglou, Felienne Hermans

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

30/06/2022

Abstract

This research is an observation study examining how children demonstrate computational thinking and the concept of abstraction in their behaviour while working on programming assignments in Hedy. For this research, participants of twelve or thirteen years old worked on a set of assignments in pairs, in order to stimulate verbal discussion. To analyse the behaviour of the students, the audio and the screen of the laptop they were working on were recorded. Their behaviour was encoded using an operationalisation of the layers of abstraction model. This model consists of four layers, being the problem layer, the design layer, the code layer and the execution layer. We found that the problem layer occurred the least and mostly at the beginning of the assignment. The design layer was used for different reasons. Some pairs used it mostly for discussion, while others used it more for studying learning materials. The code layer occurred the most and the students often switched between this layer and the execution layer. In general, the students switched often between the different layers, which is in agreement with literature. Some students discussed more than others and the concept of indentation seems to be difficult to understand. Hedy seems like a promising tool to introduce children to programming at an early age.

Contents

1 Introduction

Learning how to program is considered to be a difficult topic for inexperienced people [DA12][Jen02]. Dijkstra argued in 1989 already that programming should be viewed as a "radical novelty" and the general approach of learning something new does not work properly for programming. He also compares programming to "creating and learning a new foreign language that can not be translated into one's mother tongue" [D+89]. Skills such as problem solving skills and logical reasoning are perceived to be important to "survive in the information age" [KG14]. However, literature shows that many young students lack these skills [MM12][PT12]. Programming is seen as important knowledge to develop these skills [KG14]. Nowadays, programming education is getting increasingly integrated into the education of children of all ages and this requires programming tools.

Several programming languages and tools have been developed to support the introduction of programming to children. This research is focusing on the programming language Hedy [Her20]. Hedy is a textual programming language, designed to help children learn programming step by step. To conduct this research, students of twelve and thirteen years old have been observed while doing assignments in Hedy. Many aspects of the workflow of students during an programming assignment could be observed, but this research focuses on the cognitive process of abstraction, which is part of computational thinking [SW13]. In order to accurately transform the observations into behaviour related to a certain use of abstraction, an operationalisation of the layers of abstraction model will be used, based on the model used by Faber et al. [FKW+19]. This model can be found in Table 3.

The goal of this research is to gain insights in how children learn programming using Hedy. These insights can help in understanding if Hedy suffices in gradually introducing students to new concepts. Finally, the results can be used to improve Hedy. A good way to assess how children work on assignments in Hedy is to look at how children use abstraction in their observed behaviour. This results in the following research question:

How do children demonstrate the layers of abstraction in programming exercises in Hedy and how does this differ per level?

A total of eight students have worked on three different assignments in Hedy. These students were grouped into four pairs, to stimulate verbal discussion during the assignments. While the students were working on the assignments, their audio and screen were recorded for analysis.

The remainder of this thesis is organised as follows: Section 2 discusses related work, which contains topics that form a basis for this research. Section 3 discusses how this research is conducted. Section 4 describes the results. Section 5 concludes.

Finally, I want to thank my supervisors Fenia Aivaloglou and Felienne Hermans from LIACS for their continuous support during my bachelor thesis. I have learned a lot during this period and it was a pleasure to be a part of the PERL research group.

2 Related Work

Section 2.1 and Section 2.2 discuss the main concepts this research will examine in Hedy. Section 2.3 discusses the current situation of programming education and Section 2.4 discusses some background theory about Hedy and how this language is designed. Finally, related work that has previously used a similar method will be listed and potential differences will be discussed in Section 2.5.

2.1 Computational thinking

Computational thinking is a concept that has been broadly researched. A definition of computational thinking that is frequently used in literature is the following:
"Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent."[Win11] An information-processing agent refers to a subject that can solve tasks given a set of instructions. As this definition suggests, computational thinking is a way of analytical thinking in order to solve (computational) problems.

This concept originates in computer science, but literature suggests that computational thinking is a fundamental skill that can be used outside of computer science as well. Therefore, it is also suggested to integrate computational thinking into children's education [Win06]. Nowadays, computational thinking is getting increasingly integrated into the curriculum of students of all ages. An exploratory case study has been conducted on children of five to six years old solving programming problems in a Logo-based environment [FGM13]. Another study shows how children of four years old can already learn aspects of, amongst others, computational thinking and robotics [BFKS14]. Besides these examples, one study discusses the use of the tool CRISPEE to understand how children use computational thinking in relation to life science and biology subjects [Str21]. This also shows how computational thinking is getting increasingly involved in disciplines outside of computer science.

2.2 Layers of abstraction

Literature suggests that some terms should always be mentioned when proposing a definition for computational thinking. These terms include abstraction, decomposition, algorithmic thinking, generalisation and evaluation [SW13]. This research will focus on the term abstraction. Abstraction can be described as a cognitive process that entails "deciding what details we need to highlight and what details we can ignore" [Win08].

One of the models for the analysis of abstraction that literature has identified is the layers of abstraction model (LOA) by Perrenet et al. [PGK05][PK06] and forms the basis for the model in this research. Several publications have used this model as a basis for their own layers of abstraction model, all focusing their research on children [FKW+19][SA16][WCM+18]. The original abstraction model consists of four layers [PGK05][PK06]:

- Problem layer: This layer discusses behaviour related to understanding the exercise.
- Design layer: This layer discusses behaviour related to designing a solution for the exercise and expressing this in human language, without actual code.

- Code layer: This layer discusses behaviour related to translating the design of the solution to actual code.
- Execution layer: This layer discusses behaviour related to the output of the code.

In order to assess how children use abstraction in their observed behaviour, we need an operationalisation of the layers of abstraction model. The model used in this research can be found in Table 3 and is discussed in Section 3.4.

2.3 Teaching programming

Starting to learn programming is not just one skill or one set of skills that needs to be acquired. Many different aspects come together while building a piece of code. When programming is taught to inexperienced students, one should use a careful approach, in order to teach the students programming with all its facets. To educate students in the best way possible, it is necessary to indicate potential pitfalls that the students may encounter while programming.

