Impacts of Block-based Programming on Young Learners' Programming Skills and Attitudes in the Context of Smart Environments

Ph.D. thesis

by

Mazyar Seraj

Supervisor

Prof. Dr. Rolf Drechsler

Universität Bremen

Impacts of Block-based Programming on Young Learners' Programming Skills and Attitudes in the Context of Smart Environments

Ph.D. thesis

by

Mazyar Seraj

A dissertation submitted for the degree of Dr.-Ing.

Supervisory Committee

Prof. Dr. Rolf Drechsler (University of Bremen, Germany)

Prof. Dr. Ira Diethelm (University of Oldenburg, Germany)

April 2020

Acknowledgments

I would like to start by thanking my advisor Rolf Drechsler for his continuous support and guidance. I deeply appreciate the opportunity he gave me to be part of his research group, which enabled me to learn, research, and present my ideas about various interesting topics.

I take this opportunity to express gratitude to my co-authors, Serge Autexier, Eva-Sophie Katterfeldt, and Cornelia S. Große, who helped me with their valuable scientific knowledge, as well as insightful discussions and suggestions. This work would not have been possible without their help. I would also like to thank my committee members and my external examiner Ira Diethelm for their time and their constructive comments and suggestions. I greatly appreciate the help and support of my friends and colleagues in DFKI, AGRA, and SMILE project team members.

Finally, words can not express how grateful I am to all members of my beloved family for their continuous support. Special thanks to my parents Belgheys and Mansour for their constant encouragement during stressful times and endless patience with their busy son. Without them, this work would have no meaning.

Mazyar Seraj
April 2020

Disclaimer

I hereby declare that

- this dissertation has been composed by myself,
- no portion of this dissertation has been submitted for any other degree or professional qualification except as specified,
- it has been completed without claiming any illegitimate assistance, and
- I have acknowledged all sources used (both, verbatim and regarding their content).

Mazyar Seraj
April 2020

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Inexperienced and young learners typically have difficulties with respect to the programming experiences and activities. These difficulties are mainly due to these facts that first, learning and recalling code syntax is hard because it requires a high level of concentration for the targeted learners (syntactic knowledge). Second, assembling and manipulating code structures is error prone because it requires the learners to have high level of conceptual knowledge. Third, due to having lack of strategic knowledge among the learners, understanding the requirements of designing, executing and debugging computer programs is hard for them [BGK+17, QL17]. Considering the complexity of introductory programming for the learners, visual programming has become more and more popular [Wei19]. In particular, block-based educational programming systems have emerged as an area of active research. Block-based programming is introduced as a form of visual programming that reduces the syntactical errors by encapsulating the code into smaller code chunks and relying on recognition of blocks instead of remembering the code syntax. In block-based programming environments, blocks assist learners to assemble the code without basic errors for manipulation of code structure [BGK+17, KMA04]. In this respect, visual block-based programming environments have been widely used with a graphical interface to introduce young learners to general features of programming languages such as variables, data types, loops, conditional statements, functions, and operators. Moreover, these environments aim to enable the learners to author programs without having a high level of syntactic, conceptual and strategic knowledge.

As the visual block-based programming environments have become the standard medium of instruction in the design of introductory programming courses [WW17b, KLS+14, MGB15], they are employed by researchers and educators to enable young

learners to learn programming and author computer programs. However, in addition to these programming environments, an interesting and motivating context is needed to encourage the learners to start with programming activities. Scientific works emphasize that tangible and interactive objects benefit learning, especially for young learners [MCK17, MGB15, QBBD13, KLS+14, KDS09]. Tangible objects have been used to introduce computer programming to learners and the benefits of learning to code for tangible computers (e.g., robots and computational artifacts) has been explored. Moreover, countless block-based programming environments have been employed together with tangible objects in order to improve young learners' emotional engagement, attitudes, and their computer programming performance [Zim17, QBBD13, KDS09, MCK17]. Nevertheless, researchers and educators still face three issues while working with educational block-based programming environments:

(1) lack of block-based programming tools as hassle-free programming environments to be compatible with different tangible and smart devices and environments,

(2) lack of block-based programming environments to enable young learners to learn and author programs for real-world environments (e.g., real life-size smart homes and living labs), and

(3) lack of block-based programming environments to allow the learners to have a short time span between the development of ideas and their implementation in real life-size smart environments.

Inexperienced and young learners on the one hand want an easy way to customize ideas into the real world; on the other hand, they need support in order to implement their ideas. Thus, in this thesis, an educational block-based programming tool is proposed (which is built with the Blockly library [Fra14]), considering both aspects. This approach helps the learners to learn and author programs which are also applied into real life-size smart environments, mobile robots and micro-controllers. Furthermore, it helps to prevent syntax errors and errors when assembling and manipulating code structure. This environment is considered as a settlement between a pure programming Integrated Development Environment (IDE) and a simple interface designed for inexperienced and young learners, allowing them to learn and to achieve results quickly. Different features of Hypertext Preprocessor (PHP) and Arduino programming languages are included in this environment.

Irrespective of the concrete approach, the question arises how to support the targeted learners in interacting with educational block-based programming environments in the context of smart tangible objects and environments. From a psychological perspective, as introductory programming is difficult for the learners,

block-based programming environments is still not fully intuitive for them. Thus, two instructional approaches (in the form of supplementary documents) are promising to help them to work with blocks and solve programming problems, namely worked examples and instructional procedures. Worked examples have been introduced as a common support to teach learners how to solve programming and mathematical issues by presenting a solution [ZLP18, MGG+14]. A large number of empirical studies demonstrate beneficial effects of worked examples, for an overview see for example [ADRW00, SVMP98, SKR+10]. Another instructional approach is to present instructional procedures to the learners. However, although instructional procedures are widespread used, they often do not support learning optimally [WR08, MGG+14]. Zhi et al. [ZLP18] used text-based instructional procedures in a puzzle-based educational programming game called BOTS [HCB14] to solve programming problems. In this learning context, they found that presenting instructional procedures of the solution was less effective for learning than worked examples. Still, it remains an open question whether worked examples or instructional procedures are more beneficial for young learners as supplementary documents in the context of educational block-based programming, as well as smart tangible objects and environments.

Therefore, in this thesis, in addition to the development and evaluation of the block-based programming environment (which aim to facilitate programming for the learners), results of empirical evaluations are presented. These results are beneficial for educators and researchers to understand the benefits of supplementary documents for acquisition of programming skills and attitudes towards programming among young learners.

The main emphasis of introductory programming courses is to show the application of programming to young learners in reality, using block-based programming environments together with tangible and interactive objects [MCK17, PHEC17, MGB15]. Learning by constructing artifacts with physical computing material has roots in constructionist learning theory [Pap80]. Constructionism refers to educational practices that are learner-focused. Young learners learn about underlying abstract concepts by acting as designers of personal-meaningful artifacts. The artifacts become objects-to-think-with and constitute micro-worlds as a self-contained, constraint world inviting to explore its underlying assumptions. Constructionism has been applied in the design of introductory programming concepts to engage the learners with a focus on programming and computer science, in general [KDS09, KLS+14, MSS+13, Ben12]. In the 1990s, programmable bricks for education came up, which are linked to constructionist learning [RMB+98, B+15]. These micro-controller boards can be equipped with sensors and actuators, and be programmed. They enable young learners to design and build ubiquitous computing objects and explore concepts of programming and computer technology. Fur-

thermore, block-based programming environments for early access to coding (e.g., Scratch) have also emerged from constructionist learning approaches [RMMH+09].

Nowadays, physical computing technologies and programming environments with low barriers have been established. These technologies and environments have been used to explore computational concepts by building smart tangible objects in educational contexts. However, the application area of designing smart objects can be extended to the state-of-the-art area of smart environments. From a technical point of view, smart environments comprise networking, intelligent control and home automation of key electrical appliances and services [JLY04]. From a user perspective, a smart environment is a space equipped with smart technologies to enhance the quality of its inhabitants' life by providing services that control, monitor and support their well-being [MPA19]. Social relevance of computing has been identified as attracting, especially for women, to participate in computer science education and society [KLR16]. Therefore, the theme of smart environments offers high potential for a purposeful application in computer science education that is meaningful for diverse target groups of learners.

The application of constructionist approaches in computer science education has resulted in a growing use of tangible objects and block-based programming to enable young learners to start with programming activities [MCK17, MGB15, QBBD13]. In an effort to address this intervention, block-based programming environments are employed in previous research to teach the basic programming skills to young learners in the context of robotics [MCK17, PHEC17, MGB15], computational artifacts [QBBD13, KLS+14, KDS09], and doll houses [KD18, Str09]. Moreover, it is addressed that introducing young learners to modern technologies fosters their attitudes and perceptions towards programming and computer science, in general [MGB15, MCK17]. According to [MCK17], attitudes and perceptions of a person determine how s/he is likely to act in different situations such as learning computer programming. Therefore, providing a positive view towards computer programming is beneficial to increase the understanding of programming skills [MGB15, MCK17], and interest in computer science [CLKL14, MCK17] among inexperienced and young learners.

There is much to show on teaching programming via tangible objects and on the effectiveness of using these platforms in order to acquire new computer programming skills. However, less is known about how inexperienced and young learners' performance and attitude towards programming are influenced over time in the context of real life-size smart environments. Unfortunately, these environments are not accessible for young learners due to relying on modern and powerful technologies. Therefore, relatively little attention has been given to show potential for using educational block-based programming environments to make state-of-the-art smart

technologies accessible for the learners. In particular, there is lack of investigation on how:

- introducing young learners to real life-size smart environments as an interesting context for them to begin with programming activities,
- teaching basic programming concepts to them via block-based programming in the context of smart environments, and
- letting them apply their new gained programming skills in a personal-meaningful tangible object and make it smart in order to improve their programming performance and attitudes towards programming.

According to the discussion above, there is a need to fill the gap between the regular usage of block-based programming and improvement of young learners' programming performance and attitudes towards programming and computer science. In this context, this thesis shows that by using block-based programming along with real life-size smart homes (as an example for real life-size smart environments), we are able to not only introduce programming to young learners, but also to show computing applications in a meaningful way that considers aspects of everyday modern living. Young learners' trajectories of acquisition of programming skills and their attitudes towards programming in the context of smart objects and real life-size smart homes are also explored (see Figure 1.1). Thus, this thesis addresses this gap by answering the following two-part research question:

How do young learners' programming performance and attitude change over time in non-formal programming training sessions with respect to using block-based programming and smart homes as a medium of teaching programming?

Overview. This thesis explores educational block-based programming environments in the context of smart objects and environments to achieve two main objectives. First, exposing young learners to programming activities in order to help them to realize that computer programming can be presented in a way which is not necessarily difficult to understand. Second, utilizing the results of these programming activities to effectively develop the learners' basic programming skills and engage them in future learning computer programming. In this respect, a total of 108 German secondary school students (60 girls and 48 boys; ages 10–15) participated in six non-formal programming training sessions (from 2-hour to 4-day). The training sessions were conducted in 17 months, from April 2018 until August 2019. All the training sessions which are less than four hours have been held in one day. Those training sessions that are more than four hours are divided into two to four days and they were conducted in one week.

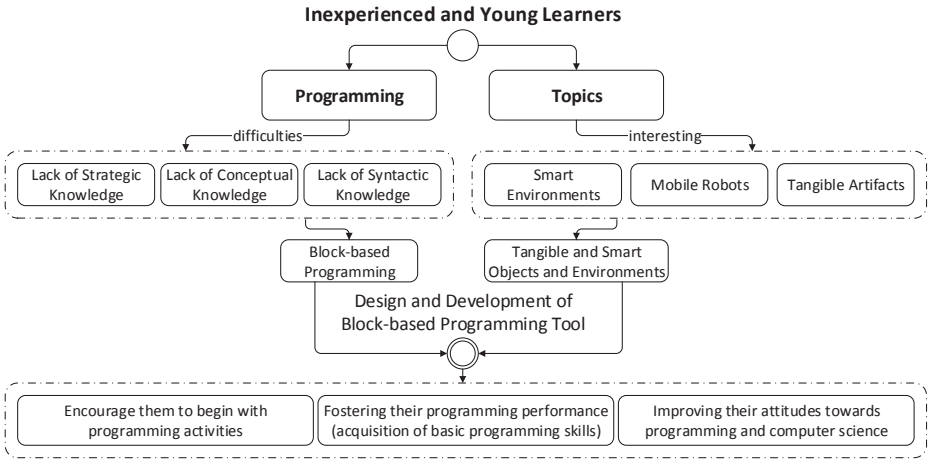


Figure 1.1: An overview of main contributions of this thesis.

The exposure of young learners to programming activities focuses on presenting an educational block-based programming tool that brings together the hot topic of smart environments and the visual programming paradigm. In order to illustrate the effectiveness of our approach, it is employed to design two one-day non-formal programming training sessions in context of smart homes, thereby comparing two supplementary documents to support learners, namely worked examples versus instructional procedures. The collected data and extracted information from them encourage us to focus more on young female students. Additionally, fewer than 1 in 5 computer science graduates are women across 35 European countries [Cor17]. Ertl et al. [ELP17] mentioned that approximately 25% of females are pursuing a career in STEM (Science, Technology, Engineering, Mathematics) in the EU, and this number is even lower in Germany with approximately 18% (where this thesis took place). In this respect, the proposed block-based programming environment is employed and compared to a similar and most used programming environment (namely mBlock [Mak19]) in other two one-day non-formal programming training sessions. This describes how girls' programming skills and attitudes influenced by different programming environments.

In the second phase of this thesis, to offer insights into the impacts of embedding the construction of smart objects in context of smart environments, we conducted longer period of training sessions (2- and 4-day). This helped us to explore young learners' programming performance and their attitudes towards programming over

time. Each training session was divided into two parts: (i) introduction to basic programming concepts, and (ii) implementation of these concepts on tangible objects and construction of a smart object in the context of smart homes. In this respect, programming tasks become more diverse and complicated for both groups of students (with and without prior programming experience), and both genders (boys and girls). In this phase, we focus more to find out the learners' trajectories of (i) attitudes towards programming and computer science (in terms of confidence, interest, and enjoyment), (ii) acquisition of programming skills, and (iii) programming experience (in terms of ease-of-use, ease-of-learning, usefulness and satisfaction), using the block-based programming environment and tangible interactive objects.

Outline. This thesis consists of nine chapters, including the current introductory chapter. Chapter 2 presents the state of the art and necessary background in the thesis. This chapter consists of an introduction to visual and block-based programming, as well as the related works in the programming of tangible artifacts, mobile robots, and smart homes. Chapter 3, 4, 5, 6, 7 and 8 present the main contributions on this thesis. Chapter 3 presents the proposed educational block-based programming tool as a novel approach to enable inexperienced and young learners to learn basic programming concepts. In this chapter, we illustrate how this approach provides a hassle-free environment for educators and researchers that aims to enable young learners to start with programming activities. It is also indicated that how the programming tool can support learners to rapidly prototype and program smart environments, mobile robots, and micro-controllers one at a time and in combination with each other. Chapter 4 provides a detailed discussion of designing the survey and programming questions. The information about survey instruments (questionnaires) and programming questions (tests) is summarized in this chapter. We also explain why the questionnaires and tests contained a set of specific questions, how they relate to the literature, and why they changed during our studies. The other chapters indicate the usage of the block-based programming environments together with smart objects and environments in order to show the learners' trajectories of attitudes towards programming and programming performance. These chapters are briefly described in the following.

- **Chapter 5** presents an application of using block-based programming in the context of smart homes in order to boost up young learners' programming skills and increase their interest in this topic. Two instructional interventions to support learners, namely worked examples and instructional procedures, are compared. The results do not strongly support one of these instructional interventions. Thus, both seem to be appropriate in order to help learners to work with the programming environment and acquire basic programming

skills. Moreover, this chapter offers first insights about the tight connection of non-formal programming training sessions to a concrete real-world scenario.

- **Chapter 6** presents the comparison of the designed block-based programming tool (which is based on Google Blockly) with an industrial, widely used block-based programming environment, namely mBlock (which is based on Scratch). Scratch and Google Blockly have been chosen as they are the most popular block-based programming editors in current education use of block-based programming. These editors are mainly used in order to foster young learners' programming skills and improve their attitudes towards programming.

- **Chapter 7** presents an application for creating smart everyday objects together with block-based programming. This shows how to leverage young female students' interest in programming, and, at the same time, supports the acquisition of programming skills. We designed and implemented a 4-day non-formal programming workshop to introduce the students to basic programming concepts based on block-based programming. They are also enabled to implement these concepts in a real object to make it smart in the context of smart homes. Learners' trajectories of performance and attitudes towards programming was evaluated based on repeated programming questions and qualitative open-ended questionnaires throughout the workshop.

- **Chapter 8** presents an important aspect of introducing programming to young learners via block-based programming, which is the application of programming in reality. The impacts of a real life-size smart home along with block-based programming on young learners' programming performance and attitudes has been investigated during a 2-day non-formal programming workshop.

The content of this thesis can be found in the following five publications, which have been published in international conferences in the field of Computer Science Education (CSE) and Human-Computer Interaction (HCI):

[1] **Impacts of Creating Smart Everyday Objects on Young Female Students' Programming Skills and Attitudes**
Mazyar Seraj, Eva-Sophie Katterfeldt, Serge Autexier, Rolf Drechsler
Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE), pp. 1234-1240, 2020.

[2] **Look What I Can Do: Acquisition of Programming Skills in the Context of Living Labs**
Mazyar Seraj, Cornelia S. Große, Serge Autexier, Rolf Drechsler

Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), pp. 197-207, 2019.

[3] **BEESM, a Block-based Educational Programming Tool for End Users**
Mazyar Seraj, Serge Autexier, Jan Janssen
Proceedings of the 10th Nordic Conference on Human-Computer Interaction (NordiCHI), pp. 886-891, 2018.

[4] **Scratch and Google Blockly: How Girls' Programming Skills and Attitudes are Influenced**
Mazyar Seraj, Eva-Sophie Katterfeldt, Kerstin Bub, Serge Autexier, Rolf Drechsler
Proceedings of the 19th Koli Calling International Conference on Computing Education Research (Koli Calling), pp. 1-10, 2019.