Related work has identified two main reasons that contribute to programming being seen as a difficult concept. The first reason is the absence of general problem solving skills. Programming includes many computational problems, that require computational thinking to solve. If the general solving skills are lacking, solving computational problems will be even more difficult [GM07]. This issue will be discussed more thoroughly in Section 2.1.

The second reason is about conventional teaching methods being used by many introductory programming courses to teach programming to novice programmers [RRR03]. For a significant portion of students, these methods do not supply in the needs of the students to learn programming in a good manner. It was found that teaching is frequently focused too much on teaching the programming language and its syntax itself, rather than using the programming language to teach what problems can be solved with programming. Though it is important to get used to how a certain language and its characteristics work syntactically, knowing how to use these to create for example an algorithm is of high importance as well [GM07]. Education focusing more on the syntax of a programming language introduces another problem. What programming language is best to use for educational purposes? Many problems can be solved with various programming languages, but some problems require a (group of) specific language(s). This means that it seems useful to first teach students about required programming concepts using a programming language that is suitable for education, before focusing on syntactical features of certain languages [GM07].

2.4 The incremental approach and Hedy

Many different approaches have already been discovered to assist novice students with their first steps in learning programming. One of these approaches is the incremental approach, where a language is divided into smaller sections. Each of these sections introduces new features and operators, which enables students to learn a language more gradually [BKMT94]. This also enables students to focus more on problem solving from the start instead of learning and remembering a lot of syntax. The first time this method was introduced dates from 1977, by Holt et al. [HWBC77]. They used

this method to teach the programming language PL/1[1], which IBM is still using today. Over the years, the incremental approach has been widely applied on the teaching of different programming languages. For instance, in 1985 PMS was created to help students learn the language Pascal [TMK85][BKMT94]. In 2016, Hong Huang discussed the importance of incremental teaching in the language Java [Hua16]. These examples are only a small part of a bigger quantity of incremental teaching applications.

In order to change the general perception of programming being difficult, it is necessary to make learning programming easy and accessible by all ages. For this, it will be valuable to introduce children to programming, in such a way that it is appealing to them. Scratch is a good example of such a tool [RMMH+09]. Scratch is a visual programming language, using blocks that can be connected by children to create programs. As pointed out by the creators of Scratch, "there is none of the obscure syntax or punctuation of traditional programming languages" [RMMH+09]. This means that we need more tools, to aid children's knowledge of other programming concepts.

Hedy[2] is a programming language that is designed to enable children to start learning programming. Hedy is created by Felienne Hermans at the Leiden Institute for Advanced Computer Science (LIACS). Hedy is a gradual programming language, which means that children can learn new syntax and operators step by step. This approach is clearly in line with the incremental approach, discussed earlier. Hedy consists of many levels, each introducing new features. Unlike Scratch, Hedy is a textual language implemented in Python, hence young students are already getting into contact with operators and features that are being used in professional languages [Her20]. From a theoretical perspective, Hedy seems like a promising tool to teach children programming. In order to test this practically, we will examine how children use computational thinking and what levels of abstraction children reach while programming in Hedy.

2.5 Abstraction in other programming languages

Previous literature has used methods that are at least partially comparable to the method used in this research. One study attempted to operationalise the layers of abstraction model described in Section 2.2. This study conducted their research with five to six year old children. A significant difference with this research is that they have used Cubetto for the programming, which is a educational robot, where as this research uses Hedy, which is a textual programming language [FKW+19]. Another study has roughly adopted the method of Faber et al.[FKW+19] and used this to observe the behaviour of ten to twelve year old children with a visual impairment using the programming tool Sonic Pi [Bol21]. Besides the different programming language used, a clear difference is that their study was focused on children with a visual impairment, where as the children in this research have no visual impairment.

There has also been one study that observed how children interact with CRISPEE, which is a tool for children to learn about genes and how they work with living organisms [SVSB20]. Like in this research, a significant part of the participants worked in pairs and the observed interactions of the participants with the program were encoded to categories. However, their study focuses more

[1]https://www.ibm.com/products/pli-compiler-zos
[2]https://www.hedycode.com/

on the possible interactions with the program, while this research focuses more on the cognitive processes involved in those interactions.

3 Methodology

3.1 Participants

All students that participated in this observation study are first year HAVO/VWO high school students (following the Dutch school system). The data collection was done during the programming classes of the students, where they reached level ten of Hedy before the researchers arrived.

Observing the layers of abstraction in the behaviour of the students while they are doing the assignments might be difficult if only the flow of programming is analysed. To improve the analysis, students were asked to think aloud while programming. To stimulate this behaviour even more, students did the assignments in pairs, so they would more naturally discuss what they are programming. The teacher of the students made the pairs of students, taking into account whether students could get along well or not. This improves their collaboration as well. Previous programming experience was not considered while making the pairs, so this differed significantly across the pairs, as can be seen in Table 1.

A total of eight students participated in this research. six of those students, grouped in pairs, did the first and second assignment. The other two students participated in the third assignment, also being grouped in a pair.

Pair	Age	gender	Programming experience	Assignments participated
1	12 & 13	Both female	None	1 & 2
2	Both 12	Both male	None	1 & 2
3	Both 12	Both female	Both Scratch	1 & 2
4	12 & 13	Both female	Scratch	3

Table 1: Information about the pairs of students that participated (shown for each assignment)

3.2 Procedure

Participating students did the assignment in a separate, isolated room on a provided laptop. The students had access to a document stating the assignment at hand and a browser with Hedy opened. After a short introduction, the students could start with the assignment. While the students were working, the screen of the laptop and the audio was recorded. This recording will help to analyse after the data collection which layers of abstraction occurred at what time during the assignment. To observe the process of correcting mistakes, students were given as few hints as possible. Only when the students were clearly stuck a hint was provided.

We obtained ethical approval for this research from the Ethics Committee of the Faculty of Science of Leiden University, prior to collecting and analysing the data. The parents of the students received a letter, where they were informed about the attendance of multiple researchers in the programming classes of their child. This letter contained information about what the research entails and that students are required for the experiments. It was clearly stated that participating in this research is optional. Parents were informed that the audio and the screen would be recorded during the experiments. This also included information about how the data is processed. Parents were asked to give consent for the participation of their child in the research and the recording of the screen and audio separately.

Before the students were assigned to participate in the research, they got a short introduction to the research and why this research is conducted. Then the participants were taken from the group that was willing to participate. The students were instructed on the details of the assignment and the recording. There was room for questions before the start of the assignment and the students were motivated to ask any questions they had, even if it was during the assignment.

3.3 Assignments

Many programming concepts are introduced to students via Hedy. For the best understanding of how children approach computational problems using Hedy, we have decided to create the assignments around the concepts of conditional statements, loops and logical operators, because these concepts have been found to be difficult for novice programmers [QL17][AH16][SF13]. All three assignments are roughly based on already available assignments in the different levels of Hedy. Every assignment in Hedy consists of an example code block that the students can use to get to a solution. To keep consistency with the lectures, the students were allowed to use the example code block that was selected beforehand by us. Table 2 shows what levels of Hedy the assignments are based on and what programming concepts are important in that level.

Assignment	Hedy level	Programming concept(s)
1	7-10	if-statements, repeat, For-loop
2	10	For-loop
3	13	And & Or statements

Table 2: The different assignments and associated concepts

Assignment 1

The first assignment asks the students to print the total score of eight thrown dice from the game Pick-omino. In this game, the numbers one to five are on each die, including a 'rainworm', which equals five points. The first assignment can be solved in different ways, using different programming concepts, such as a for-loop, or just if-statements. This freedom in solving the problem aims to help the students get to a good solution.

Assignment 2

Since loops and the for-loop in particular are important concepts in programming languages, the second assignment revolves around this concept. This assignment asks the students to take an order in a restaurant for three courses, ask what the customer would like to drink and print the price of the full dinner. There are no restrictions on the prices and what is on the menu.

Assignment 3

The last assignment is about 'and' and 'or' statements. Since the students did not have a lecture about this concept before the experiment, they received a short introduction to the concept from us. For this assignment, the students ask the user for their favourite (and possible second favourite) football club. When the user answers 'Feyenoord' or 'Sparta', the students print 'You are probably from Rotterdam!'. When the user answers 'Ajax' and 'AZ', the students print 'You are having a hard time picking!'. Note that since this assignment is given in Dutch, the statements to be printed have been translated here from Dutch to English.

3.4 Data processing

Data collection

The data was collected in four sessions, spread over two weeks. While the students were working on the assignments, their screen and audio was recorded. Observations will be extracted from both what happened on the screen and what students discussed during the assignments. To transform the observations into clear behaviours, we use an operationalisation of the LOA model, based on the model used by Faber et al. [FKW+19]. We have adjusted the expected behaviour section of their model to represent behaviour occurring while using a textual language, instead of a physical robot. This model is shown in Table 3. The behaviours stated in this model were determined before the data collection took place. In combination with additional unexpected behaviours, this model gives a complete view on the possible observed behaviours during these assignments.

Data analysis

The method used for the analysis of behaviour of the students is based on the method used by Faber et al. [FKW+19]. The constructed model contains a definition of each layer and behaviour that we expect to see by the students per layer. With the use of this model, the observed behaviours were encoded to the designated layer of abstraction. The observed behaviours will be presented over an interval of ten seconds. A fifth layer will be added for behaviour that does not correspond with any of the layers of the model.

Helping students has been listed in different ways, depending on the nature of the problem. Helping students with small errors in their code was listed as behaviour in the code layer, since it involved a discussion of the code. When the students were given a hint to get to the right solution, this was listed as behaviour in the design layer, since this usually involved discussing the steps to take to get to a solution. Other issues, such as issues with the laptop or Hedy were listed as unrelated.

To be able to make a good comparison of the abstraction between the different levels of Hedy between the different pairs of students, we make a quantification of the observed behaviours. For every pair in every assignment, we determined the frequency of each layer of abstraction and how the students switched between the layers. This method results in line graphs, showing the time on the x-axis and the different layers on the y-axis. The full analysis of the collected data and creation of the graphs will be done using Microsoft Excel.

Layer	Layer definition	Expected behaviour
Problem layer	Behaviour related to understanding the exercise.	- Students discuss the starting point of the exercise. - Students discuss the goal of the exercise. - Students read the description of the exercise. - Other
Design layer	Behaviour related to designing a solution for the exercise and expressing this in human language, without actual code.	- Students discuss the programming concept(s) they need to use. - Students discuss what steps to take for the solution. - Students look into previous learning materials. - Other
Code layer	Behaviour related to translating the design of the solution to actual code.	- Write code. - Read code. - Discuss code. - Remove code. - Other
Execution layer	Behaviour related to the output of the code.	- Run the code. - Observe the output of the code. - Relate the outcome of the code to what the students expected. - Predict the outcome of the code, discuss what the outcome of the code will be. - Other.

Table 3: Layers of abstraction model

4 Results

The results of this research will be discussed divided over the different layers of the abstraction model. These sections end with a paragraph comparing the different assignments. For all figures discussed here, the numbers on the y-axis represent the different layers of the layers of abstraction model. That is, one is the problem layer, two is the design layer, three is the code layer, four is the execution layer and five is the extra layer for unrelated behaviour. As stated earlier, the behaviour is analysed in a time interval of 10 seconds.