[5] **Smart Homes Programming: Development and Evaluation of an Educational Programming Application for Young Learners**
Mazyar Seraj, Cornelia S. Große, Serge Autexier, Rolf Drechsler
Proceedings of the 18th ACM International Conference on Interaction Design and Children (IDC), pp. 146-152, 2019.

[6] **Impacts of Block-based Programming on Young Learners' Programming Skills and Attitudes in the Context of Smart Environments**
Mazyar Seraj, Rolf Drechsler
Proceedings of the 25th ACM annual conference on Innovation and Technology in Computer Science Education (ITiCSE), to appear, 2020.

This thesis is concluded in Chapter 9, which includes the general limitations of it and a brief discussion on potential research avenues for future work in the area of visual programming and smart objects and environments.

Chapter 2

Background

In this chapter, first, research in the area of end-user programming is reviewed. Then, approaches using visual programming in educational tools and environments are introduced. In addition, an overview of block-based programming environments is provided as a form of visual programming. The main emphasis of this thesis is to reveal possibilities how the use of state-of-the-art smart technologies can improve young learners' programming performance, as well as develop a positive attitude towards programming and computer science among them. Thus, the next part of reviewing the literature is dedicated to the use of tangible and smart objects and environments in computing education. Finally, as the two areas of student assessment are their programming performance and attitudes towards programming, the definition of both learning outcomes and attitudinal aspects of learning programming is described.

2.1 End-User Programming

A large number of studies in the area of *end-user programming* (EUP) aim to enable inexperienced users [LPKW06,HC15,WAS+18] and young learners [ML18,PHEC17, Kit18] without practical experience and programming knowledge to write programs. In this respect, many techniques and design principles [RI06,KMA04,KM06, MKP+08] for EUP have been applied to develop end-user applications. One such technique is behavioral programming where all individual events are programmed as independent behavior threads which are interwoven at runtime [MWW12,HMW12]. In [MWW12,AMWW15], behavioral programming was implemented using JavaScript in Google Blockly [Fra14] for the user side web-based applications, and decentralized scenario oriented programming techniques were used. However, behavioral

programming does not provide a full expression of a programming language. For example, it would be difficult to change smart objects' behaviors dependent on specific events.

Other forms of end-user programming include Trigger-action programming which trades off expressivity for ease of use [UMPYHL14, HC15]. Trigger-action programming is used in do-it-yourself (DIY)-style smart home products [HH12, WL15], enabling inexperienced users to create smart home features by connecting various sensors (e.g., lighting sensors), actuators (e.g., door controllers), social network services, SMS, and email. Trigger-action programming is used to enable the users who are not familiar with professional programming to program a smart home, including "*if this then that*" (IFTTT) service [UMPYHL14]. This service specifies the behavior of smart objects as an event (trigger) and corresponding action to be executed when the event occurs (action). IFTTT service was used in LittleBits' electronics smart home kit to enable young learners to link their own devices or create their own linked scenarios in the context of smart homes [Kit18].

Programming by demonstration (PBD) is another form of end-user programming, in which inexperienced users are able to control home devices using a set of examples [DLY+06, DHB+04]. In the context of a smart home, using PBD enables users to create a situation and a desired behavior for home devices (associated action) without any programming knowledge [DHB+04]. These approaches allow inexperienced users to rapidly prototype and apply changes to different objects in smart homes, though, without learning general purposes and basic knowledge of programming.

2.2 Visual Programming

Visual (graphical) programming has been used to make programming problems easier to understand and solve. Young learners can create complex programs with little training in visual programming environments [Mye86, Mye90]. Visual programming is used in block-based programming editors such as Scratch [RMMH+09], Snap! [HM10], Alice [CDP00] and mBlock [Mak19]. Scratch and Alice are generally used to make animations, games and interactive applications. Snap! is an extended reimplementation of Scratch, letting young students build their own blocks. Using mBlock allows young students to program robots and microcontrollers. Furthermore, Pencil Code is a block-based coding tool developed based on Droplet [Bau15] to help young students work with JavaScript, CoffeeScript and HTML [BBDP15, WW17a]. Students are enabled to toggle between text code and blocks freely. This approach enhances familiarity with syntax while transferring from blocks to text code [BBDP15, WW17a, WH17]. Visual programming,

in particular Google Blockly, has been used as a client side of web-based programming environments in a number of educational and commercial applications and tools. MIT App Inventor [Inv18], ArduBlockly [PA19], CodeIt [PHEC17], CustomPrograms [HLC16], CoBlox [WAS+18], and MakeCode [Mak18] take advantage of Google Blockly in order to enable young learners and inexperienced users learning and making programs.

In contrast to a large part of this previous work, we seek to explore impacts of visual block-based programming environments with educational focus on inexperienced and young learners. We intend to support them in order to learn the general purpose of programming, as well as rapidly prototype and customize ideas in the context of smart devices and environments. Furthermore, we would like to introduce learners to new technologies which provide possibilities to tightly connect computer science to reality. In other words, we aim to introduce the future to young learners. In this respect, we took advantage of visual block-based programming to enable the learners to learn programming, and implement their ideas into the construction and control of real smart devices and environments in order to motivate them to take part in that future.

2.2.1 Block-based Programming Environments

Visual block-based programming has been used to enable inexperienced users and young learners to learn and author programs with little training [BLV+17, HLC16, KMA04, RI06]. In recent years, numerous block-based programming environments have been introduced for the learners to program on-screen animations [RMMH+09, MCK17], micro-controllers [Mak19, PA19, Sna19], and other programmable tangible objects [MCK17, MB11]. As such, these environments reduce the complexity of programming for the learners because they help them via using visual blocks to generate code syntax instead of memorizing it. This approach reduces syntax errors and eases the manipulation of code structures, and therefore, they are widely-used to teach programming, in particular, to young learners [BGK+17, KMA04, KP05, WW17a, QL17]. The blocks are shaped to assist users and learners to assemble the code without basic errors regarding manipulation of code structure [BGK+17]. For example, a string block can be plugged into a length block, but not into a logical operator block. In the background, the system translates block-code into code syntax, which is visible and editable in some of the environments (e.g., [Bau15, BBDP15]).

Nevertheless, there is a lack of block-based programming environments in educational contexts to support the programming of real-world state-of-the-art applications. Inexperienced and young learners who intend to customize programming ideas into tangible objects and life-size environments face two issues while working

with block-based educational programming tools. First, at least to the best of our knowledge, these tools have never been applied to real life-size smart environments such as smart home environments. Second, although the tool can in principle be applied to tangible and smart objects, programming, for instance, mobile robots or micro-controllers requires to work and become familiar with different tools, e.g., CustomPrograms, and ArduBlockly. In this thesis, we propose and employ an educational block-based programming environment which is built with the Blockly library to enable young learners to learn and author programs in the context of real life-size smart environments, micro-controllers and mobile robots.

2.3 Related Works

One important feature in learning programming, especially for young learners, is to enable them to understand how relevant programming and computer science are to their daily life [MCK17, BECC08]. Being rooted in constructionist learning theory [Pap80], programming tangible objects has a long history in education [RMB+98, B+15]. Meanwhile, countless programmable kits and computational textiles are on the market and have entered into educational institutions [Zim17, SWYM17,MCK17,BECC08]. With respect to constructionist learning theory, young learners can learn better when they design and construct interactive and tangible objects that are personally meaningful to them, such as computational textiles, robots, and interactive objects [MCK17, RMSS96]. Thus, researchers and educators designed introductory programming environments to support acquisition of programming skills through designing and creating visible and tangible objects. In both the Computer Science Education (CSE) and Computer-Human Interaction (CHI) research communities, several scientific studies tried to investigate various forms of smart devices (e.g., tangible artifacts [BEE06, GCNB18, QBBD13, KDS09], robots [MM19, PHEC17, MGB15, MSS+13], and smart homes [KD18]) in order to motivate young learners and show them how modern technologies relate to their daily life (see Figure 2.1). However, relatively little attention has been devoted to the potential of smart homes to support the claim, which they have a direct impact on young learners' performance and attitude towards programming and computer science.

2.3.1 Tangible Artifact Programming

The number of formal and non-formal programming courses and workshops that aim to introduce programming and computer science to young learners is growing. Increasingly, young learners start with programming activities via visual block-based

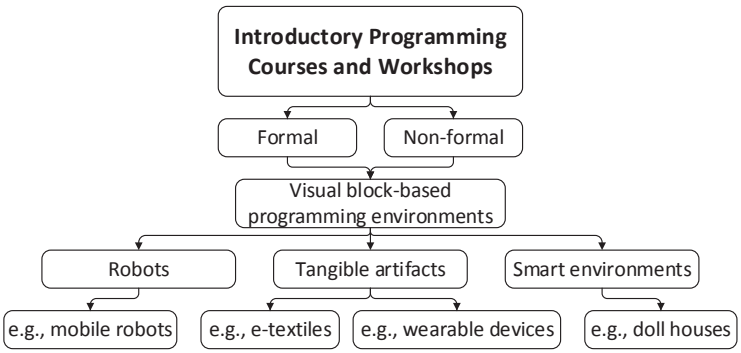


Figure 2.1: An overview of introductory programming courses and environments.

programming environments. These environments are widely used in the design of introductory programming courses and workshops [WW17b, WHHF18, MGB15]. Furthermore, in computer programming education, the application of computing in reality, tends to be shown to the learners via providing possibilities to experience programming for tangible interactive artifacts [MCK17, PHEC17, MGB15]. In addition to allowing them to learn the general purpose of programming and author programs via block-based programming environments, previous work provides possibilities to experience and implement new ideas into computational textiles [MCK17, KLS+14, QBBD13]. Computational textiles (e-textiles) such as wearable devices manage to merge the physical and virtual worlds and offer a tactual experience that complies with constructionist ideas [MCK17, QBBD13, KDS09, BECC08]. Findings show that young learners are able to learn better when they construct knowledge for a purpose that they found relevant for their lives. Additionally, the literature reports that teaching programming is more beneficial for young learners when they are engaged in designing and creating visible objects, such as wearable devices and computational textiles [MCK17, B+15, RMB+98]. With this regard, researchers and educators have developed introductory programming environments and pedagogical strategies. These environments and strategies have been employed in order to improve the learners' technological confidence and programming skills through playing with real-world environments or tangible objects. Tangible artifacts has been proposed as a framework for getting young learners involved within computer programming. Also, the benefits of learning to author programs for computational textiles has been explored in several recent scientific works [MCK17, KDS09, QBBD13]. E-textiles toolkits (e.g., LilyPad Arduino) presented in [BECC08] seem to be highly effective in motivating young learners to learn programming and shaping their attitudes towards programming and computer science positively.

Tangible Artifacts, in particular computational textile educational activities make use of soft material, and thus, introduce other forms of expression [MCK17, BECC08]. This form of expression historically has a more feminine orientation that attracts girls in computer programming. According to Katterfeldt et al. [KDS09], during computational textile activities, students gain more confidence in dealing with technology and they were able to link technology that they found relevant for their environment to their own created tangible object. According to [QBBD13, BEE07], using LilyPad and working with e-textiles can both increase programming knowledge and interest in working with electronics and computer programming among high school students. More specifically, research carried out with young learners indicated that in comparison to desktop and robot programming, learning computer programming with wearable devices would provoke more positive emotions to female students, and inspire them to acquire more programming skills [MCK17]. In addition, although prior technological experiences affect attitudes towards computing [Bei05], findings show that girls do not have as much confidence as boys with regard to technology and underestimate their ability to program [GC02]. With respect to technological confidence, results show that girls' comfort level increases with experience and they have benefited by visual programming environments [Sny14, MCK17, Bei05]. Nevertheless, research has not yet been conducted on the effectiveness of constructing smart objects together with block-based programming environments on young learners' programming skills and attitudes towards programming. This requires middle renege construction techniques as the male and female students can make use of both hard materials (e.g., sensors and actuators) and soft materials for designing and crafting techniques (e.g., colored papers and LED lights). Thus, it is important to evaluate computer programming in terms of student attitudes towards programming and their intention of learning programming in the context of smart tangible artifacts.

2.3.2 Mobile Robot Programming

Robots are one of the most common tangible and smart devices which are used for learning and educational purposes. According to [Mat04], robotics has the potential to influence engineering and science education at all levels, from K-12 (Kindergarten (K) and 1^{st} through 12^{th} grade) up to graduate school. The literature reports that robotic computing platforms can be used to engage young students to write code and author programs [MCK17, PHEC17, MGB15]. In particular, robotics platforms (e.g.,LEGO Mindstorms) has been widely used at schools for teaching basic programming concepts. Findings reported positive effects on high school students' interest besides the achieved educational goals [DSK05]. Merkouris et al. [MCK17] indicated that in comparison to desktop computing, male students who learn pro-

gramming with robotics would be more engaged, report more positive attitudes, and be able to develop their programming skills more effectively. Nevertheless, the variety of activities were limited to robotics platforms, but not a combination of robots with computational artifices or smart environments. Thus, the findings can not be generalized to a larger scale when the students are enabled to experience robotic programming in the context of real life-size smart environments.

Previous studies showed that through educational robotics young learners developed skills such as thinking skills, problem-solving skills, social interaction, and teamwork skills [Ben12, MCK17]. In addition, block-based programming environments have been effectively used to enable young students to learn programming. The findings show that they are able to write code, using these environments and the robots [KLS+14, QBBD13, MCK17, MGB15]. Paramasivam et al. [PHEC17] explored the use of block-based programming together with mobile robots as a framework for suited reflection in elementary school programming courses. In this respect, block-based programming is used to support K-12 students with disabilities to program Clearpath Turtlebot, capable of delivering items, interacting with people and autonomously navigating in its environment. Similarly, Martínez et al. [MGB15] enabled preschool and elementary school learners to program and control the behavior of Arduino boards in the context of N6 robots, using block-based programming environments. Furthermore, Przybylla and Romeike [PR14] presented creative learning environments for young students. They employed programmable kits (e.g., LEGO Mindstorms as a robotic toolkit) to offer a hands-on experience that can be used to develop a constructionist computer science curriculum with physical computing. It was indicated that the learners engaged more and showed higher intention of learning programming when they program the robots. Also, their findings showed that young learners' performance in learning basic programming concepts is improved when they are enabled to program the robots. Nourbakhsh et al. [NHCW04] found that by the end of a robotic course, girls' confidence increased more than the boys. However, they assumed that girls entered the course with less confidence and they struggled with programming more than boys. Sullivan and Bers [SB16] suggested that using robots to introduce programming to girls in their early childhood can foster interest and abilities in programming. Therefore, it is very important to evaluate computer programming in terms of students' perception and attitudes towards programming, as well as their intentions of learning programming in the future.

2.3.3 Smart Home Programming

Smart homes and living labs are another inspiring approach to motivate young learners to begin with programming activities and develop a greater interest in programming and computer science. Moreover, block-based programming as a tech-

nique to support young learners to learn programming is an active field of research. Bringing together the hot topic of smart homes and the block-based programming paradigm, researchers and educators introduced a research domain to engage the learners to get interested in programming and computer science, more broadly. The main goal of using the smart homes is to provide exposure to hands-on programming experiences in the programming training sessions [PR12, Str09]. For instance, Przybylla and Romeike [PR12] introduced physical computing via creative design-based approaches to learn about programming. Interactive objects such as magic flowers (which are related to smart homes) were designed and developed by young learners in the context of "*my interactive garden*", using Arduino technology and Scratch programming environment. Thus, young learners can both participate and experience new technologies which are adapted to technical equipments, as well as learn basic programming concepts in the context of smart homes using block-based programming environments. As many researchers and educators do not have access to real life-size smart homes, they mostly designed programming workshops that young learners can actively participate and create ideas related to smart homes. Then, the learners asked to construct a doll house and implement their ideas on it. In this respect, Strecker [Str09] tried to make programming accessible to a wider population of young learners via doll houses which are suitable for modeling a real smart home. Thus, the concept of "*interactive doll house*" was developed and implemented as an interactive learning environment for programming. The interactive garden and doll house scenarios aimed to use constructionist learning theory to support creative learning and provide an attractive experience- and practice-based learning environments for young learners. The major drawback of this approach is that the learners do not have the opportunity to author programs in a real life-size environment, and to become familiar with modern and powerful technologies. These technologies which are used by researchers enable young learners to see the connection of computer science with real life and learn about the future. In this respect, an important limitation of using real life-size smart homes is that young learners can only perform limited actions on them and they are not build for educational purposes. Thus, a combination of smart homes and mobile robots or micro-controllers could be helpful to enable the learners to perform variety of creative activities in the context of and within the real life-size smart homes.

2.4 Attitudes Towards programming

When introducing young learners to programming via block-based programming and smart environments, one of the main areas to assess is attitudinal and perceptual effects from using such environments. In order to understand how block-based

programming affects the learners' perceptions and attitudes towards programming in formal and non-formal learning environments, researchers employed attitudinal assessment [BBE+09, WW17b, MCK17]. As it is addressed by [BBE+09, WW17b], three dimensions of learners' attitudes should be taken into account (which are implemented in this thesis):

- **Confidence:** The first attitudinal dimension is whether young learners' confidence in programming can change via block-based programming and smart environments. In order to calculate a reliable measure of confidence, learners' responses to three categories are important: (i) their thoughts of being good at programming, (ii) their thoughts of doing well in the programming training sessions, and (iii) how much they find programming difficult to understand and the perceived difficulty of author programs.
- **Interest:** The second attitudinal dimension is whether young learners' interest in programming and computing differs based on the programming environment and the context that they applied their programming skills into it. In order to have a composite interest measure among the learners, two categories are important: (i) how much they are willing to enroll in future computer programming training sessions, and (ii) their interest in programming learning opportunities.
- **Enjoyment:** The third attitudinal dimension is to calculate a reliable measure of enjoyment in working with electronics and programming. The composite level of enjoyment can be calculated by the learners' responses to three main categories: (i) their thoughts of seeing programming as a fun subject to learn and perform, (ii) how much they like programming, (iii) how much they are excited about the programming training sessions.