4.1 Problem layer

Students discuss the starting point of the exercise. This behaviour barely occurred during the assignments. When this behaviour was observed, it was in the beginning of the recording, when the students were discussing the assignment.
Students discuss the goal of the exercise. The students usually demonstrated this behaviour to agree on what was expected from them in the assignment. Therefore it is no surprise that this behaviour was mostly observed within the first couple of minutes. An example of this can be seen in Figure 1. Pair one had some struggles with understanding what was required for this assignment, so they visited the problem layer multiple times in the beginning of the recording.
Students read the description of the exercise. Reading what an assignment expects from you as a student is a necessary step. This explains why this behaviour was observed in every recording and mostly at the start. Occasionally this behaviour was also observed later on in the recording, due to the students checking what the assignment or the statement to be printed was exactly.

The problem layer is about understanding the assignment, so it is understandable that behaviour corresponding to this layer is mostly observed in the early stages of the recordings. What is also remarkable, the problem layer is by far the least occurring layer out of the core layers (excluding the unrelated layer in the graphs). In most cases when the problem layer was visited, the students instantly switched to the code layer, which is unexpected.

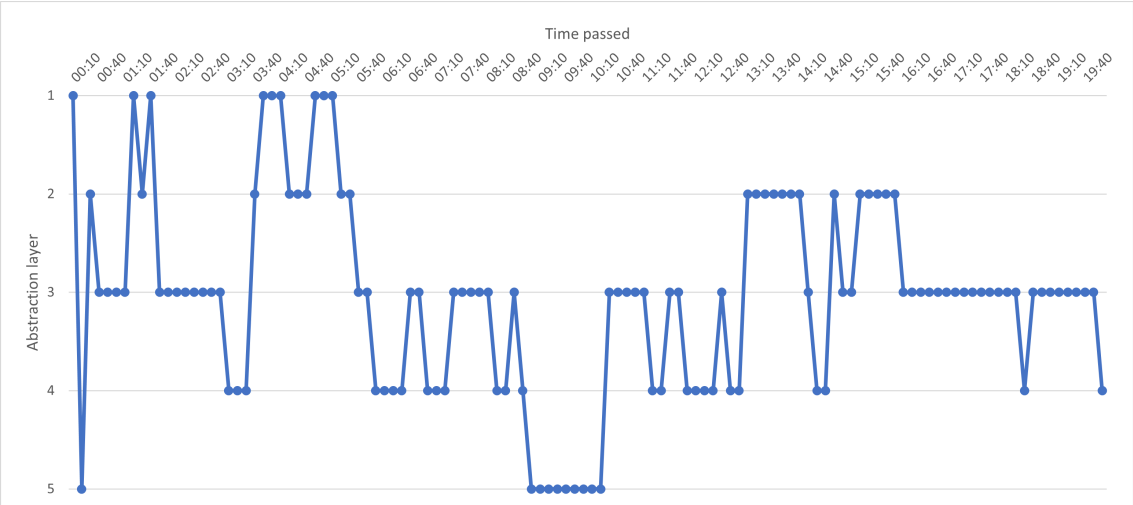

Figure 1: Line graph describing the behaviour of pair one during assignment 1.

4.2 Design layer

Students discuss the programming concept(s) they need to use. Frequently when design behaviour was observed, the discussion revolved around potential next steps. This usually included a general approach, where specific programming concepts were rarely discussed.
Students discuss what steps to take for the solution. Discussing what to do is important if you work in pairs and this behaviour was observed the most out of the options in this layer. Though there are small differences, all pairs of students frequently discussed the steps to take. In most cases, after discussing these steps a successful block of coding behaviour occurred.
Students look into previous learning materials. Previous learning materials can help understand what is needed for a specific assignment. While all pairs used this tool to their advantage in assignment 1, this behaviour barely occurred for assignment 2 and 3.

Something that stands out here is that behaviour related to design usually occurred over a longer period of time, instead of students visiting design frequently. In most cases this was due to the students thoroughly checking previous materials or discussing what to do next. The students actually took the time they needed before moving on with the assignment. When comparing assignment 1 to assignment 2, it showed that the students demonstrated more behaviour related to discussing what steps to take for the solution in assignment 2 than in assignment 1. In-between pairs, there was also a difference in usage of the design layer. As an example, comparing Figure 2 and Figure 3, we find that pair two visited the design layer more frequently than pair three, but when pair three arrived in the design layer, they usually stayed there for a longer period of time.

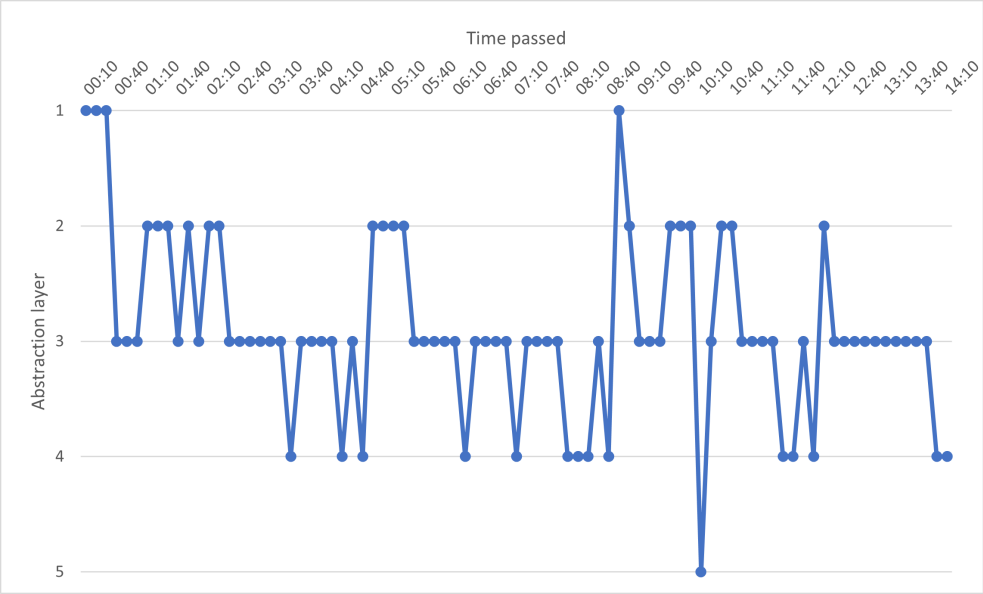

Figure 2: Line graph describing the behaviour of pair two during assignment 1.

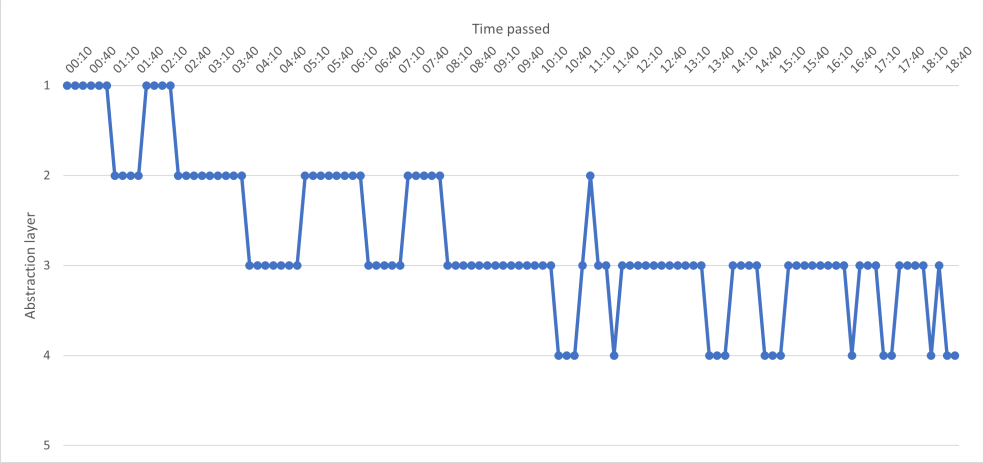

Figure 3: Line graph describing the behaviour of pair three during assignment 1.

4.3 Code layer

The code layer is the most observed layer in this research. The obvious reason for this is the fact that in this layer the actual solution of the assignment is created. Because of this, the code layer is the connection between the design and execution layer. Figure 4 shows pair one being 'stuck' in this layer. These students struggled heavily with understanding the concept of indentation. In some situations, they had not enough indentation, in other cases they indented the wrong code. The error messages they received because of these reasons did not push them in the right direction, often resulting in the students trying eight or even twelve spaces of indentation, instead of the regular four. In Figure 1 pair one also visited the code layer frequently to try and fix their code. The problem here was related to the students using the repeat command, which they were taught earlier in the course, instead of a for-loop. Hedy did not accept the repeat command in the current level of the assignment.

Write code. Writing code was the core behaviour of the assignments. Whether it was to write new code or alter existing code, this behaviour occurred very frequent.
Read code. Reading the code usually happened in combination with other behaviour, such as discussing the code. Because of this, reading code was rarely noted as independently occurring behaviour.
Discuss code. All pairs discussed the code frequently, but pair two and three stood out. These two pairs discussed their code many times during assignment 2. Their behaviour is shown in Figure 5 and Figure 6.
Remove code. In most cases, there was no need for the students to remove big chunks of code. Usually, the code needed a few modifications, rather than a complete replacement. This resulted in only a few observations during the assignments.

4.4 Execution layer

Run the code. This behaviour was mostly noted when the only observable behaviour that happened in a time interval was the execution of the code. It is very interesting that not all students

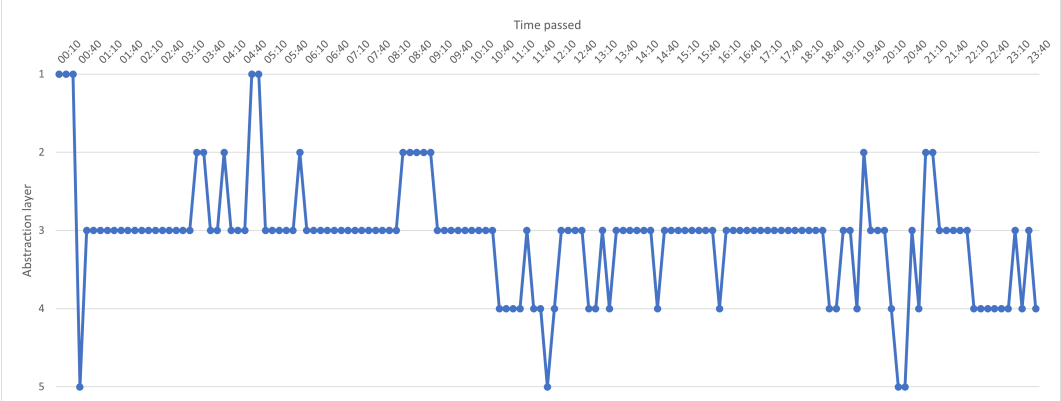

Figure 4: Line graph describing the behaviour of pair one during assignment 2.

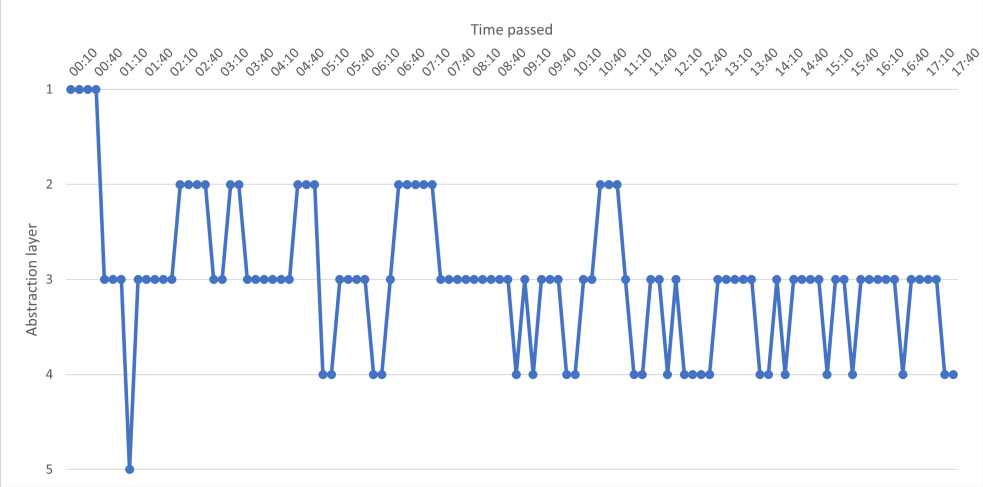

Figure 5: Line graph describing the behaviour of pair two during assignment 2.