2.5 Programming Performance

In computing education, several studies tended to introduce computer programming to young learners via block-based programming and measure learning outcomes [WW17b, MCK17, Lew10]. In this respect, students performance in learning programming is evaluated, focusing on three basic computational concepts, which are variables, loops, and conditional statements (conditionals and logical operators) [Lew10, PH07]. Development and evaluation of these concepts in introductory programming environments enable researchers to focus on the learning of:

- **Variables:** It is used to demonstrate how naming variables and different types of variables are important in programming. For instance, learners need

to know that naming variables is mandatory, so it is clear which one is being used at any time in the program, and thus, it is important to use meaningful names for variables. Also, learners should be able to differentiate between different type of variables as they come in all shapes and sizes. For example, some variables are used to store numbers (`numerical` variables), some are used to store text (`string` variables). Thus, variable concept is taught to young learners to show them how to store data and how to transfer data between different variables.

- **Loops and iterative logics:** It is used to demonstrate how to assemble and manipulate program structure by executing particular blocks of code. Loops and iterative logics, however, enable learners to make long programs short (e.g., `for` loop) and to iterate through a list of data (e.g., `foreach` loop).

- **Conditional statements:** It is used to demonstrate how to control the flow of execution by employing conditions such as `if` statements or `switch`. Moreover, differences between operators can be shown to the learners (e.g., greater vs. smaller, plus vs. minus, equal vs. not equal, etc.) in order to enable them to author a program that makes a decision based on multiple conditions.

2.6 Summary

In this chapter, first, an introduction to end-user programming and one of its frameworks which is visual block-based programming was presented. Second, the existing applications of block-based programming in the context of smart devices and environments were presented in detail. The results show that existing approaches have two major limitations in terms of having a precise extension of block-based programming environments, and evaluation of the young learners' programming performance and attitudes over time. The first limitation is that most of the environments can only be applicable for a restricted range of tangible objects and components (e.g., robots, micro-controllers and smart homes). This does not allow the learners to program different kinds of objects, using a single programming environment. The second limitation is that most of previous scientific works are limited to mobile and toy robots, doll houses, and e-textiles, but not to real-world environments. These approaches that work with tangible devices mostly evaluate the learners' performance and attitudes at the beginning and at the end of training (i.e., learners' trajectories of attitudes towards programming and performance are not indicated).

Therefore, we conclude that for tangible interactive objects together with real life-size smart environments currently no educational block-based programming environment is available. Thus, there is no visual programming environment that can help young learners to begin with programming activities (achieve results quickly), and support them to acquire basic programming skills (learn and author programs) in the context of smart environments. In order to tackle the aforementioned drawbacks in the following chapter of this thesis (Chapter 3), we present a new educational block-based programming tool. This tool is designed to help young learners to learn programming and to easily implement their programming ideas into the smart objects and environments. Moreover, concerning the learners' programming performance and attitudes, we present an experience-based approach to learning programming based on physical computing and constructionist learning theory. This includes both learning outcomes and attitudinal effects from using block-based programming in the context of smart environments. Hence, in addition to presenting a new educational block-based programming tool, this thesis contributes by showing the:

- usage of *block-based programming* is beneficial for *young learners* to be introduced to computer programming,
- applications of computing to young learners in a meaningful way that considers aspects of *everyday modern living*,
- improvement of young learners' *programming performance* (acquisition of programming skills) over time, and
- young learners' trajectories of *attitudes towards programming* in the context of *smart tangible object* and *real life-size smart homes* (as an example for real life-size smart environments).

Chapter 3

Design and Development of the Block-based Programming Tool

In the light of the complexity of introductory programming for inexperienced and young learners, visual programming has become more and more popular as a technique to support them to learn programming. In particular, educational block-based programming environments have emerged as an active field of research. Considering block-based programming in the context of smart environments, an educational block-based programming tool is required, enabling learners to learn and author programs in the context of smart objects and environments.

This chapter contributes the design and development of BEESM, a *Block-based End-user programming tool for SMart Environments*. The dedicated application domain engages learners to get interested in programming. Moreover, BEESM allows to learn the general purpose of programming and rapidly prototype and customize applications in the context of smart objects and environments. This approach enables learners to program smart environments, micro-controllers and mobile robots one at a time and in combination with each other. It also provides an educational block-based programming tool as a hassle-free environment for educators and researchers. Thus, they can make it compatible with different smart objects and environments for their formal and non-formal programming courses and workshops.

3.1 Introduction and Motivation

Inexperienced and young learners, the targeted users of BEESM, often have difficulties with respect to designing, integrating, compiling, executing, and debugging

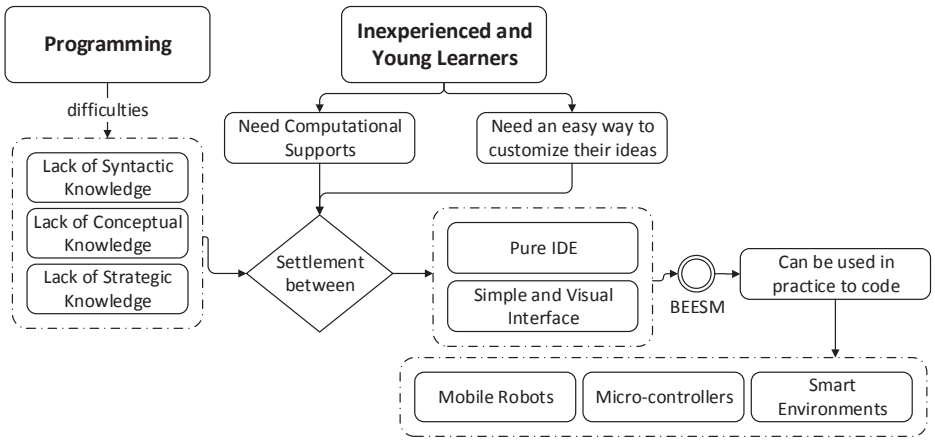


Figure 3.1: Design process of BEESM.

in introductory programming [GSH+18, QL17]. These difficulties experienced by the learners are related to (i) their syntactical knowledge (e.g., syntax errors), and (ii) their conceptual and strategic knowledge (e.g., errors when assembling and manipulating code structure) [QL17]. Educational programming tools generally either support learners to achieve results quickly, or introduce them to real programming development environments used by professionals [BBDP15]. However, we face the lack of a settlement between a pure programming development environment (e.g., IDEs), and a simple interface designed for the learners, allowing them to learn programming and to achieve results quickly (see Figure 3.1). In this respect, visual block-based programming environments are designed to allow the learners to learn programming, and overcome the obstacles of syntax and manipulation of code structure [BBDP15, Bau15, Com17]. Compared to the complexity of source code programming, visual programming has a great potential to facilitate programming for these learners [KAB+11, MKP+08]. Besides the block-based programming environments, the existence of a motivating context is necessary. In the present contribution, the context is given by smart objects (e.g., mobile robots and tangible artifacts) and environments (e.g., smart homes) that reflect the programming activities of the learners. This context perfectly matches the educational programming purpose and maintains the motivation of the learners as (i) they can easily see the consequences of their programming activities, and (ii) they can experience the latest technologies and learn about the future. Nevertheless, the targeted users

who are interested in having an easy way to customize their programming ideas into tangible objects and real-world environments face two issues while working with educational block-based programming tools. First, these tools are not applied to tangible objects and real-world environments. Second, although the tool can in principle be applied to the objects and environments, actually adapting it to, for instance, mobile robots, micro-controllers or smart environments requires to work and become familiar with many other tools [HC15, HLC16, KMA04, MGB15].

We designed and developed an educational block-based programming tool to program tangible objects and real-world environments, having both aforementioned issues in mind (see Figure 3.1) [1]. Having a tool like BEESM helps learners to have a short time span between the development of ideas and the transformation and integration into tangible objects and real-world environments. This should leverage their interest for programming and help them to acquire basic programming skills. Furthermore, learners have access to the standard programming language and can make modifications in BEESM. They are able to control the logic and flow of their programs. In this respect, this approach provides several opportunities for them: (i) creating, editing and running programs in tangible objects and real-world environments, and (ii) programming smart objects and environments which are also used by researchers, i.e., learners can actively participate and experience latest research efforts on a mobile robot or a smart home and learn about computer programming.

3.2 Overview of BEESM

BEESM is designed as a rapid educational block-based programming tool for educational purposes to help inexperienced and young learners to learn programming. It provides block-based programming in a web-based environment to program mobile robots, micro-controllers and smart environments. In this tool, in order to simplify programming for the learners, behaviors of smart objects are encapsulated in different functions and presented as basic programming primitives. BEESM can thus be used to program:

- *Smart Environments*: BEESM can be adapted and applied to smart environments if they support the web socket communication protocol [Fet18], and Remote Procedure Call (RPC) technology [BN84] (Smart-environment Controller).

[1] We make our tool BEESM and all source files available at https://github.com/projekt-smile/BEESM

- *Mobile Robots*: BEESM can be applied to any mobile robot with autonomous navigation running on Robot Operating System (ROS) [QCG+09].

- *Microcontrollers*: BEESM can be applied to any micro-controller running on Arduino Software by generating Arduino code for them.

In addition to learn programming, inexperienced and young learners are enabled to author programs in BEESM. It includes different programming language features like variables, conditionals, loops, predefined functions and operators based on the Blockly library. BEESM provides visual programming for the PHP programming language and Arduino code. It allows to program smart environments and mobile robots using PHP.

With respect to smart environments, BEESM is the first—to the best of our knowledge—educational block-based programming tool, which enables inexperienced and young learners to author programs in the context of real life-size smart environments.

Concerning the robot programming, it takes advantage of "*CodeIt!*" [Lab18], which is also used in [HLC16]. CodeIt! is a web-based programming tool based on JavaScript which is integrated with ROS for programming mobile and service robots (such as TurtleBot2, PR2, Fetch, and Savioke Relay robot). CodeIt! back-end provides a ROS action-library server that runs JavaScript programs to call ROS services. In doing so, they provide a sandboxed interpreter that the programs run through it. Other developers must define the robot primitives that the interpreter will run to call a ROS service for a robot. However, in BEESM we used PHP programming language to program robots and run functions that are understandable for ROS. This approach minimizes the code syntax (i.e., number of lines of code) and helps our target users to find programming errors and debug their programs when they encounter errors. In order to call ROS services on a mobile robot, a mini web-server that uses ROS functions is designed to upload the generated code to a connected mobile robot.

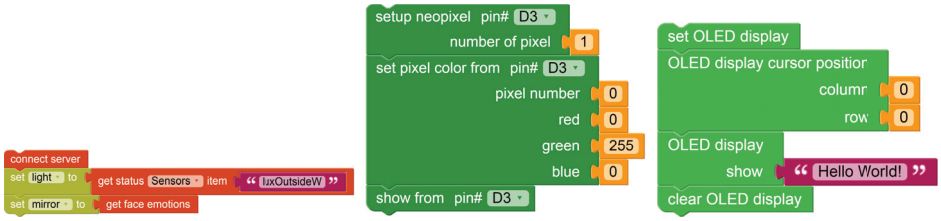
Concerning the Arduino programming, BEESM takes advantage of "*blocklyDuino*" [Lin19], which is also used in [MGB15]. BlocklyDuino is a web-based tool for Arduino programming. It provides a fully compatible Arduino source code, which enables users to program Arduino boards without manually pasting code to the Arduino IDE. To this end, a mini web-server that uses the Arduino IDE is running to upload the code to a connected Arduino board. Nevertheless, in addition to Grove sensors (light, sound, humidity, etc.), BEESM supports program DHT sensors (temperature, humidity, etc.) and AdafruitIO (which is a system that makes, for example, whether data are available to use). Users are also enabled to see the compiling and uploading process in BEESM. Errors and the output of the Arduino IDE Serial Monitor are shown to the users for error handling and debugging pur-

poses. Furthermore, in all Smart Environment, Mobile Robot and Micro-controller parts of BEESM, our target users are enabled to program not only the corresponding part but also program this part in a way to communicate with other parts. For instance, users are enabled to use Arduino code to program WeMos boards (with an on-chip Wi-fi Transceiver) to connect to either smart environment and mobile robot and send/receive data.

3.2.1 Primitives and Blocks

In previous research, capabilities of smart devices were organized into *Primitives*. CustomPrograms [HLC16] and CARMEN [MRT03] use this method to implement robots' behavior and capabilities into primitives. Each primitive robot behavior can be called through a function such as navigating the robot to a location. These primitives are used in order to design a visual block-based application which enables students with disabilities (e.g., deafness, muscular dystrophy, and attention deficit disorder) to program Clearpath Turtlebot, capable of delivering items and autonomously navigating in its environment [PHEC17]. BlocklyDuino which is used in [MGB15] takes advantage of this method to enable preschool and elementary school children to program and control the behavior of Arduino boards. Furthermore, smart environments are composed of a set of smart and controllable household appliances. Web socket communication protocol and RPC technology are fairly used as a Web-based interaction to control them. In this way, HTTP GET and POST requests that return JavaScript Object Notation (JSON) [Bra17] responses are used to communicate between user and Web server.

BEESM takes advantage of using primitives in order to enable inexperienced and young learners to program smart environments (e.g., smart homes), as well as mobile robots (e.g., TurtleBot) and micro-controllers (e.g., Arduino or WeMos boards) one at a time and in combination with each other. These primitives are defined as a set of custom blocks implemented in BEESM to allow learners to work with them and see the reactions of smart objects in real-time. In this way, apart from predefined blocks, a set of custom blocks (customized blocks) are provided for all primitives with inputs, outputs and types of connections. In a smart environment, the smart objects have a set of primitive behaviors, such as changeable status or being creatable by learners. Robot's behavior and capabilities were implemented into primitives. Each primitive can be called through a function such as navigating the robot to a location. These primitives generate PHP code syntax and enable learners to work with them in order to see the reaction of robots and smart devices. PHP programming language was chosen as it is a powerful server side scripting language to interact with web servers. It is also a widely-used, free and efficient programming language. Furthermore, primitive Arduino's behavior is wrapped in a set of blocks



(a) Smart Home (b) LED Lights (c) OLED Displays

Figure 3.2: Three examples of execution blocks for BEESM.

which generate Arduino code. Learners can then integrate these primitives into general purpose of programming languages to author Arduino programs.

3.2.2 Three Examples of the Designed Blocks

To enable the learners to work with (i) smart homes generated data, (ii) LED (Light Emitting Diode) lights and OLED (Organic Light-Emitting Diode) displays in micro-controllers, and (iii) mobile robots, we designed new blocks in addition to blocks for general programming features, which are discussed in the following.

With respect to the smart home, three blocks were designed for connecting with the smart home server, getting sensors and other items status, and the status of a smart mirror. The `server connection` block enables learners to connect to the smart home server, and has access to the name and status of all items; it is labeled as `connect server`. The `get status` block allows learners to use the name of each item in the smart home in order to access to its current status; it is labeled as `get status <type> item <name>`. The `<type>` filled in with the type of items (e.g., switches, dimmers, RGB lights, sensors, etc.). The `<name>` always refers to the name of the corresponding item. Furthermore, `get mirror status` is a simple block to return the facial expression which is recognized by the smart mirror; it is labeled by `get face emotions` (see Figure 3.2a).

With respect to the LED lights, the first block needed to define how many LED lights are connected to a pin is called `setup neopixel`. The color of LED lights can change through the `set pixel color` block. Furthermore, in order to colorize the LED lights, the command is encapsulated into another block called `show color` (see Figure 3.2b).

With respect to the OLED displays, the first block needed to define the display is called `setup display`. The cursor position can change through the `display cursor position` block. Furthermore, in order to show the data on OLED displays,

the command is encapsulated into another block called `display show`. Finally, OLED displays can be cleared via the block called `clear display` (see Figure 3.2c).

With respect to the mobile robots, we designed two sets of blocks. First, designed blocks are based on "*Free Navigation*" style, which enables learners to move around and navigate the robot `forward` and `backward` with different speed based on number of seconds or meters. In this style, learners are also enabled to `rotate` the robot with different speed based on number of seconds or degrees. Furthermore, learners access "*Laser Scanner*" data in order to recognize the obstacles around the robot—In an unknown environment, it helps the learners to avoid hitting the obstacles, or recognizing different obstacles (e.g., round shape obstacles) and go towards them. Second, blocks are designed based on "*Map Navigation*" style, which enables learners to navigate the robot based on the SLAM (Simultaneous Localization and Mapping) map of known environments. In this style, learners are enabled to navigate the robot based on different coordinates (X, Y, θ coordinate system) in the SLAM map. In both "*Free Navigation*" and "*Map Navigation*" styles, we used Gazebo robot simulation [PP03] to show a 3D model of the smart environment (virtual environment) and the robot's actual position (see Figure 3.3b). As stated in [KH04], Gazebo is "*a free 3D dynamic multi-robot environment capable of recreating the complex worlds that will be encountered by the next generation of mobile robots.*" In addition to Gazebo, a 3D visualization tool for ROS (RViz) [KLPK15] is employed to present the robot's percepts, e.g., laser scanner data, to the learners based on a SLAM map (see Figure 3.3c). Additionally, BEESM shows a 2D view of the robot's current position (see Figure 3.3a). In Figure 3.3, we illustrate a full picture of what is shown to the learners while working with BEESM in "*Map Navigation*" style.

3.2.3 User Interface Design

BEESM is divided into three parts which are *Smart Environment*, *Mobile Robot* and *Micro-controller*. We used visual programming, in particular Google Blockly to facilitate programming. Blockly is not a programming language, but it is a framework to help developers build visual programming environments [HLC16]. Blockly includes programming elements; each element is represented as a block shaped element that it can be snapped together like puzzle pieces. Blockly allows custom blocks by defining a block's appearance, inputs, outputs, and type of connections. The type of a block's connection is defined to show how blocks can snap together. The code generator for each block, which is written by the developer, enables Blockly to generate the code syntax of the programming language of our choice.

A useful feature of BEESM is the ability to run the code either from Block Panel and/or from the Code Panel. This feature allows learners to switch between

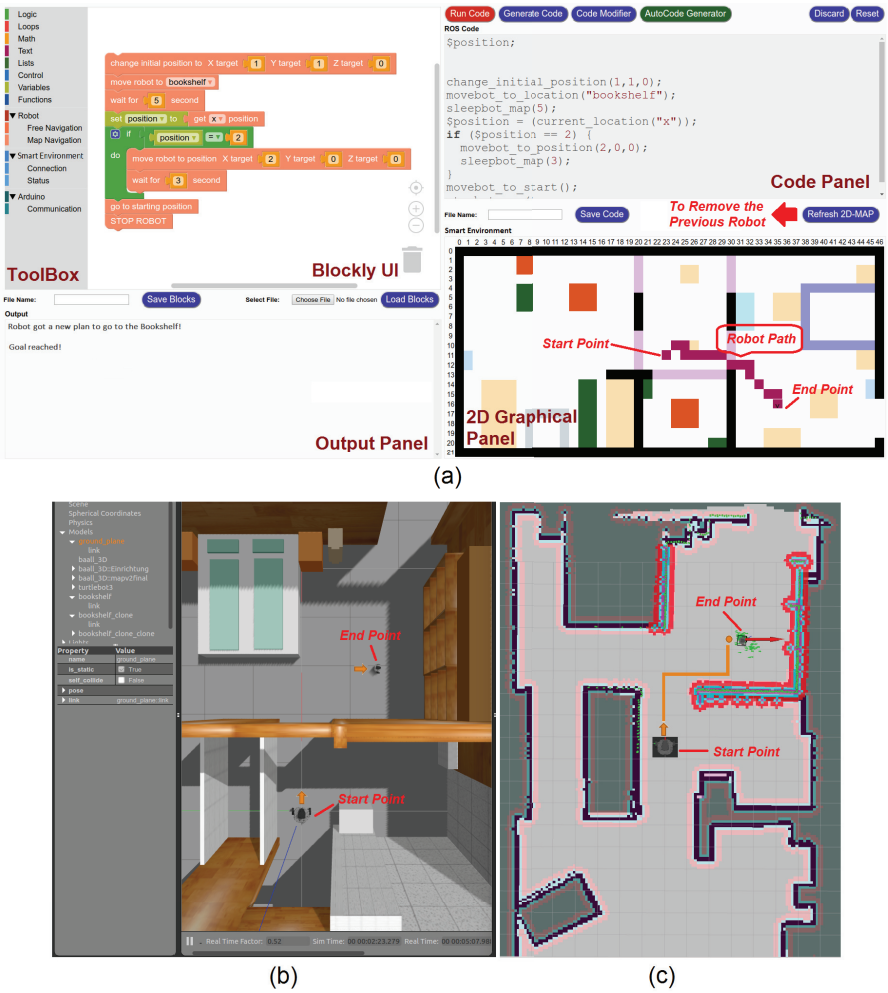


Figure 3.3: An overview of (a) BEESM user interface together with (b) GAZEBO robot simulation, and (c) RViz for ROS.

text codes and blocks freely. This enables them to program smart environments, micro-controllers, and mobile robots using both blocks or directly with code syntax. Following the argumentation in [Bau15, BBDP15, WW17a, WH17], this approach can also help inexperienced and young learners to get first insights in code syntax.

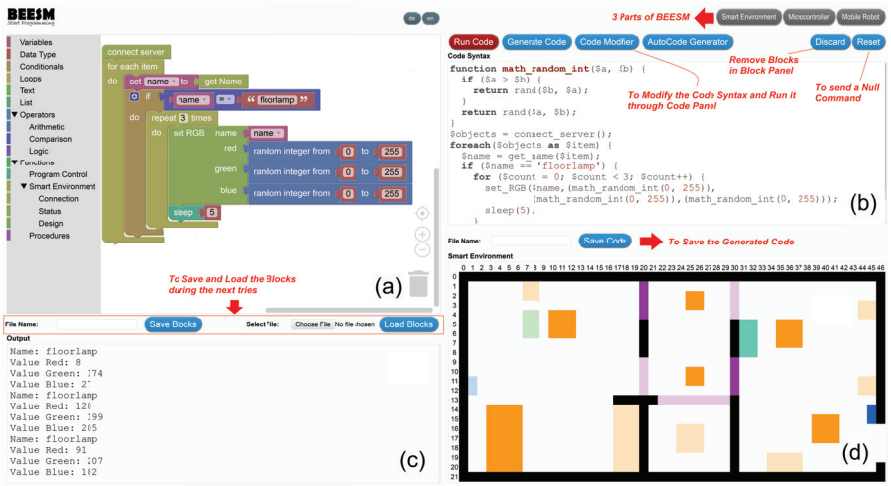


Figure 3.4: BEESM graphical user interface.

When learners click on the "*Code Modifier*" button, they can change the code and run it from the Code Panel instead of the Block Panel. The panel changes to the code which is generated by blocks as soon as the "*Generate Code*" button is clicked again. The "*AutoCode Generator*" button enables learners to see the code which is generated by blocks in real-time as soon as a block is added, removed or modified in the Block Panel. Additionally, the application presents a 2D view of the smart home (2D Graphical Panel) and an Output Panel. Thus, learners can see changes which are happened to different items in real-time in the 2D Graphical Panel as well as in the Output Panel (see Figure 3.4):

(a) *Block Panel* consists of a simple block workspace (Blockly UI) where learners can assemble blocks, and access block categories (Toolbox) that contain predefined and customized blocks for all primitives. Blocks in the same category share the same color to enable learners to find them easily.

(b) *Code Panel* demonstrates the generated code syntax by blocks to learners. They are also enabled to edit the code syntax which helps them to get first insights in code syntax . To this end, a simple version of programming languages used to help learners identify mistakes.

(c) *Output Panel* shows program output and errors for debugging purposes. The compile and upload process of the code into the micro-controller, as well as all return values are also shown in this panel.

(d) *2D Graphical Panel* includes a view of the smart environment and mobile robot's position, which provides learners a picture of the status of different items in real-time.

BEESM is designed and developed as an educational block-based programming tool to enable inexperienced and young learners to learn programming and author programs. Thus, two panels, Block and Code Panels, are mandatory to be part of the BEESM user interface. Block Panel is necessary because it allows users to see available blocks and to drag and drop blocks into this area in order to create a program. Code panel is also essential because it shows the code syntax which is generated by blocks. Additionally, using this panel, users can make direct changes in the code and see the output. This approach helps the user to become familiar with the code syntax and to get first insights into it, and thus, this feature remained the same throughout the workshops from the beginning until the end. Output and 2D Graphical panels are in the initial BEESM user interface. Error handling is part of programming, and thus, it plays an important role in introductory programming environments. BEESM includes an Output Panel to show errors to the users and help them to debug their program. It also helps educators and teachers to see the outcome of the program and help learners to solve the issues and errors. As BEESM is initially designed in the context of the smart environment, a 2D view of the environment is necessary in order to show the changes which have been made to the users in real-time—mobile robots and micro-controllers are added to it later to provide more options for the programming.

During our workshops, initial BEESM user interface has changed and we decided to have a larger section for blocks (addressed in Chapter 6 of this thesis). For learners, this is suitable to improve the visibility of finding and reading blocks in the program. In this respect, for those workshops which we only used Micro-controller part of BEESM, 2D graphical view of the smart environment was removed from the interface. This helps us to move the output panel to the right side of the interface and extend the Block Panel to cover the right side of the interface (more information can be found in Section 7.3.1). Furthermore, for those workshops which we need the 2D graphical Panel, BEESM user interface has changed in a way that Output Panel hides behind the Code Panel. This helps learners to toggle between the Code and Output Panels, using "*Output*" button. Also, "*AutoCode Generator*" button has been removed and it is replaced by the "*Output*" button. This also helps us to extend the Block Panel to cover the right side of the interface (more information can be found in Section 8.3.1).

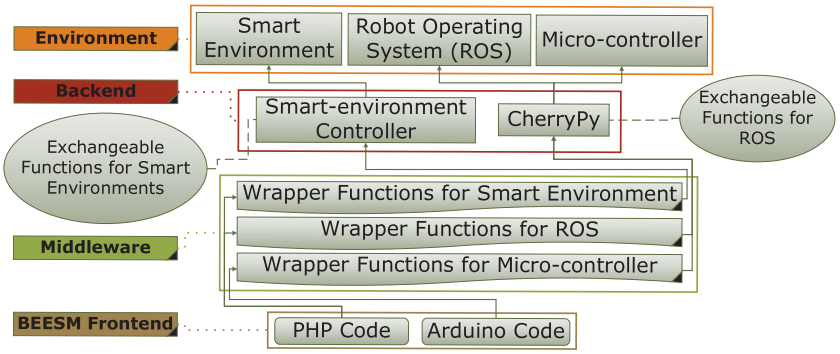


Figure 3.5: Architecture of BEESM.

3.3 BEESM Architecture

As shown in Figure 3.5, BEESM is divided into the three layers which are *Frontend*, *Middleware* and *Backend*:

- The *Frontend* includes BEESM user interface. Learners can generate either PHP code or Arduino code, using a set of blocks or directly using code syntax. In the *Smart Environment* and *Mobile Robot* parts, each block generates PHP code based on their inputs and outputs. Arduino code is generated in the *Micro-controller* part in order to upload the code directly towards micro-controllers.

- The *Middleware* consists of three files, containing a set of wrapper functions for the primitive behavior of smart objects, mobile robots and micro-controllers. These wrapper functions help to execute corresponding functions, which are understandable for smart environments using Smart-environment Controller, as well as Robot Operating System (ROS) [QCG+09] and micro-controllers using CherryPy (an open-source web-framework) [Che18].

- The *Backend* consists of Smart-environment Controller and CherryPy, which includes exchangeable functions for the respective environment. CherryPy contains the functions for mobile robots based on ROS, and Smart-environment Controller contains the functions for smart environments. For micro-controllers, CherryPy is used as a helper for Google Blockly to receive the generated code via an HTTP request and upload it on micro-controllers.

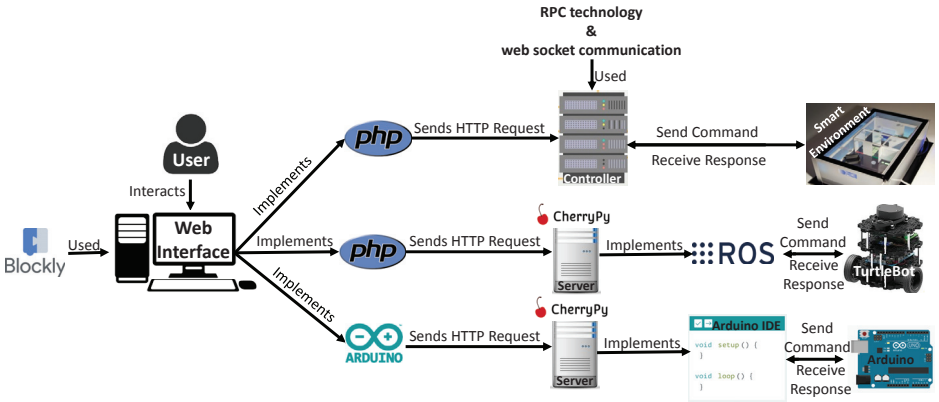


Figure 3.6: Detailed description BEESM architecture.

In order to have a clear discussion on BEESM architecture, Figure 3.6 shows the whole life cycle of it, starting form the user interaction with the web interface and ending commands send to the Smart Home, TurtleBot and Arduino board. The architecture of BEESM is divided into three levels:

(1) First level which is introduced as the *Frontend*: users interact with the web interface which is implemented using Blockly library to generate PhP and Arduino code.

(2) Second level which is introduced as the *Middleware*: a set of wrapper functions handle the transmission from the Frontend to the smart-environment controller and cherryPy web-server. These functions are called code syntaxes which are created by blcoks.

(3) Third level which is introduced as the *Backend*: smart-environment controller and cherryPy web-servers are implemented and send commands to the target objects and environments. This level is the actual part of BEESM structure that contains a set of functions in order to enable the target objects to communicate with each other. Thus, this approach enables users to program smart environments, microcontrollers and mobile robots one at a time and in combination with each other.

Overall BEESM is a customizable tool to assist inexperienced and young learners to learn and build PHP code in order to program smart environments and mobile

robots. Furthermore, Arduino code is used to program micro-controllers. The code to program is generally done by generating it for blocks which are at the top level of the program. When the learner runs the program, the generated code is sent to our *Backend* through *Middleware*. In this approach, smart environments can be controlled through mobile robots and micro-controllers. They can communicate with each other either directly or via *Middleware*. In an educational tool, this helps educators to give different accessibility to learners based on their needs and prior knowledge. BEESM backend consists of exchangeable functions for corresponding environments. Educators and researchers can change these functions to adapt and customize BEESM to other smart environments and mobile robots. In the *Micro-controller* part, the generated code is directly uploaded into the board using our backend as a helper for Google Blockly. Thus, BEESM architecture and its primitives provide a hassle-free environment for educators and researchers. They are able to use BEESM for educational purposes for different smart objects and environments. Moreover, BEESM can help learners via using graphical blocks to generate code syntax instead of memorizing it. This approach reduces syntax errors and eases the manipulation of code structures.

3.4 Integration and Discussion

BEESM is designed for inexperienced and young learners to enable them to learn the general purpose of programming in the context of smart objects and environments. It is a block-based programming tool for educational purposes that helps learners to experience and customize applications for mobile robots, micro-controllers and smart environments. It includes different programming language features and predefined functions based on the Blockly Library. Furthermore, in BEESM, different behaviors of tangible and smart objects are encapsulated in different functions which are presented as basic programming primitives. Each primitive can be called through a function, such as getting sensor input (e.g., light and temperature senors), the status of switchable devices (e.g., doors and lights), or the status of a mobile robot (e.g., current position). These primitives are defined as a set of custom blocks which are implemented to allow the learners to work with them and see the reactions of tangible and smart objects in real-time.

In the micro-controller part of the BEESM, Arduino's behavior and capabilities were implemented into primitives and wrapped in a set of blocks which generate Arduino code. Learners can then integrate these primitives into general programming features to author programs. Furthermore, the behavior and capabilities of the smart homes (programming primitives) can be applied to other smart environments using web socket communication protocol and RPC technology as a web-

based interaction to control household appliances. In this respect, OpenHab2 can be considered as a similar approach with a graphical user interface to control household appliances [HHKM17]. Similar to our approach, OpenHab2 which is a unified open-source home automation tool uses HTTP GET and POST requests that return JavaScript Object Notation (JSON) responses to communicate between user and Web server. Primitives were implemented in BEESM to work with all devices such as dimmable and switchable devices as well as sensors. In BEESM, the robots have a set of primitive capabilities, such as navigating to a location or reading the laser scanner data. Similarly, learners can then compose these primitives with general futures of programming languages to author programs. The programming language is wrapped in a graphical interface to enable inexperienced and young learners to benefit from it and learn basic concepts of programming.

Inexperienced and young learners are enabled to work with the primitives to program and observe the impacts of their programs on tangible smart objects and real life-size smart environments. Furthermore, BEESM has the advantage of helping learners to learn general features of programming, allowing them to implement new ideas while using blocks, and observing the impacts of their programs in reality. Using the proposed approach, learners are motivated to begin with programming activities and to learn and author programs to control different smart objects and environments.

Google Blockly offers a framework for developers to make programming easier for inexperienced and young learners. However, keeping in mind that introductory programming is difficult for them to solve programming problems [BBDP15, QL17], the visual block-based programming tool may still not be fully intuitive for them. In this respect, an oral introduction to general features of programming and the programming tool should be presented to the learners. Furthermore, supplementary documents (e.g., instructional procedures) can be implemented in order to help the learners to work with the tool and solve programming problems. As the learners might not listen carefully to the oral explanation or might not remain fully concentrated, these documents should be given to them in paper form while using the programming environment. Thus, in the Chapter 5 of this thesis, we look at the young learners' acquisition of programming skills and their interests in learning programming, using BEESM and supplementary documents in the context of smart homes. Prior to this chapter, we present detailed information about the design and development of our questionnaires and programming questions in the Chapter 4. With this regard, we begin by detailing the design of the questionnaires and continue with information about the programming questions in the next chapter of this thesis.

3.5 Limitations

The presented approach was tested on a real life-size smart home, a TurtleBot3, as well as on Arduino Uno and WEMOS D1 Mini boards. A necessary feature that needs to be added to BEESM is to enable inexperienced and young learners to program and control mobile robots using Python or C++ programming languages instead of PHP. This will help them to upload the code directly towards robots without having PHP functions in between. One of the most important future contributions is the evaluation of usability of BEESM amid the learners with and without prior programming knowledge with different genders. Furthermore, BEESM does not yet provide parallel programming and real-time hints for debugging purposes to help learners, but developing such an implementation is future work.

3.6 Conclusion

We presented the design and development of BEESM, a *Block-based End-user programming tool for SMart Environments*. BEESM allows inexperienced and young learners to rapidly prototype and experiment with new applications for smart devices and environments. BEESM broadens the application domain of existing block-based programming environments for the learners as they are not only enabled to program micro-controllers or mobile robots but also can connect to and program smart environments.

In conclusion, BEESM can be considered as a visual approach, offering fast configuration and programming through basic composition of blocks that represent predefined functions. Considering educators and researchers who are working in the area of computer programming education, BESSM is designed to be a block-based rapid programming tool which provides a hassle-free environment for them. In this regard, they can make it compatible with different smart environments and devices for educational purposes.