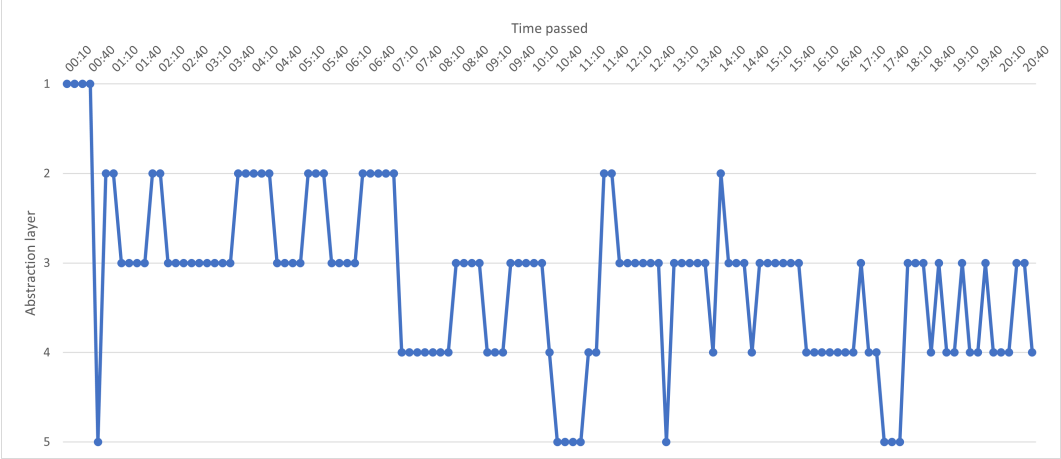

Figure 6: Line graph describing the behaviour of pair three during assignment 2.

showed this behaviour for the same reasons. For example, Figure 5 shows a fast alternation of the code layer and the execution layer. This is because pair two used the execution option as a tool to check whether they fixed the errors in their code. Now, when observing the data in Figure 7, we notice only a couple of visits to the execution layer. This is mainly due to these students only running the code when they were certain that the piece of code they created seemed good and working code. When there were errors, pair four usually switched to the design layer, to discuss what they need to do differently.

Observe the output of the code. This behaviour was mostly observed when students were giving inputs to their program (via 'ask' commands in the code) or looking at the output that was printed. Observing the output of the code helped many pairs in fixing grammar mistakes in their print statements.

Relate the outcome of the code to what the students expected. Verbal responses to the output of the code were a key part of observed behaviour here. Statements like "Huh" and "How is this possible?" show that this behaviour was usually demonstrated when the output of the code was different than the students expected. Most of the times, this behaviour was followed up by switch to the code layer, to fix the errors that were made in the code.

Predict the outcome of the code, discuss what the outcome of the code will be. Even though discussing what the code will possibly produce seems a valuable piece of discussion, this behaviour was not observed a single time. In many cases, one of the students in a pair would just press the button to execute the code without discussing with their peer. When something was said by a student prior to executing the code, it was usually a quote such as "Let's see whether this works".

Some students use the option to execute the code significantly later than others. While some pairs already execute code after three or four minutes, for example in Figure 2, other pairs only execute the code for the first time after ten or eleven minutes (see Figure 4).

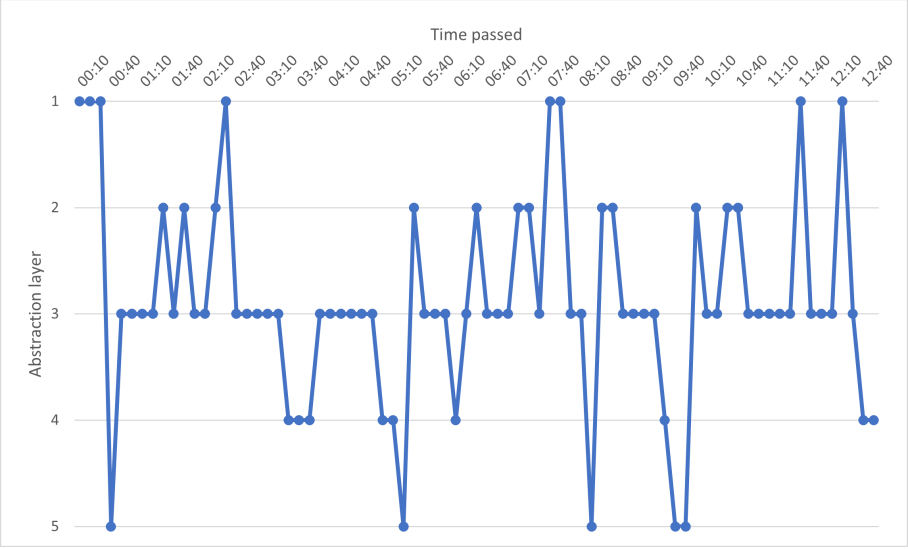

Figure 7: Line graph describing the behaviour of pair four during assignment 3.

4.5 Other observations

There are also some interesting findings in the results of this research that are not bound to a certain layer. In later levels of Hedy, some previously taught commands are replaced by more professional ones. An example of this is the replacement of the repeat command by the for-loop concept. Not all students realised this, resulting in some pairs being stuck with errors in their code. Figure 1 shows this issue around nine minutes. Here the students needed some assistance from the researcher, in order to continue the assignment. This is a good example of students suffering from prior knowledge.

The design layer seems like a necessary step to take from a theoretical perspective, in order to solve the assignment at hand successfully. It is interesting to see that not all pairs used this layer as frequent as others. It seems like the pairs that visited the design layer more frequently were able to make progression faster than the pairs that did not.

4.6 Discussion

Problem layer

The problem layer was mostly observed in the beginning of the assignments. A reason for this could be that the assignments were easy to understand, which takes away the need to discuss what is expected from the students. Looking at the research of Faber et al. [FKW+19], these findings correspond with their results. In their research, the children only visited the problem layer once every assignment. The fact that the problem layer is the least occurring layer in this research not only corresponds with Faber et al. [FKW+19], but is also observed in the research of Bolt [Bol21].

In the majority of the recordings, students switched from the problem layer to the code layer, instead of visiting the design layer first. This does not match exactly with related work, but this research also registered more behaviour in the problem layer than related work did. Since the assignments seem easy to understand, students might know where to start immediately after reading the assignment, which might explain why students switch to the code layer after the problem layer.

Design layer

The pairs used the design layer for different reasons. Where one pair used it more to look in to previous learning materials, some students used it more to discuss what steps to take for the solution. This results in varying use of the design layer in the figures, but also explains why some students, such as pair three, stayed in the design layer longer. The students also discussed the steps for the solution more in assignment 2 than in assignment 1. A reason for this could be the level and nature of the assignments. Assignment 1 was more like a general task to solve, where as assignment 2 was more story based and required more print statements and input from the user.

The students regularly switched back and forth between the code and the design layer. Various reasons could apply here. When students find mistakes in their code, they might return to the design layer to discuss what to do or look into previous materials. Students will often switch from the design layer to the code layer when they think they have found a solution for the assignment.

This continuous switch between these two layers is also found by Faber et al. [FKW+19] and Bolt [Bol21].

Code layer

The code layer was the most observed layer in this research. While this matches roughly with related work [FKW+19][Bol21], the students in this research spent even more time in this layer in this research. This could be due to Hedy being a textual programming language, where as other studies have used different tools, such as robots. Another thing that matches with these studies is the regular switch between the execution layer and the code layer. After finding errors in the code by executing it, students usually switched to the code layer to fix the errors. This shows behaviour related to debugging.

Execution layer

Behaviour related to discussing what the output of the code will be is the only behaviour stated in Table 3 that has not been observed during the assignments. This could be due to the computational complexity of the assignments. For example, solving an algorithmic problem using recursion is a computationally demanding task. The students working on such an assignment will probably discuss potential outputs more often than the students needed for the assignments in this research. This might also explain why the design layer is not used as often as one might expect beforehand.

Other observations

Table 3 was created before the data collection, so this model has not been influenced by any of the results found in this research. It is good to notice that no behaviour that was not included in this model has been observed during the assignments.

In general, all pairs switch frequently between the different layers. It rarely happened that a pair stayed in the same layer for longer than 2 minutes. This rapid alternating of behaviour matches with Faber et al. [FKW+19] and Bolt [Bol21], despite using a significantly different tool and group of participants.

Pair one had more difficulties with the understanding of programming concepts than the other pairs. In assignment 2, they struggled significantly with fixing the errors regarding indentation and in assignment 1 they did not use the concepts that they were taught latest (for-loop instead of repeat command). As stated in Table 1, this pair did not have any experience with programming so far, so this course on Hedy was their first encounter with programming. This could explain their difficulties in understanding the concepts they struggled with.

Hedy specific observations

Some observations found in this research are related to the contents of Hedy and will probably not be observed in other languages. One of these observations is the difficulty students had with the repeat command and for-loops. The repeat command is not used in most professional languages, but it is used in Hedy to introduce students to for-loops. The difficulties students had with this

could happen because they do not see that the for-loop is a substitute for the repeat command. They might just use repeat since that is what they are used to.

Another concept specific to Hedy is the additional example code provided in every level. When students demonstrated behaviour related to looking into previous materials, this was always to view the example code that was available. Since this is not available in other languages, students might show different behaviour instead of the behaviour described here.

4.7 Limitations

There were some limitations during this research. This research only has a limited number of participants, due to time constraints and not all parents giving permission for their child to participate in research. This resulted in a smaller group of students from whom data could be collected. Secondly, due to time constraints, some students could not finish the last details of their assignment. Though this might influence the results, it has been ensured that all students at least finished the most important parts of the assignment. Another limitation is the varying degree of programming experience, prior to starting with Hedy. Since some pairs had no prior experience and some had experience with Scratch, this difference could influence the results of this research. Associated with this is the fact that all students were already familiar with Hedy up until level 10. This will likely give different results than examining students that have never worked with Hedy before. The last limitation is the limited number of assignments in this research. It is difficult to completely cover all the content of Hedy in three assignments, so more assignments are necessary to make a more extensive comparison.

5 Conclusions and Further Research

This research was conducted to find answers to the question how children navigate the layers of abstraction while working on programming exercises in Hedy. Children rapidly switched between the different layers of the abstraction model. However, the students regularly stayed in a certain layer when this was necessary. As an example, multiple pairs took their time to discuss the next step of the solution or look into previous learning materials. While the problem layer was mostly observed in the beginning of the exercise, the execution layer occurred everywhere, sometimes early in the assignment, sometimes relatively late. Students spent the most time in the code layer, where the solution is transformed into code. Usually when the students ran their code, a visit to the code layer followed to fix errors in the code. Some pairs had more trouble with the correct use of some of the programming concepts than others, but in the end all students reached a decent solution. Hedy appears to be a good and promising programming tool to help children learn programming. In this way, children get familiar with the skills and concepts of programming that are necessary in professional programming languages.

There is still plenty of room for future research to build upon this research. While this research gives insights in how children work on programming assignments in Hedy, it would be interesting to make an extensive comparison with children working on assignments in professional programming languages. This will create the opportunity to analyse possible differences between working with a

tool like Hedy and working directly with a language like Python or Java. Since the operationalisation of the LOA model, used in this research, captured all observed behaviour, this model can be used in future work to make the comparison with other languages.