Chapter 4

Measuring Instruments: Questionnaires and Programming Questions

The main goal of this thesis is to enable inexperienced and young learners to begin with programming activities, and to experimentally evaluate the benefits of block-based programming environments as target platforms for learning to program. In addition to block-based programming, we employed computing platforms such as smart and tangible objects and environments as motivational contexts. In this respect, we designed and implemented six non-formal programming training sessions and workshops (from 2-hour to 4-day) in 17 months, from April 2018 until August 2019. In this chapter, we refer to these programming training sessions and workshops as *Experiment 1* to *Experiment 6*. A total of 108 German secondary school students (60 girls and 48 boys) participated in these experiments, who were between 10 to 15 years old. In the first four experiments, learners' attitudes towards programming and computer science were measured with questionnaires before and after using the programming environment and the computing platform. Furthermore, in the first two experiments (Chapter 5), two programming tasks were designed in order to assess the learning outcomes. In the next two experiments (Chpater 6), two programming questions were applied for the assessment of students' programming skills at the beginning and at the end of each experiment. Moreover, we provided three attitudinal questionnaires and programming questions in the fifth experiment (Chapter 7) at the beginning, in the middle and at the end of the experiment. This helps us to measure both learners' attitudes towards programming and assess their

programming skills in solving programming problems throughout this experiment. The sixth experiment (Chapter 8) consists of three attitudinal questionnaires to measure the learners' attitudes towards programming. In addition, students were required to answer two programming questions at the beginning and end of this experiment. This helps us to assess their programming performance in answering programming questions.

In this chapter, we present details on the design and development process of the questionnaires and programming questions in each experiment. Thus, we begin by detailing the questions which are included in each questionnaire and how they changed throughout the six experiments. Then, we present the information about the designed programming tasks and questions in each experiment, as well as how they changed to meet our needs in each experiment. This chapter is concluded with information about the preparation of these measuring instruments in a different phase of our research.

4.0.1 Attitudinal and Perceptional Questionnaires

The attitudinal questionnaires were generally based on the questions from [BBE+09, WW17b] with specific questions being added and being employed for each experiment; all translated from German to English and available in appendix A to appendix D. In order to understand how the learners' attitudes were affected in the respective experiment, we turn to responses to the attitudinal assessment. Additionally, our questionnaires consist of several questions to find out our students' perceptions of the block-based programming environment and the computing platform which are employed in each experiment. In all questionnaires, we asked each student to provide a unique code to be assigned to his/her particular questionnaires. We informed them that this code is necessary so that the data can be collected and analyzed anonymously.

Demographic questions were placed at the end of the post questionnaires in all experiments. It is mainly because of the findings from [TPO12, Lav08], which show the best place for demographic questions is at the end of questionnaires. In this respect, we refer to four advantages to place demographic questions at the end of the questionnaires, which are addressed by [Lav08]:

(1) It enables respondents to answer the questions before *boring* demographic questions.

(2) It is helpful to prevent primacy effects (e.g., due to gender differences).

(3) It helps respondents to answer the questions without any personal questions being asked at the beginning of the questionnaire.

(4) It engages and builds connections between the respondent and the questionnaire.

Experiment 1 & Experiment 2. The first phase of our experimental evaluation started with the first and second experiments. In these two experiments, we looked at two dimensions of students' attitudes towards programming and computer science in both pre and post questionnaires (see appendix A.1 and appendix A.2):

- *confidence* in finding computer science difficult to understand, and
- *interest* in learning programming.

Students were also required to rate their programming skills in the pre questionnaire. This question was designed to record the learners' prior programming experience at the beginning of each experiment. Furthermore, in the post questionnaire and after performing the computer programming tasks, four questions were asked. These questions were designed to understand how learners find programming with blocks and seeing the impacts of their programs in a real life-size smart environment. In both pre and post questionnaires, all questions asked on a 5-point Likert scale.

Please note that these two experiments were the first two programming training sessions that we conducted in our research. In these experiments, we designed and asked a limited number of questions. This helped us to get the first insights into how young learners react to the use of block-based programming together with real life-size smart environments in order to begin with programming activities and learn to program.

Experiment 3 & Experiment 4. In the second part of the first phase of our experimental evaluation (third and fourth experiments), we started to expand the scope of attitudinal questions via four different 5-point Likert scale questions. Furthermore, in this phase, we designed our question to focus more on the learners' attitudes towards computer programming. Thus, in this part of our research, we look at four dimensions of learners' attitudes (see for example, appendix B.1 and appendix B.2):

- *confidence* in finding themselves good at programming,
- *enjoyment* of programming,
- *perceived difficulty* of understanding programming, and
- *interest* in future programming learning opportunities.

Similar to the first two experiments, students were required to rate their programming skills in the pre questionnaire. Additionally, we used one "*yes*" or "*no*" questions to know whether all participants have ever worked with a block-based programming environment or not. These questions were designed to record the learners' prior programming experience with block-based programming at the beginning of each experiment. Furthermore, in the post questionnaire and after performing the computer programming activities, two questions were designed and asked to find out learners' attitudes towards programming a real object, using block-based programming. Five experimental questions were added to the post questionnaires in order to record the learners' experience in using block-based programming environments. In both pre and post questionnaires, all questions asked on a 5-point Likert scale. In this part of our research, we added an extra column, which is called "*I don't know*" to help students to use this option for those questions that they do not know how to answer. However, we found out that some of the students tended to pick this answer and did not read the questions. Thus, we removed this column in the questionnaires that we provided in the upcoming experiments.

Experiment 5. In the second phase of our research, we started to dig into the learners' trajectories of attitudes towards programming, using block-based programming and smart tangible objects. Thus, in the first part of this phase, we asked a set of open-ended questions to enable students to answer them using short responses. In contrast to the first phase of our research, the programming workshop was designed in a longer period of time. Our workshop is divided into two half: (i) learning basic programming concepts, using block-based programming, and (ii) applying these programming concepts on a tangible object and make it smart. Thus, in addition to the pre and post questionnaires at the beginning and the end of the workshop (see appendix C.1 and appendix C.3), we added another evaluation point, using an intermediate questionnaire in the middle of the workshop (see appendix C.2). In this experiment, we look closely into three dimensions of students' attitudes towards programming and the programming workshop, using five open-ended questions in the pre questionnaire:

- *confidence* in finding themselves good at programming and programming workshop,
- *enjoyment* of programming, and
- *interest* in future programming learning opportunities.

Please note that one question which is related to students' enjoyment in the workshop was changed to two questions in the intermediate and post questionnaires. All

other questions remained the same and asked gain in the intermediate and post questionnaires. Furthermore, two questions were added to the intermediate and post questionnaires to find out students' experience, using block-based programming and programming smart systems and objects. Using an extra question in both intermediate and post questionnaires, students were enabled to tell us if there was anything left and did not ask in the questionnaires. Similar to the two previous experiments, we used one "*yes*" or "*no*" question in the pre questionnaire and students were required to demonstrate if they have ever worked with a block-based programming environment. Some of the participants mentioned that in addition to block-based programming, they worked with micro-controllers in the past. For the next experiment, this led us to design another question in our pre questionnaire to ask students if they previously worked with a micro-controller.

Experiment 6. In the second part of the second phase of our experimental evaluation, we started to collect both quantitative data (5-point Likert scale questions), as well as qualitative data (short-response questions) to support the quantitative data. Similar to the previous experiment, the programming workshop was designed in a longer period of time and another evaluation point was added, using an intermediate questionnaire in the middle of the workshop (see appendix D.1, appendix D.2 and appendix D.3). This workshop is also divided into two half: (i) learning basic programming concepts, using block-based programming, and (ii) applying these programming concepts on a tangible object and make it smart. The quantitative questions were designed to find out how the learners' attitudes towards programming are affected by block-based programming and smart tangible objects over time. In this experiment, eight 5-point Likert scale questions were asked based on the three dimensions of students' attitudes towards programming:

- *confidence* in finding themselves good at programming and how they found programming difficult,
- *enjoyment* of programming and the workshop, and
- *interest* in future programming learning opportunities.

As it is mentioned in the previous experiment, in the pre questionnaire, we asked them two "*yes*" or "*no*" questions, which are:

- first question to record the students' prior programming experience with block-based programming, and
- second question to record their prior experience with using a micro-controller in the past.

In the intermediate and post questionnaires, one question was added to find out students' experience when they able to program smart tangible objects. Furthermore, eight 5-point Likert scale questions are designed based on the Weintrop et al. [WAS+18] study to measure the learners' experience using the programming environment in terms of its ease-of-use, ease-of-learning, usefulness, and satisfaction.

With respect to the short-response questions, we started to find out how students think about programming and the programming workshop, using two open-ended questions in the pre questionnaire. In the intermediate and post questionnaires, these two questions divided into four in order to understand how they like and dislike the workshop and the programming environment.

4.0.2 Programming Questions

The programming questions was generally based on the questions from [Lew10, MCK17]; all translated from German to English and available in the appendix A to appendix D. In order to understand how the learning outcome were affected in the respective experiment, we turn to students' performance in programming tasks and their responses to the programming questions. We focused on three basic programming concepts as follows: (i) variables, (ii) loops, and (iii) conditional statements and logical operators. In all programming questions, we asked each student to use the unique code which was provided in his/her particular questionnaire.

Experiment 1 & Experiment 2. We designed and implemented two programming tasks in the first two experiments (see 5.3.4). In the first task, we focus on learning variables and iterative logics. In this task, differences between variables were demonstrated, and the students were introduced to use an iterative loop (e.g., `foreach` loop) in order to iterate through a list of objects. In the second task, differences between operators were demonstrated, and the students worked with loops (e.g., `for` loop) and conditional statements (e.g., `if-else` statement). They continued to use loops and they were introduced to several if-else statements based on random numerical values.

Experiment 3 & Experiment 4. In the second part of the first phase of our experimental evaluation, we provided two novel programming questions at the beginning and the end of each experiment (see for example, appendix B.5, appendix B.6). In both pre and post programming questions, block-shaped elements were designed independent of the shape and color of blocks which are used in each experiment (see 6.4.1). Using these type of programming questions helped us to assess students' performance in solving programming problems based on their understanding of basic programming concepts but not the color and shape of blocks. For each

Table 4.1: 10-point Grading Rubric Scale

10-points (Connection)	8-points (Analysis)	6-points (Summary)	4-points (Incomplete)	2-points (Attempted)
Answer shows mastery of content and a deeper understanding of it.	Answer shows mastery and understanding of content, but it has a minor issue.	Answer shows some understanding of essential content, but it has a lack of greater evidence.	Answer does not show understanding of basic content, or it shows that mastery of the general content is missing.	Answer does not address the programming question or is off-topic.

programming question, we collected the solution made by the students. A 10-point grading rubric [RA10] was created to evaluate the students' performance (see Table 4.1). Please note that we informed students, which they need to put blocks in correct logical order to answer each programming question. We also noted that there is no unique correct answer, and some of the blocks may appear more than once while some others may not be needed in the answer to the programming question.

Experiment 5. In the second phase of our research, and due to having another evaluation point at the middle of the programming workshop, we designed three programming questions at the begging (see appendix C.4), in the middle (see appendix C.5) and at the end of the workshop (see appendix C.6). In pre programming question, students' prior programming experience was validated via answering to the question. Furthermore, students' performance in basic programming concepts which were taught was assessed, using the two other programming questions at the middle and the end of the workshop. Similar to the previous experiments (experiment 3 and 4), block-shaped elements were designed independent of the block-based programming environment in all programming questions (see 7.3.1). We asked the students to answer the question via selecting the blocks and putting them in order in a correct logical way. Students' performance were assessed and scored, using the 10-point grading rubric. All programming questions were designed to be slightly different from each other, and their level of difficulty were increased from the pre to the post programming question. This is to ensure that students read the questions carefully and do not answer the intermediate and post programming questions same as each other or the pre programming question.

Experiment 6. In the final part of our research, we followed not only the concepts which are covered in [MCK17, WW15] but also we designed our programming questions based on their questions. In this respect, we designed two programming questions to assess the students' understanding of variables, loops and conditional statements at the beginning and the end of the workshop. These programming questions consist of gap-filling and open-answer questions as it is designed and implemented by Merkouris et al. [MCK17]. Similar to the previous programming question, we designed our programming questions slightly different from each other

to ensure that students read the questions carefully and do not answer the post programming question same as the pre programming question.

4.0.3 Summery

The preparation of measuring instruments was guided by six programming training sessions and workshops which were held in 17 months. These training sessions and workshops helped us to explore attitudinal and programming questions to measure our learners' attitudes towards programming and assess their programming performance. Moreover, these training sessions and workshops led us to explore feasible programming activities for learning computer programming with block-based programming, using tangible and smart computing platforms as motivational contexts.

In the first phase of our research, 67 students (26 boys and 42 girls) participated in four short programming training sessions (between 2 to 4 hours). In this stage, which is conducted in the first nine months of our experimental evaluation period, the initial attitudinal questions to be answered were explored and selected (Chapter 5 and Chapter 6). Additionally, the main programming activities and questions to be performed were designed and implemented in this stage. Students' feedbacks and responses helped us to refine the attitudinal and performance measuring instruments, as well as to add an extra evaluation point at the middle of upcoming programming training workshops (Chapter 7 and Chapter 8).

In the second phase of our research, 40 students (22 boys and 18 girls) participated in two longer programming training workshops (2 to 4 days). In the first part of this stage, the attitudinal questionnaires and programming questions were tested in a 4-day programming workshop (Chapter 7). In this workshop, students had a greater chance to show their creativity in design and development of a smart tangible object, which can be integrated into real life-size smart homes. We designed and implemented open-ended attitudinal question to record short responses to our questions. Furthermore, similar to the second part of the first phase (Chapter 6), we used block-shaped elements to assess students' programming skills in solving programming problems. Final rectification of the measuring instruments was performed during this part. In the second part of this stage (Chapter 8), attitudinal questionnaires consists of both close-ended and open-ended questions. Moreover, programming questions were designed and developed based on the block-based programming environment to assess students' programming performance in answering programming questions.

Chapter 5

Instructional Supports for Block-based Programming

There is scientific knowledge about how to teach programming, and the necessity to foster young learners' interest in computer science is broadly addressed. However, there is a lack of research on how to teach programming skills in a way that increases the learners' interest in the topic. In the Chapter 3, a novel block-based programming approach was introduced, allowing inexperienced and young learners to learn the basic concepts of programming and author programs. Apart from learning programming and rapidly prototyping applications in the context of smart environments, this approach provides a programming environment for educators to make it compatible with different tangible and smart objects and environments.

In this chapter, we present a one-day non-formal training session for young students, in order to support the acquisition of programming skills and, at the same time, a positive view towards programming and computer science in general. The programming environment is based on one application of *BEESM* within a smart home. Thus, the abstract concept of programming is presented within a real context and tightly connected to real experiences. In this training session, the learners were introduced to a real life-size smart home and to programming concepts in order to acquire basics of programming. Two user studies with 44 7^{th} and 8^{th} grade students were conducted, specifically, the students' interest in computer science and programming, as well as their acquisition of programming skills were assessed. Furthermore, two instructional interventions to support learners, namely worked examples and instructional procedures, were compared. This chapter contributes by showing the potential of using visual block-based programming in the context of smart homes in order to enable students to begin with programming activities.

5.1 Introduction and Motivation

Computer science and programming are becoming more and more important in K-12 education [GSH+18, Kal15]. Several studies aimed to introduce programming to young students in order to integrate computer science in K-12 education [GSH+18, LK14, KM16, BS11]. These studies investigated how different programming environments affect K-12 students' computational thinking towards problem solving [GSH+18, LK14, FSK+13]. According to the K-12 Computer Science Framework [Com17], programming is one of the computer science core concepts [Com17] and core practices [BS11]. Thereby, programming is more than just coding [LK14, GSH+18], but involves computer science concepts such as abstraction and debugging to solve problems [LK14, Com17]. During programming, students are exposed to computational thinking aspects such as creativity, critical thinking and problem solving [LK14, Com17, KB13].

Acknowledging that the main goal of programming training for K-12 students is to minimize the distractions of "*thinking at multiple abstractions*" [LK14, Com17], motivating students through programming activities helps them to understand how computer science influences their daily life [PHEC17]. Thinking at multiple abstractions (e.g., students are enabled to find a piece of code that could be more efficiently implemented as a loop) and programming activities can be considered to be fundamental for K-12 students in order to reduce complexity and increase efficiency of training [Cof17, GSH+18, LK14]. In this way, one of the challenges that computer science research still faces is the lack of studies investigating empirical evidence in computer science education and training among young learners [Com17, LK14].

Young learners typically have difficulties to understand the requirements of designing, executing and debugging programs [GSH+18, QL17]. Likewise, they do not have experience in procedural and modular programming to solve programming issues. Thus, in the first place, training them to acquire programming skills seems to be difficult for school teachers and educators [GSH+18, PHEC17, HA17]. Furthermore, learning and recalling code syntax as well as assembling and manipulating code structures are error prone as they require a high level of concentration [Bau15, BBDP15]. Kalelioğlu [Kal15] tried a new way of teaching programming skills to K-12 students in order to investigate the effect of teaching programming on reflective thinking skills towards problem solving. It was shown clearly that students learned computer science and programming concepts in the code.org platform which is based on Scratch while playing in an enjoyable way. In this respect, many block-based programming editors such as Scratch [RMMH+09], Alice [CDP00], Snap! [HM10] and Google Blockly [Fra14] have been evolved in today's advancing technology and applications [Com17, LK14] in order to reduce the complexity of programming. These editors use a visual programming language, where they rep-

resent a vision in which K-12 students engage in the concept of computer science and programming [Com17, MRR+10]. Visual block-based programming is designed to allow the students to program without the obstacles of syntax and manipulation of code structure errors [Com17, Bau15, BBDP15]. Young learners can learn programming through drag and snap blocks together in order to generate code syntax [Com17, LK14]. They can take advantage of visual block-based programming in order to facilitate the design, the execution and the debugging process of programming, and thus, they are able to solve programming problems more easily.