Another concept that would be interesting to examine in combination with this research is the positive or negative transfer of prior knowledge while working in Hedy. In this research, only a couple of clear situations of a negative transfer of prior knowledge were observed, so it would be useful to research this more thoroughly. To improve this research further, more participants could be selected for future research to extensively investigate the use of computational thinking by children in Hedy.

References

[AH16] Efthimia Aivaloglou and Felienne Hermans. How kids code and how we know: An exploratory study on the scratch repository. In *Proceedings of the 2016 ACM conference on international computing education research*, pages 53–61, 2016.

[BFKS14] Marina Umaschi Bers, Louise Flannery, Elizabeth R Kazakoff, and Amanda Sullivan. Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, 72:145–157, 2014.

[BKMT94] P. Brusilovsky, A. Kouchnirenko, P. Miller, and I. Tomek. Teaching programming to novices: A review of approaches and tools. 1994. Educational Multimedia and Hypermedia, 1994. Proceedings of ED-MEDIA 94–World Conference on Educational Multimedia and Hypermedia (Vancouver, British Columbia, Canada, June 25-30, 1994).

[Bol21] Julia Bolt. Blind leren programmeren: Een studie naar de computational practices van kinderen met een visuele beperking bij het programmeren van muziek in sonic pi, January 2021. BSc thesis University of Leiden.

[D+89] Edsger W Dijkstra et al. On the cruelty of really teaching computing science. *Communications of the ACM*, 32(12):1398–1404, 1989.

[DA12] SRM Derus and AZ Mohamad Ali. Difficulties in learning programming: Views of students. In *1st International Conference on Current Issues in Education (ICCIE 2012)*, pages 74–79, 2012.

[FGM13] Georgios Fessakis, Evangelia Gouli, and Elisavet Mavroudi. Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, 63:87–97, 2013.

[FKW+19] Hylke H. Faber, Josina I. Koning, Menno D. M. Wierdsma, Henderien W. Steenbeek, and Erik Barendsen. Observing abstraction in young children solving algorithmic tasks. In Sergei N. Pozdniakov and Valentina Dagienė, editors, *Informatics in Schools. New Ideas in School Informatics*, pages 95–106, Cham, 2019. Springer International Publishing.

[GM07] Anabela Gomes and Antonio Mendes. Learning to program - difficulties and solutions. In *Conference: International Conference on Engineering Education – ICEE 2007*, pages 283–287, 01 2007.

[Her20] Felienne Hermans. Hedy: a gradual language for programming education. In *Proceedings of the 2020 ACM conference on international computing education research*, pages 259–270, 2020.

[Hua16] Hong Huang. The incremental teaching project design for project-based learning and its application in java programming course. *Online Submission*, 4(6):191–197, 2016.

[HWBC77] Richard C. Holt, David B. Wortman, David T Barnard, and James R. Cordy. Sp/k: a system for teaching computer programming. *Communications of the ACM*, 20(5):301–309, 1977.

[Jen02] Tony Jenkins. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, volume 4, pages 53–58. Citeseer, 2002.

[KG14] Filiz Kalelioglu and Yasemin Gülbahar. The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education*, 13(1):33–50, 2014.

[MM12] Fernando ORNELAS Marques and Maria Teresa Marques. No problem? no research, little learning... big problem! *Systemics, Cybernetics and Informatics*, 10(3):60–62, 2012.

[PGK05] Jacob Perrenet, Jan Friso Groote, and Eric Kaasenbrood. Exploring students' understanding of the concept of algorithm: levels of abstraction. *ACM SIGCSE Bulletin*, 37(3):64–68, 2005.

[PK06] Jacob Perrenet and Eric Kaasenbrood. Levels of abstraction in students' understanding of the concept of algorithm: the qualitative perspective. *ACM SIGCSE Bulletin*, 38(3):270–274, 2006.

[PT12] Yannis Papadopoulos and Stergios Tegos. Using microworlds to introduce programming to novices. In *2012 16th Panhellenic Conference on Informatics*, pages 180–185. IEEE, 2012.

[QL17] Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, 18(1):1–24, 2017.

[RMMH+09] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[RRR03] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer science education*, 13(2):137–172, 2003.

[SA16] David Statter and Michal Armoni. Teaching abstract thinking in introduction to computer science for 7th graders. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*, pages 80–83, 2016.

[SF13] Linda Seiter and Brendan Foreman. Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of the ninth annual international ACM conference on International computing education research*, pages 59–66, 2013.

[Str21] Amanda L Strawhacker. Computational thinking and life science: Thinking about the code of life. In *Teaching Computational Thinking and Coding to Young Children*, pages 107–133. IGI Global, 2021.

[SVSB20] Amanda Strawhacker, Clarissa Verish, Orit Shaer, and Marina Umaschi Bers. Designing with genes in early childhood: An exploratory user study of the tangible crispee technology. *International Journal of Child-Computer Interaction*, 26:100212, 2020.

[SW13] Cynthia Selby and John Woollard. Computational thinking: the developing definition. *Special Interest Group on Computer Science Education (SIGCSE)*, 2013.

[TMK85] Ivan Tomek, Tomasz Muldner, and Saleem Khan. Pms—a program to make learning pascal easier. *Computers & Education*, 9(4):205–211, 1985.

[WCM+18] Jane Lisa Waite, Paul Curzon, William Marsh, Sue Sentance, and Alex Hadwen-Bennett. Abstraction in action: K-5 teachers' uses of levels of abstraction, particularly the design level, in teaching programming. *International Journal of Computer Science Education in Schools*, 2(1):14–40, 2018.

[Win06] Jeannette M Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[Win08] Jeannette M Wing. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3717–3725, 2008.

[Win11] Jeanette Wing. Research notebook: Computational thinking—what and why. *The link magazine*, 6:20–23, 2011.