We conducted two user studies in order to examine the acceptance of computational thinking through problem solving among 7^{th} and 8^{th} grade students (12 to 15 years old). We claim that it is possible to introduce new technologies which provide possibilities to connect computer science to reality and cover basic programming skills in one-day non-formal programming training sessions in order to enable students making programs and seeing their impacts in real-world environment. To this end, a visual block-based programming environment (*BEESM*) is used in order to reduce the complexity of programming and facilitate it for the students in the context of a real life-size smart home. The main goal of the smart home is to provide exposure to programming activities in the programming training sessions. An empirical evaluation of the training sessions with respect to the interest in computer science and the acquisition of programing skills was conducted. Furthermore, as introductory programming is difficult for young students [BBDP15, QL17], visual block-based programming environments may still not be fully intuitive for them. Thus, they need help in order to solve programming problems using these environments. Based on instructional supports presented in [SKR+10, MGG+14, ZLP18], it was addressed that young learners can be supported by worked examples and instructional procedures in order to solve programming problems. Therefore, by using these supplementary documents, we helped our young students to work with the programming environment and solve programming problems.

Our training sessions were conducted in Germany, where the number of female graduates in the field of computer science is considerably lower than the number of male graduates [ELP17, Cor17]. Thus, introducing programming to our target users has two aspects. On the one hand, the acquisition of programming skills among inexperienced and young students was measured in the context of a real life-size smart home using a visual programming paradigm. On the other hand, the acceptance of computer science and programming among young learners, especially girls, was assessed. It was a special feature of our training sessions that students were able to program a real-world environment. Collecting and analyzing data in this environment extended and refined prior results [ELP17, GSH+18] taken from different non-formal education environments where students were able to produce and deploy pieces of code that can be applied to smart homes.

This study seeks to contribute to computer science education through the development, implementation, and evaluation of a programming training session, with the goal to teach basic programming skills and principles to young learners, and with the goal to arouse their interest in computer science and programming.

The rest of this chapter is structured as follows. In Section 5.2, background and related work are described. Our training sessions are described in Section 5.3. Section 5.4 presents two user studies, the first one with 8^{th} grade students without prior programming experiences, the second one with 7^{th} grade students with little prior programming experience. The results are discussed in Section 5.5; the chapter closes with conclusions in Section 5.6.

5.2 Related Work

Over the past few years, several block-based programming tools have emerged to provide visual environments [LK14,HLC16,SDP+15]. The environments aim to help K-12 students getting started with programming activities [Mak18, Bau15]. These initiatives focus on promoting computational thinking among K-12 students [Kaf16, KB13]. However, there have been very few studies with a special focus on promoting real-world environment programming activities among K-12 learners—both female and male— with different levels of prior programming experience [PHEC17].

Visual programming environments were designed in a variety of approaches (e.g., [Kal15, FSK+13, HA17, HLC16]) for introducing K-12 students to general programming features and principles. The literature reports that programming is more accessible to young learners and novices using these environments [LK14], and results show that young learners are able to learn programming in visual block-based programming scenarios. In addition, according to [PHEC17, KM16], researchers are interested in the "*robot programming*" approach in end-user programming tools. Huang et al. [HLC16] took advantage of Google Blockly to develop a rapid block-based programming tool called "*CustomPrograms*" for end users. Paramasivam et al. [PHEC17] used this tool to design an end-user application which enables K-12 students to program a mobile robot. With this regard, twelfth-grade high-school students with disabilities (e.g., deafness, low vision or blindness, Asperger's Syndrome) attended a robot programming workshop. The results obtained from this experience are encouraging, as the authors were successful to establish the confidence that robot programming is both accessible and interesting. However, no information was provided whether this procedure also inspired interest in programming and computer science in general.

The literature also reports block-based programming tools used for stimulating interest in computer science among K-12 learners [LK14, SDP+15, PHEC17,

KM16, HA17]. In terms of teaching programming, two of the preferred visual block-based programming editors are Scratch and Google Blockly. For instance, Scratch—one of the most successful visual programming editors—is used to present MOOC (Massive Open Online Courses) [SDP+15] programming courses in which researchers teach elementary programming concepts to inexperienced high [KM16] and elementary [HA17] school students. The results showed that the majority of the participants had a positive attitude towards programming and stated that they plan to continue programming in the future. Nevertheless, teaching elementary programming features to inexperienced and young learners in the context of smart environments using visual programming tools is still an active area of research.

5.2.1 Summary

The use of visual block-based programming editors (e.g., Scratch or Google Blockly) alone may not be sufficient for K-12 learners to gain a deep understanding of computer science concepts and programming skills. Although these editors are able to reduce the complexity of programming for young learners, they may not enable the learners to develop a well-grounded connection between computer science and its impacts in their daily life. Granting access to a real life-size smart home may be particularly useful in order to show the learners how a real-world environment can react to their program when it is following the programming principles and structures. To this end, we designed and implemented one-day non-formal programming training sessions to provide learning opportunities for young learners in order to program a real life-size smart home in the German Research Center for Artificial intelligence (DFKI). Thus, learners can participate and experience new technologies which are adapted to technical equipments in order to help elderlies and people with disabilities. In other words, by using the smart home and letting learners to program it, we aim to motivate them to learn programming and to understand influences of computer science in their daily life.

5.3 Training Sessions

Training sessions were held in premises of the University of Bremen. All used equipments and the smart environment—computers and the smart home—which were used were provided by the DFKI. The core idea of such one-day non-formal programming training session is to enable inexperienced and young students to learn basic programming concepts and improve their attitudes towards programming independent of their regular curricular and outside of their school environment. Three instructors were involved in each session. The participants were introduced to the

smart home by one instructor. The other two instructors were in charge of the teaching sessions. One instructor was female and all of them had a computer science background with experience in working with K-12 students. The goal of the training was to enable inexperienced and young students to acquire primary programming skills and to use these skills in the context of a real life-size smart home.

At the beginning of each session, the students received a pre-questionnaire. Likewise, at the end of each session a post-questionnaire was given to them in order to assess the learners' view on computer science and programming. Apart from this, each session was divided into three parts: (i) introduction to the smart home, (ii) introduction to programming structures and principles with the block-based programming environment, and (iii) performing two programming tasks. The questionnaires and the three parts of the training sessions are described in the following.

5.3.1 Questionnaires

The questionnaires were designed in order to evaluate the learners' view on computer science and programming. The learners were asked to indicate their prior programming experience in a pre-questionnaire at the beginning of each session. In addition to demographical questions (e.g., age, gender), students were asked to indicate their view on computer science and on programming, specifically with respect to the block-based programming environment, in a post-questionnaire at the end of the sessions. Pre- and Post-questionnaires can be found in the appendix A.1 and appendix A.2, respectively; all translated from German to English.

5.3.2 Overview of the Smart Home

Our smart home is an approximately 60 m^2 automated smart ambient assisted living lab apartment (BAALL) including different smart items such as a sink adjusting automatically to a person's height, height-adaptable kitchen, an intelligent wardrobe suggesting outfits, voice recognition system, and smart mirror and fridge (see Figure 5.1) [FSR+10]. It comprises all necessary conditions for trial living, intended for the elderly and people with physical or cognitive impairments. The smart home also contains various actuators (e.g., doors, toggleable, dimmable and RGB lights), and sensors (e.g., lighting, temperature, and thermal cameras). Furthermore, the smart home aims to compensate physical impairments of the users through mobility assistance, such as wheelchairs and walkers. With this regard, the smart home has been equipped with five automated sliding doors that open to let the wheelchairs pass through. All lights and doors are remote controllable via a restful HTTP interface, or proactively by the intelligent environment. The smart home's main educational use here is letting young students program different ob-

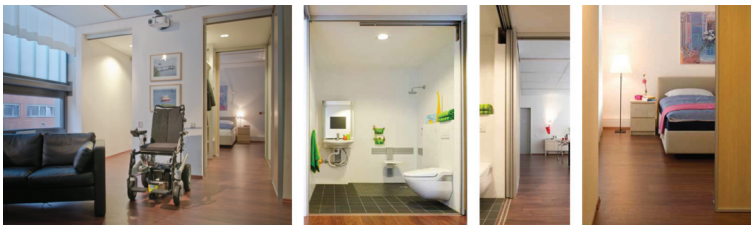


Figure 5.1: A view of the smart home (BAALL).

jects for different purposes in a real life-size smart environment which is also used by researchers. Additionally, it provides a unique opportunity for them to experience latest innovation and to learn about the future. In this respect, students are enabled to program different real objects in the smart home and to observe reactions of these objects to the program in real-time.

During each session, the students were divided into two groups in order to enable an introduction to the smart home in smaller groups. All objects and their functionalities in the smart home were explained to each group in order to (i) enable students to understand how computer science can influence real-world environments, and (ii) identify different smart items which are used in the smart home.

5.3.3 Introduction to Programming Structures and Principles

In this study, a block-based programming environment was used to enable inexperienced and young students to make programs in the context of a real life-size smart home (see Figure 5.2). This environment is based on *BEESM* which is primarily designed as a visual block-based tool, being applicable for smart objects and environments. We followed the *BEESM* user interface (see Figure 5.2) to enable learners using four different panels to have a full vision of the blocks (Block Panel), code syntax (Code Panel), output of the code (Output Panel), and a 2D view of the smart home (2D Graphical Panel).

All students were introduced to programming and how to use the programming environment. They learned general aspects of using the environment, (i) how to identify the blocks relevant to solve the programming tasks, and (ii) how to recognize the main elements and panels of the programming environment.

The main computational concept which was taught is "*Programming Structures and Principles*", exemplified through visual block-based programming samples. With this regard, programming features such as variables, data types, conditional statements, loops, and logical operators, as well as pre-defined functions (consist of

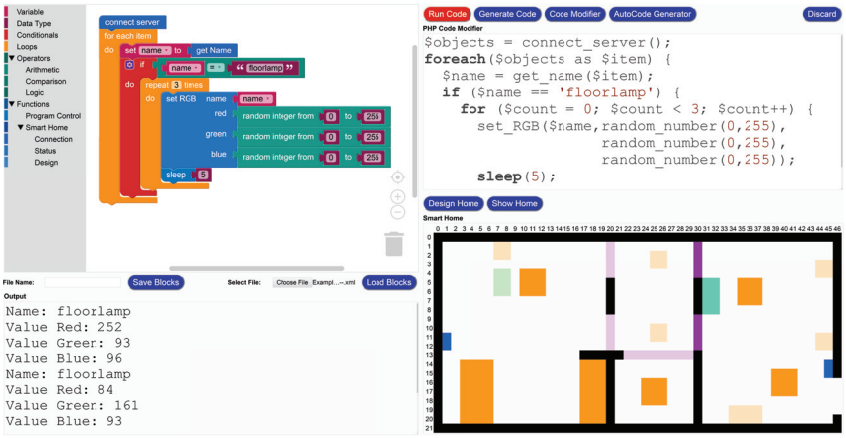


Figure 5.2: A sample execution of the programming environment.

different kinds of behaviors of smart objects) were taught. Programming concepts were also introduced using simple block-based programs that include variables, iterative logics and conditional statements, and later through more complex examples that add loops, logical operators and pre-defined functions in order to make different applications in the smart home. Students are encouraged to load pre-defined examples and execute them in order to recognize which block corresponds to a particular action.

5.3.4 Programming Tasks

During each session and before each task, the students were introduced to the protocols of the programming tasks, specifically (i) that they have only 20 minutes to finish each task, and (ii) that they need to use their own supplementary document which comes along with each computer. In this study, in addition to an oral introduction to programming and the programming environment, we used supplementary documents—namely worked examples and instructional procedures—in order to help our students working with it. As introductory programming is difficult for young learners [BBDP15, QL17], and they might not listen carefully to the oral explanations or might not remain fully concentrated, supplementary documents were provided in paper form. These documents supported the students while solving the programming tasks with the programming environment. A worked example for each task was attached to half of the computers in paper form. The document contained a visual block-based representation (blocks are snapped together) of the

task, along with an explanation of each block and of the code which is generated by the blocks. Worked example documents for both programming tasks can be found in the appendix A.3 and appendix A.5, respectively; all translated from German to English. To the other half of the computers, an instructional procedure for each task was attached in paper form. This document contained a visual block-based representation (blocks are not snapped together) of the task, along with an explanation of each block. The output of the task was also included in both documents. Instructional procedure documents for both programming tasks can be found in the appendix A.4 and appendix A.6, respectively; all translated from German to English. In the first programming task, one of the blocks was used incorrectly in the documents which was highlighted by a different color. The students needed to identify the incorrect block and to replace it by the correct one in order to correctly perform the task. In the second programming task, one whole loop was used in an incorrect format which was also highlighted by a different color. The students needed to identify the incorrect set of blocks and replace them by the correct blocks in order to correctly perform the task. Furthermore, they were encouraged to perform the task using a simple example as an introduction to the particular task. The two programming tasks were:

(1) Showing the name and status of each object in the smart home. The task helps students to learn variables and iterative logics. Students are required to connect to the smart home's server, iterate through the list of objects (`foreach` loop), and fetch the name and status of each object. Then, they assign object name and status to two variables and show them in the Output Panel.

(2) Changing the status of an RGB light. The task helps students to learn differences between operators as well as working with loops (`for` loops) and conditional statements (`if` statements). Students are required to connect to the server, iterate through the list of objects and find the corresponding RGB light. Then, they change the status of the light using random integers for 3 times, with 5 seconds delay between the changes.

In this respect, the computational concepts are extended by introducing these tasks to the students. We require students to use the supplementary documents in order to identify the issue and where the change should be made, remove extra blocks and add new blocks, integrate them with the rest of the program, and finally test the program.

5.4 User Studies

Two user studies were conducted in order to understand the impact of visual block-based programming on young learners' programming skills and their attitudes towards programming and computer science in the context of smart homes. The training sessions were conducted with *inexperienced* students without prior programming experience (Experiment 1) and with *novices* who already had minor experiences in visual and text-based programming (Experiment 2). Specifically, we wanted to understand whether the students could assimilate and use supplementary documents—namely worked examples and instructional procedures—including an issue in order to perform programming tasks. In that respect, the students were divided into two groups which were supported either with worked examples (Example Group) or with instructional procedures (Instruction Group). Concretely, the user studies addressed the following two research questions:

(1) When learning how to program with a visual block-based programming environment embedded in a real life-size smart home, is it more effective to present learners worked examples compared to instructional procedures? Does this effect depend on gender?

(2) When learning how to program with a visual block-based programming environment embedded in a real life-size smart home, is interest in computer science and programming fostered more when learners are presented worked examples compared to instructional procedures? Does this effect depend on gender?

In both studies, data with respect to the acquisition of programming skills and with respect to interest in computer science and programming were collected.

In the following, we describe the sample, the training session, the collected data, and the results for both studies.

5.4.1 Experiment 1

Sample and Design. A total of 22 8^{th} grade students of a German secondary school (12 girls, 10 boys, age: $M = 13.80$, $SD = 0.56$) participated in the study. The students were randomly assigned to two experimental groups, 6 girls and 4 boys to the Example Group (they revived a worked example for each task), and 6 girls and 6 boys to the Instruction Group (they received an instructional procedure for each task), respectively.

Procedure of the Training Session. The duration of training for each student was 3 hours, with a 15 minutes break. Students were randomly assigned to the

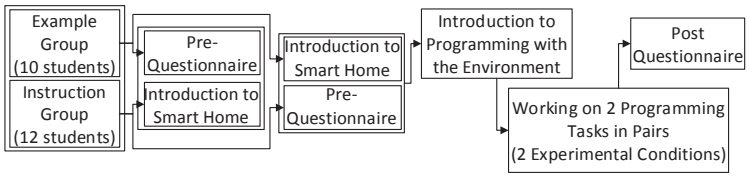


Figure 5.3: Procedure of the first experiment.

two experimental groups at the beginning of the session. The Example Group students were asked to answer the questions of a pre-questionnaire, while the Instruction Group was introduced to the smart home; afterwards, the Example Group was introduced to the smart home, while the Instruction Group answered the pre-questionnaire (see Figure 5.3). All objects and their functionalities in the smart home were explained for 20 minutes per group. Then, as gender effects were considered, pairs of two students (2 boys or 2 girls) were assigned to one computer. Each computer showed a real-time full vision of the smart home during the session using three IP cameras. All students were introduced together to programming features, structures and principles as well as how to use the programming environment for one hour. Before working on each task, students were presented a program introducing the corresponding task. The two programs respectively were (i) demonstrating the name of all available objects in the smart home, and (ii) changing the status of a dimmable light for one time. At the end of the second task, each group of students went to the smart home to see the changes in reality. All students were asked to complete a post-questionnaire at the end of the session.

Acquisition of Programming Skills. It was assessed whether the students were able to perform the tasks or not. Each task consisted of several steps. The performance was operationalized by the rate of steps completed without errors. In this respect, at the end of each training session, the generated blocks were checked in each computer. We labeled each block with a value and gave a final rate to the whole program based on the blocks which were correctly used and placed. Furthermore, the number of errors were counted and the type of errors was categorized as "*major*" or "*minor*" errors based on students' difficulties in introductory programming which are discussed in [QL17]. *Minor* errors are related to the students' syntactic knowledge; for example, missing variable names or typing errors while using blocks. In contrast, *major* errors are mostly happened during assembling and manipulating code structure using blocks; for example, using an `if` statement block to check a condition without using a `foreach` loop block to get all the objects names; thus, demonstrating flaws in the students' conceptual and strategic knowledge [QL17].

Confidence and Interest in Computer Science and Programming. The subjective data regarding confidence and interest in computer science and programming was collected using a questionnaire. The students were asked to rate the items (1) "*is it easy to program with blocks?*", (2) "*do blocks help you to understand program better?*", and (3) "*do you think that it is helpful to be able to see directly in reality whether the program works as desired?*" using a 5 point Likert scale (with 1 being "*no*", and 5 being "*yes*"). Furthermore, the students were asked about their preference of programming with blocks or directly with code using a 5 point Likert scale (with 1 being "*definitely with code*", and 5 being "*definitely with blocks*").

Finally, students were asked before each session started in the pre-questionnaire and at the end of each session in the post-questionnaire to rate (1) "*do you think computer science is difficult to understand?*", and (2) "*would you like to learn how to program?*" on a 5 point Likert scale (with 1 "*no*", and 5 "*yes*").

Results. At the beginning of the training session, the students were asked to indicate whether they had programming experience. Only four students answered that they had programmed before: two boys and one girl in the Example Group, and one boy in the Instruction Group. Nevertheless, all four students indicated "*low*" or "*no*" prior experience, thus, the level of prior knowledge was not included in the further analyses.

The following analyses were computed as two-factorial analyses of variance, with the factors *example vs. instruction* and *gender*, respectively (see Table 5.1). For the questionnaire items, "*no*" was coded with 1, and "*yes*" was coded with 5, respectively. With respect to finding programming easy with blocks, on average, the students indicated a medium level; no significant main or interaction effects occurred, all $F < 1$. The students indicated that blocks are helpful in order to understand programs; no significant main or interaction effects occurred, all $F < 1$.

With respect to the helpfulness of seeing the impacts of their program in a real-world environment, neither the main effect *gender* nor the interaction effect reached the level of significance, both $F < 1$. The students in the Instruction Group rated the question higher in comparison to the students in the Example Group (see Table 5.1); however, the main effect *example vs. instruction* was not significant, $F(1,18) = 1.16$, $p = 0.30$.

Concerning the preference for programming with blocks or with code, "*code*" was coded with 1 and "*blocks*" were coded with 5. The students in both groups indicated an indecisive stance (see Table 5.1). However, the Instruction Group indicated a slightly higher preference towards blocks compared to the Example Group, and the girls indicated a higher tendency towards blocks compared to the boys. However, both main effects just barely missed the level of significance, both $F(1,17) = 3.15$, $p = 0.09$. The interaction effect was also not significant, $F(1,17) = 1.14$, $p = 0.30$.

Table 5.1: Subjective Data on the Ease of Use

	Example Group			**Instruction Group**		
Questions	***All Students*** ***M (SD)***	***Female*** ***M (SD)***	***Male*** ***M (SD)***	***All Students*** ***M (SD)***	***Female*** ***M (SD)***	***Male*** ***M (SD)***
Is it easy to program with blocks?	3.10 (1.52)	3.00 (1.67)	3.25 (1.50)	3.42 (1.24)	3.33 (1.03)	3.50 (1.52)
Do blocks help you to easily understand programs?	3.90 (0.99)	3.83 (0.98)	4.00 (1.15)	4.27 (0.90)	4.20 (1.10)	4.33 (0.82)
Do you think that it is helpful to be able to see directly in reality whether the program works as desired?	3.70 (1.06)	3.50 (1.05)	4.00 (1.15)	4.33 (1.30)	4.33 (1.63)	4.33 (1.03)
Do you prefer to program with block or directly with code syntax?	2.90 (0.74)	3.00 (0.89)	2.75 (0.50)	3.45 (0.93)	4.00 (1.00)	3.00 (0.63)

M: Mean **SD**: Standard Deviation

With respect to the question whether they would like to learn how to program (see Table 5.2), before the training session the boys showed broad approval, while the girls were undecided. Accordingly, the main effect *gender* yielded a significant result, $F(1,18) = 8.89$, $p = 0.008$. However, neither the main effect *example vs. instruction* nor the interaction effect reached the level of significance, $F(1,18) = 1.38$, $p = 0.26$, and $F < 1$, respectively. After the training session, the boys still indicated that they would like to learn how to program significantly more than the girls, $F(1,18) = 7.46$, $p = 0.01$. Concerning the main effect *example vs. instruction* and the interaction effect, no significant results occurred, both $F < 1$. Descriptively, the boys and the girls indicated a higher level for liking to learn how to program before the training session compared to after the training session. In order to determine whether this decrease was significant, a regression analysis was performed. For the girls, the regression slope from the after-session score to the pre-session score was not significant, $B = 0.26$, $p = 0.59$ (constant: $B = 1.67$, $p = 0.30$). For the boys, the regression slope from the after-session score to the pre-session score was also not significant, $B = -0.28$, $p = 0.43$ (constant: $B = 5.24$, $p = 0.01$). Thus, the decrease with respect to liking to learn how to program was not significant, neither for the girls, nor for the boys.

With respect to computer science being difficult to understand, on average, the students indicated a medium level before the training session (see Table 5.2); no significant main or interaction effects occurred, all $F < 1$. After the training session, the boys indicated a medium level of difficulty for computer science, while the girls opted more in the direction of "*difficult*". However, the effect *gender* was not significant, $F(1,18) = 2.97$, $p = 0.10$. Neither the effect *example vs. instruction* nor the interaction effect were significant, both $F < 1$.

After the introduction to programming with the programming environment, the students were asked to perform two programming tasks. Due to technical problems, the results of one group (two girls) in the Example Group were not saved, and thus, cannot be included in the following analyses. On account of the small sample size

Table 5.2: Subjective Data on Students' Interest

	Example Group			Instruction Group		
Pre-Questionnaire	***All Students*** ***M (SD)***	***Female*** ***M (SD)***	***Male*** ***M (SD)***	***All Students*** ***M (SD)***	***Female*** ***M (SD)***	***Male*** ***M (SD)***
Do you think computer science is difficult to understand?	3.30 (0.67)	3.33 (0.52)	3.25 (0.95)	3.58 (0.90)	3.67 (0.82)	3.50 (1.04)
Would you like to learn how to program?	4.00 (0.94)	3.50 (0.84)	4.75 (0.50)	3.58 (1.44)	2.83 (0.75)	4.33 (1.63)
Post-Questionnaire						
Do you think computer science is difficult to understand?	3.50 (0.97)	3.83 (0.98)	3.00 (0.82)	3.33 (1.07)	3.67 (1.03)	3.00 (1.09)
Would you like to learn how to program?	3.20 (1.32)	2.50 (1.05)	4.25 (0.96)	3.17 (1.59)	2.50 (1.52)	3.83 (1.47)

M: Mean **SD**: Standard Deviation

($n = 10$ dyads), we decided not to perform analyses of variance with respect to these two tasks. Overall, the students performed 56% of task 1 and 49% of task 2 without errors (see Table 5.3). Concerning both tasks, the students in the Instruction Group performed better than the students in the Example Group. As the students worked in dyads either girls with girls or boys with boys (not in mixed-gender groups), we were able to analyze the percentages of tasks solved correctly dependent on gender. In both tasks, the girls performed considerably better compared to the boys.

For both tasks, errors were categorized into "*major*" and "*minor*" errors. In order to take into account the different number of students in the two experimental groups, the number of errors was divided by the number of dyads: 2 girl dyads and 2 boy dyads in Example Group; 3 girl dyads and 3 boy dyads in Instruction Group (see Table 5.3). Overall, in the first task, the number of major errors was higher than the number of minor errors. The students in the Example Group made more major errors than the students in the Instruction Group. Likewise, the students in the Example Group made more minor errors compared to the Instruction Group. In the second task, the number of major errors was higher than the number of minor errors. The students in the Instruction Group made more major errors and less minor errors than the students in the Example Group. Major errors occurred more often than minor errors, and both types of major and minor errors occurred more often among boys than among girls.

Across both tasks, the most common mistakes were setting different values to the same variable (minor error), using loops in an incorrect place (major error); for example, using a `for` loop to iterate through a list while the list is not defined yet, and placing blocks outside of loops and conditional statements where they should be placed within (major error).

Table 5.3: Students' Performance

Tasks		Example Group *All Students (per dyad)*	Example Group *Female (per dyad)*	Example Group *Male (per dyad)*	Instruction Group *All Students (per dyad)*	Instruction Group *Female (per dyad)*	Instruction Group *Male (per dyad)*
Task 1	*Major* error numbers	8 (2.00)	4 (2.00)	4 (2.00)	11 (1.83)	4 (1.33)	7 (2.33)
	Minor error numbers	7 (1.75)	5 (2.50)	2 (1.00)	2 (0.33)	0 (0.00)	2 (0.67)
	Rate of task completed without errors	50%	50%	50%	60%	75%	44%
Task 2	Major error numbers	9 (2.25)	4 (2.00)	5 (2.50)	16 (2.67)	5 (1.67)	11 (3.67)
	Minor error numbers	5 (1.25)	1 (0.50)	4 (2.00)	6 (1.00)	3 (1.00)	3 (1.00)
	Rate of task completed without errors	45%	59%	31%	51%	67%	36%

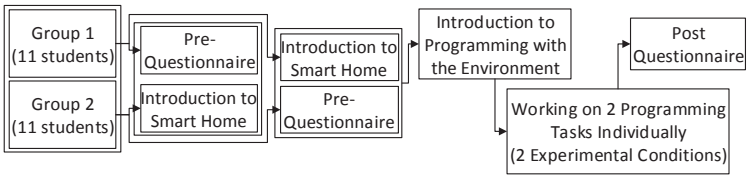


Figure 5.4: Procedure of the second experiment.

5.4.2 Experiment 2

Sample and Design. A total of 22 7^{th} grade students of a German advanced secondary school (6 girls, 16 boys, age: $M = 12.45$, $SD = 0.60$) participated in the study. The students were randomly assigned to two experimental groups, 3 girls and 8 boys to each the Example Group and the Instruction Group, respectively.

Procedure of the Training Session. The duration of the training session for each student was 3 hours, with a 15 minutes break. At the beginning of the session, students were randomly assigned to one of two groups (11 students per group) which were trained separately one after another. In the first training group, 6 students (3 girls and 3 boys) received a worked example for each task (Example Group), and 5 students (3 girls and 2 boys) received an instructional procedure for each task (Instruction Group). In the second training group, 6 students (all boys) received an instructional procedure for each task (Instruction Group), and 5 students (all boys) received a worked example for each task (Example Group). The first training group was asked to answer the questions of a pre-questionnaire, while the other group was introduced to the smart home; afterwards, students in the second training group were introduced to the smart home, while the first group answered the pre-questionnaire (see Figure 5.4). All objects and their functionalities in the smart home were explained for 20 minutes per group. Then, each student was assigned to one computer. Each computer showed a real-time full vision of

the smart home during each session using three IP cameras. All students were introduced together to programming features, structures and principles as well as how to use the programming environment for one hour. Before working on each task, students were presented a program introducing the corresponding task. The two programs respectively were (i) demonstrating the name of all available objects in the smart home, and (ii) changing the status of a dimmable light for one time. At the end of the second task, students went to the smart home to see the changes in reality. All students were asked to complete a post-questionnaire at the end of the session.

Acquisition of Programming Skills. It was assessed whether the students were able to perform the tasks or not. The students' performance on the programming tasks were evaluated in the same way as in Experiment 1.

Confidence and Interest in Computer Science and Programming. The subjective data regarding confidence and interest in computer science and programming was collected and analyzed in the same way as in Experiment 1.

Results. At the beginning of the training session, the students were asked to indicate whether they had programming experience. All but one (in the Example Group in the first training group) had already programmed before, and on a scale from 1 to 5 they indicated a medium level of prior programming experience ($M = 2.64 \mid SD = 0.79$). With respect to this score, the students in the Instruction Group indicated a higher level of prior knowledge compared to the Example Group (Instruction Group: $M = 2.82 \mid SD = 0.87$, Example Group: $M = 2.45 \mid SD = 0.69$). A two-factorial analysis of variance with the factors *example vs. instruction* and *gender* revealed no significant main effect for *gender*, $F < 1$; however, the main effect *example vs. instruction* and the interaction effect just barely missed the level of significance, for both effects $F(1,18) = 3.35$, $p = 0.08$, respectively. Thus, for the following analyses, this score was included as a covariate.

With respect to finding programming easy with blocks, neither the main effect *example vs. instruction* nor the interaction effect reached the level of significance, both $F < 1$ (see Table 5.4). However, the main effect *gender* was significant, $F(1,17) = 4.75$, $p = 0.04$. Boys indicated a higher level towards easiness of programming with blocks compared to the girls. The effect of prior programming skills was not significant, $F < 1$. Likewise, students indicated that blocks are helpful in order to understand programs; no significant main or interaction effect occurred, main effect *example vs. instruction*: $F < 1$, main effect *gender*: $F(1,17) = 2.46$, $p = 0.14$, interaction effect: $F < 1$. The effect of prior programming skills was also not significant, $F < 1$.

Table 5.4: Subjective Data on the Ease of Use

	Example Group			**Instruction Group**		
Questions	***All Students*** ***M (SD)***	***Female*** ***M (SD)***	***Male*** ***M (SD)***	***All Students*** ***M (SD)***	***Female*** ***M (SD)***	***Male*** ***M (SD)***
Is it easy to program with blocks?	4.18 (1.08)	3.67 (1.15)	4.38 (1.06)	3.82 (0.75)	3.00 (0.00)	4.13 (0.64)
Do blocks help you to easily understand programs?	4.91 (0.30)	4.67 (0.58)	5.00 (0.00)	4.82 (0.40)	4.67 (0.58)	4.88 (0.35)
Do you think that it is helpful to be able to see directly in reality whether the program works as desired?	4.73 (0.47)	4.33 (0.58)	4.88 (0.35)	4.73 (0.65)	5.00 (0.00)	4.63 (0.74)
Do you prefer to program with block or directly with code syntax?	4.27 (1.10)	5.00 (0.00)	4.63 (0.74)	4.73 (0.47)	4.67 (0.58)	4.75 (0.46)

M: Mean **SD**: Standard Deviation

With respect to the helpfulness of seeing the impacts of their program in real-world environment (see Table 5.4), neither the main effect *gender* nor the main effect *example vs. instruction* reached the level of significance, both $F < 1$. The interaction effect and the effect of prior programming skills were also not significant, $F(1,17) = 1.16$, $p = 0.30$ and $F(1,17) = 2.12$, $p = 0.16$, respectively.

Concerning the preference for programming with blocks or with code, "code" was coded with 1 and "blocks" were coded with 5. On average, the students indicated an opinion strongly towards blocks (see Table 5.4). The students in the Instruction Group indicated a higher preference for programming with blocks compared to the Example Group. Girls opted more for blocks compared to the boys. However, no significant main or interaction effects were obtained, main effect *example vs. instruction*: $F < 1$, main effect *gender*: $F(1,17) = 1.33$, $p = 0.26$, interaction effect: $F(1,17) = 1.06$, $p = 0.32$. The effect of prior programming skills was also not significant, $F < 1$.

With respect to the question whether they would like to learn how to program (see Table 5.5), no significant main or interaction effects occurred before the training session, all $F < 1$. However, the effect of prior programming skills yielded a significant result, $F(1,17) = 6.30$, $p = 0.02$. After the training session, no significant main or interaction effects were obtained, main effect *example vs. instruction*: $F < 1$, main effect *gender*: $F(1,17) = 1.38$, $p = 0.26$, interaction effect: $F < 1$. The effect of prior programming skills was also not significant, $F(1,17) = 2.65$, $p = 0.12$. Descriptively, the boys indicated a slightly higher level for liking to learn how to program before training session compared to after the training session. The girls showed a strong opinion for liking to learn how to program before the training session which remained the same after the training session.

With respect to computer science being difficult to understand, on average, the students indicated a medium level before the training session (see Table 5.5); no significant main or interaction effects occurred, all $F < 1$. The effect of prior programming skills was also not significant, $F < 1$. After the training session, the

Table 5.5: Subjective Data on Students' Interest

	Example Group			Instruction Group		
Pre-Questionnaire	***All Students*** ***M (SD)***	***Female*** ***M (SD)***	***Male*** ***M (SD)***	***All Students*** ***M (SD)***	***Female*** ***M (SD)***	***Male*** ***M (SD)***
Do you think computer science is difficult to understand?	3.10 (0.94)	3.00 (1.00)	3.13 (0.99)	3.10 (0.94)	3.00 (1.00)	3.13 (0.99)
Would you like to learn how to program?	4.73 (0.47)	4.67 (0.58)	4.75 (0.46)	4.73 (0.65)	5.00 (0.00)	4.63 (0.74)
Post-Questionnaire						
Do you think computer science is difficult to understand?	2.91 (1.14)	3.00 (1.00)	2.88 (1.25)	3.00 (0.63)	2.67 (0.58)	3.13 (0.64)
Would you like to learn how to program?	4.45 (0.52)	4.67 (0.58)	4.37 (0.52)	4.73 (0.65)	5.00 (0.00)	4.63 (0.74)

M: Mean **SD**: Standard Deviation

students indicated a lower level of difficulty for computer science. Neither significant main nor interaction effects occurred, all $F < 1$. The effect of prior programming skills was also not significant, $F < 1$.

After the introduction to programming with the programming environment, the students were asked to perform two programming tasks. Due to technical issues, the results of two students (one girl in the Example Group and one boy in the Instruction Group) were not saved, and thus, cannot be included in the following analyses. Overall, the students performed 76% of task 1 and 58% of task 2 without errors (see Table 5.6). Concerning both tasks, the students in the Example Group performed better than the students in the Instruction Group. In task 1, the girls performed slightly better compared to the boys. In contrast, the boys performed slightly better in task 2 compared to the girls. Overall, in the first task, no significant main or interaction effects occurred, all $F < 1$. The effect of prior programming skills was also not significant, $F(1,15) = 1.11$, $p = 0.31$. In the second task, no significant main or interaction effects were obtained, main effect *example vs. instruction*: $F < 1$, main effect *gender*: $F(1,15) = 1.10$, $p = 0.31$, interaction effect: $F < 1$. The effect of prior programming skills was also not significant, $F(1,15) = 3.11$, $p = 0.10$.

For both tasks, errors were categorized into "*major*" and "*minor*" errors. In order to take into account the different number of boys and girls in the two experimental groups, the number of errors was divided by the number of them in each group: 2 girls and 8 boys in Example Group; 3 girls and 7 boys in Instruction Group (see Table 5.6). Overall, major errors occurred more often than minor errors. There were no large differences between girls and boys, neither with respect to the number of major errors, nor with respect to the number of minor errors. In the first task, the students in the Instruction Group made more major errors than the students in the Example Group. Neither the main effects *gender* and *example vs. instruction* nor the interaction effect were significant, all $F < 1$. The influence of prior programming experience was also not significant, $F(1,15) = 1.44$, $p = 0.25$. No minor errors occurred in task 1.

Table 5.6: Students' Performance

Tasks		Example Group			Instruction Group		
		All Students (per student)	*Female (per student)*	*Male (per student)*	*All Students (per student)*	*Female (per student)*	*Male (per student)*
Task 1	*Major* error numbers	12 (1.20)	1 (0.50)	11 (1.38)	15 (1.50)	5 (1.67)	10 (1.43)
	Minor error numbers	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)
	Rate of task completed without errors	78%	94%	74%	73%	71%	74%
Task 2	*major* error numbers	20 (2.00)	5 (2.50)	15 (1.87)	21 (2.10)	7 (2.33)	14 (2.00)
	Minor error numbers	15 (1.50)	3 (1.50)	12 (1.50)	19 (1.90)	5 (1.67)	14 (2.00)
	Rate of task completed without errors	60%	53%	61%	56%	54%	56%

In the second task, the number of major errors was slightly higher than the number of minor errors (see Table 5.6). The students in the Instruction Group made more major errors and more minor errors compared to the students in the Example Group. With respect to major errors, no significant group differences were obtained, main effect *example vs. instruction* and interaction effect: $F < 1$, main effect *gender*: $F(1,15) = 1.70$, $p = 0.21$. The influence of prior programming experience was also not significant, $F(1,15) = 2.55$, $p = 0.13$. With respect to minor errors, no significant group differences were obtained, main effect *example vs. instruction*, main effect *gender* and interaction effect: all $F < 1$. The influence of prior programming experience was also not significant, $F(1,15) = 1.97$, $p = 0.18$.

Across both tasks, the most common mistakes were typing errors (minor error) and placing blocks outside of loops and conditional statements where they should be placed within (major error).

5.5 Integration and Discussion

The results presented in the previous section are now discussed. In this respect, we begin with our findings based on the research questions presented in Section 5.4 of this chapter; followed by implications and limitations.

5.5.1 Findings

Our findings are discussed in order to answer the research questions stated in Section 5.4:

(1) *When learning how to program with a visual block-based programming environment embedded in a smart home, is it more effective to present learners worked examples compared to instructional procedures? Does this effect depend on gender?*

In general, both boys and girls showed a high tendency towards using visual block-based programming environments. The results in both experiments showed that girls preferred (descriptively) to program with blocks—compared to traditional

code syntax—more than boys. In the second experiment, novice boys rated the easiness of programming with blocks significantly higher than novice girls.

With respect to the two programming tasks for the inexperienced learners in Experiment 1, the learners in the Instruction Group performed better than those in the Example Group in both programming tasks. Matching this result, the learners in the Instruction Group indicated that they found computer science slightly less difficult at the end of the session compared to the students in the Example Group. In Experiment 1, concerning both programming tasks, the girls performed better, but still the boys indicated that they would like to learn how to program significantly more than the girls, and the girls indicated (descriptively) higher values with respect to finding computer science difficult. Thus, the self-perception of the girls does not seem to match the objective achievement.

With respect to the number of errors in the two programming tasks for the inexperienced students (first experiment), major errors occurred considerably more often than minor errors, and boys made more major and minor errors than girls. Thus, the better learning outcomes for girls are also reflected in the lower number of errors. Furthermore, more major and less minor errors occurred in the Instruction Group compared to Example Group.

Concerning the type of errors, on the one hand, the students in the Instruction Group had more issues with the structure of the program, and it seemed that they did not follow the instructions strictly. On the other hand, at least some students in the Instruction Group were able to follow the instruction and solved the programming issue with a minimum number of errors and a maximum performance.

With respect to the two programming tasks for the learners with some prior programming experience (second experiment), the girls performed better in the first task, and they indicated significantly higher willingness to learn how to program compared to the boys. In the second task the boys performed slightly better than the girls, and the learners in the Example Group performed slightly better than the learners in the Instruction Group. Thus, the novice girls with some prior experience in computer science and programming had a better self-perception which matches their objective achievements. Furthermore, the learners in the Example Group performed slightly better than the learners in the Instruction Group across the two programming tasks.

With respect to the number of errors across the two programming tasks, in Experiment 2, major errors occurred considerably more often than minor errors. In the first task, boys made more major errors than girls, and in the second task, girls made more major errors than boys. Thus, the learning outcome for girls in the first and second task is reflected by the number of errors. Furthermore, more major and minor errors occurred in the Instruction Group compared to the Example Group. Concerning the type of errors, on the one hand, the learners in the Instruction

Group had more issues with the placing of blocks in their corresponding loops, and it seems that they did not strictly follow the instruction which was given to them. On the other hand, the performance and the number of errors fluctuated among the students in the Instruction Group. One might argue that the students who participated in Experiment 2 already attended a computer science course, and thus, it is not clear why they still made errors related to the program structure. However, they tried different ways to solve the programming issue. Thus, it seems that they made more errors while trying to solve the issue.

(2) *When learning how to program with a visual block-based programming environment embedded in a smart home, is interest in computer science and programming fostered more when learners are presented worked examples compared to instructional procedures? Does this effect depend on gender?*

According to the analysis of our pre-questionnaire data, overall there is a significant difference in learners' willingness to learn programming by gender. This result is important as it is observed only among learners without programming experience and repeated in the post-questionnaire where the boys indicated that they would like to learn programming significantly more than the girls. With respect to the students' programming experience, an interesting result shows that the students' willingness to learn programming dropped (descriptively) among inexperienced learners towards the end of the training session. In contrast, the level of perceived difficulty for computer science decreased (descriptively) among inexperienced learners towards the end of the training session. Thus, it can be concluded that programming experience has an important influence on students' view on learning programming and on the perceived difficulty of computer science. The type of supplementary documents did not have a significant influence in this respect in both experiments.

Even if the learners in Experiment 2 showed a positive attitude towards working with a visual programming environment, a one-day non-formal programming training session may have a negative influence on both inexperienced boys and girls. The influence of supplementary documents is not clear as they did not differ significantly in terms of the interest in computer science, student's willingness to learn programming, and the perceived difficulty for computer science. Furthermore, novices indicated an opinion strongly towards using blocks and smart homes for programming purposes. The students found computer science easier to understand after the training session. However, this short training session had also a negative influence on their willingness to learn programming; thus, further studies should assess effects in a long-term perspective with a focus on girls' programming skills and attitudes towards programming and computer science, more broadly.

5.5.2 Implications

The future needs computer scientists and programmers from different gender. However, we are aware that getting young students interested in computer science and programming, especially female learners, is difficult. The results of our training sessions show that students are able to start building their own programs which can be applied to the smart home. In this respect, visual programming environments can be helpful to simplify programming for young learners and to provide computational support for them. Our main take-home message from these user studies is that visual block-based programming within a smart environment is suitable in order to improve self-perception among novices to begin with programming activities. Furthermore, another advantage of beginning to expose young learners to programming activities is that learners can realize that computer science can be presented in a way which is not necessarily difficult to understand. The results obtained from our experience are promising, as inexperienced girls were able to perform successfully two programming tasks with the provided programming environment. This is supporting the results presented in [GSH+18] that no significant difference was observed in software-based project scores by gender. However, our studies showed that inexperienced students' interest in learning programming can be decreased, especially among inexperienced female students. This result is in contrast to results from other programs targeting young female learners in Germany like [ELP17] which showed that by providing opportunities for K-6 female students to have positive experiences in STEM fields, we may have them in the future in STEM professions. While it is not possible to trace our result back to specific features of the training session, it might be possible that the short duration of only one day of training had a negative influence. Thus, it remains an open point for future iteration of this work to assess effects of the duration of training on learning outcome and attitudes towards computer science and programming, which we studied in the following chapters of this thesis.

5.5.3 Limitations

We would like to emphasize that our results might be affected by the nature and number of programming tasks as well as by the length of the training sessions. Follow-up studies are required in order to understand how young learners, especially girls, react to non-formal programming training sessions running for several days in different contexts within smart homes such as programming robots and micro-controllers. There, we can find out the sustainability of attitudes and basic programming skills that they learned in order to transfer them to other contexts. Another question for future work is to ascertain when K-12 learners can move from

visual block-based programming environments to pure programming IDEs, using traditional text-based code syntax. Although our sample included 44 7^{th} and 8^{th} grade students, this sample size is still too small to generalize the findings on a large scale. Furthermore, studies in different countries and with learners of different socio-economic status might as well shed light on effects of visual block-based programming environments and real life-size smart homes on young learners' knowledge acquisition and interest in computer science and programming.

5.6 Conclusion

This chapter presents a training session developed for inexperienced and young students, in order to support the acquisition of programming skills and in order to support a positive view towards computer science and programming. The programming environment is based on *BEESM*, which is a visual block-based programming application within a real life-size smart home. Thus, the more abstract concept of programming is presented within a real context and tightly connected to real experiences for the learners.

The results show that students are able to build their own programs which can be applied to the smart home. Furthermore, the results indicate the importance of supporting and strengthening the learners' motivation in learning programming and their interest in computer science over a longer period.

Two prominent instructional interventions to support learners, namely worked examples and instructional procedures, were compared. As the results did not strongly support one of them, it can be concluded that both are appropriate in order to help learners to acquire basic programming skills. Thus, we will not focus on these instructional interventions in the following chapters of this thesis.

Future work should investigate how long-term training sessions, including more diverse programming tasks affect the learners' interest in computer science and programming as well as acquisition of programming skills, especially among girls. In addition, it should be investigated how to adapt non-formal training sessions to individual prior knowledge and learning processes in order to support motivation and knowledge acquisition in an optimal way. This study shows the potential of using visual block-based programming environments in the context of smart homes in order to foster young students' programming skills, and to enable them to begin with programming activities.

Chapter 6

Students' Attitudes and Skills: Impacts of Block-based Programming Environments

In the previous chapter, a non-formal programming training session was presented for young students in order to support learning of basic programming concepts and to provide a positive view towards programming. One application of the designed tool (*BEESM* which is introduced in Chapter 3) was evaluated in the context of smart homes, using two supporting features, namely worked examples and instructional procedures. The primary goal of this thesis is enabling the students to begin with programming activities, assessing their interest in computer science and their acquisition of programming skills. Nevertheless, the results of previous chapter show a greater focus on girls is needed in order to understand how their attitudes and programming skills are influenced by different block-based programming environments.

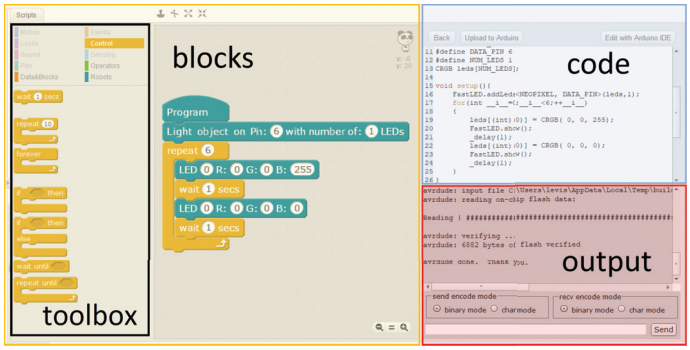
This chapter presents the results of two user studies with 24 female German secondary school students in two non-formal programming training sessions. We use and compare two block-based programming environments based on Scratch and Google Blockly in fostering the students' programming skills and changing their attitudes towards programming. The two block-based programming editors have been chosen as they are popular in the current educational use of block-based programming. The contribution of this study is showing the different impacts of Scratch and Google Blockly on young female students' interest in programming and the acquisition of programming skills in non-formal programming training sessions.

6.1 Introduction and Motivation

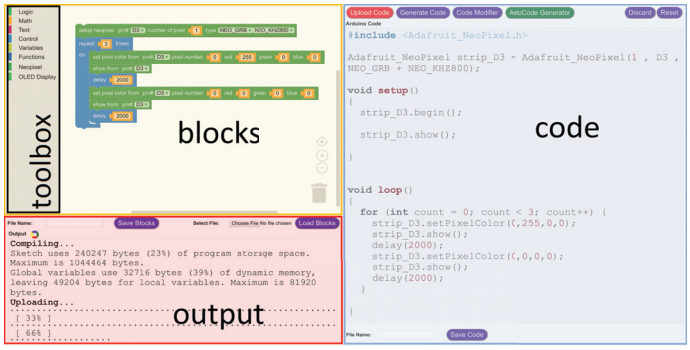
In recent years, the ability to program is increasingly becoming an important skill in our high-tech world [WW17a, WH17]. Therefore, many introductory programming environments are designed and developed in order to enable a wider range of audiences to start programming. Block-based programming is known as an approach that is widely used in the design of introductory programming environments. Using block-based programming allows inexperienced and young learners to program through visual block-shaped programming elements such as variables, loops, conditional statements, logical operators, and functions. These elements can be dragged, dropped, and snapped together like puzzle pieces [HH18, WH17]. Acknowledging that young students can be motivated through programming activities, the visual block-based approach is becoming significantly important for introducing basic programming concepts and eventually lead to develop an interest in computer science in general [JKGM18, MFM16, WH17].

The low number of women compared to the high number of men in higher computer science education is a well-known problem in most western countries [Cor17, Lag08, ELP17]. It was addressed that approximately 25% of females pursuing a career in STEM in the EU [ELP17], and fewer than 1 in 5 computer science graduates are female across 35 European countries [Cor17]. The lack of computer science interest among girls or women has been shown to emerge partly from technological aspects of computer science [Lag05, Coh02]. In this regard, block-based programming is introduced as an alternative to conventional text-based programming languages that appears less technical and brings new dimensions such as creativity to the understanding of programming among young students [WW15, WW17b, Rom07].

Since programming was taught to young students via block-based programming, many block-based programming editors such as Scratch [RMMH$^+$09], Alice [CDP00], Snap! [HM10], and Google Blockly [Fra14] have been used for introduction to programming in the context of computer science education [MGB15, WW17b, MCK17]. Much work has been done investigating what type of editor is more beneficial for young students to have a better understanding of programming [WW17b, WH17, AH16]. Despite the fact that these editors are widely adapted in the design of introductory programming environments [WW17a, HH18], an open question remains about how well such programming editors can enable *young female students* to acquire basic programming skills and, at the same time, to improve their attitudes towards programming. More specifically, one challenge that computer science educators and researchers still face is the lack of studies investigating empirical evidence in using block-based programming editors among young female students. The empirical evidence is subject to describe under what circumstances a given block-based programming editor is the better choice to foster interest and program-



(a)



(b)

Figure 6.1: A view of the user interface for the (a) *mBlock* and (b) micro-controller part of *BEESM* (MpB).

ming skills among the students in order to motivate them to join future computer science education.

In this chapter, we compare two block-based programming environments (BBPEs) that are based on two popular block-based programming editors, dominating the current educational use of block-based programming [HH18]: MBlock [Mak19] which is based on Scratch [RMMH+09] (see Figure 6.1a) and micro-controller part of *BEESM* (MpB) which is based on Google Blockly [Fra14] (see Figure 6.1b). We claim that these two BBPEs have different influences with respect to the students' attitudes towards programming and the acquisition of basic programming skills. Thus, this chapter seeks to answer the following research question:

How do two widely used block-based programming editors—Scratch and Google Blockly—perform in order to foster young female students' programming skills and positive attitudes towards programming?

In order to examine the acceptance of introductory programming and the experience with the BBPEs among young female students, two user studies were conducted with 24 beginners (10 to 14 years old) in total. The students used the BBPEs to program a micro-controller (Arduino or WeMos boards) to control LED lights. An empirical quantitative evaluation of the two BBPEs with respect to the young female students' attitudes and perceptions of programming as well as their acquisition of basic programming skills, was conducted. Collecting and analyzing data in this study refined prior results that have shown the acceptance of programming among young students [WW15, MCK17] where they were able to learn programming via block-based programming.

The rest of this chapter is structured as follows. In Section 6.2, background and related work are described. An overview of the block-based programming environments is presented in Section 6.3. Section 6.4 presents the method of study design and strategy for collecting data. The evaluation and experimental results are presented in Section 6.5, and they are discussed in Section 6.6. The chapter closes with limitations and conclusions in Section 6.7.

6.2 Related Work

Over the past years, block-based programming has become a popular approach for the design of introductory programming environments for young learners. Block-based programming editors are designed to help young learners getting started with programming with a low threshold [HH18, WW17b, ML18]. Using block-based programming allows developers and educators to reduce the difficulty of initializing basic programming concepts (e.g., programming structure and principles) among young learners [MFM16, AH16]. Block-based programming was used in a variety of studies (e.g., [MFM16, AH16, ML18, WH17, MGB15, PB15, RMH14]) to facilitate the accessibility of learning programming, especially for inexperienced and young students. Literature reports that block-based programming makes programming pleasant, engaging, and motivating for young students, and thus, it can leverage their interest in programming and computer science in general [WW17b, WW17a]. However, relatively little work has been done with a special focus on using block-based programming to promote basic programming activities among young female students in non-formal programming training sessions.

The literature also reports that in most western countries the women's lack of interest in programming and experience with computers are two of the reasons

to have a low number of women in computer science [Lag08, Lag05]. In that respect, using block-based programming is reasonable to introduce young female students to programming and basic computational skills [MFM16, AH16]. In contrast to the studies which are questioning the suitability of block-based programming (e.g., [MSABA11, PEH07]) in order to motivate young students and prepare them for future learning programming opportunities, block-based programming is recommended as the first choice in introductory programming and computer science courses [ML18, WH17]. Several studies show that the use of block-based programming in formal [WW15, WW17b] and non-formal [GSH+18] educational context has a positive influence on young students' programming skills and interest. Studies also report successes of teaching programming concepts to young students and foster their interest and motivation in learning programming using block-based programming compared to text-based programming [WW17b, WW15, RMH14]. However, understanding the impacts of block-based programming in non-formal learning environments remains an active area of research, with a focus on how best to utilize BBPEs to foster young female students' programming skills and leverage their interest in programming and computer science.

With respect to teaching programming, two popular block-based programming editors—namely Scratch and Google Blockly—have made significant contributions to the current educational use of block-based programming [HH18, AH16, JKGM18]. Scratch which is known as one of the most successful editors has been used to investigate the affordances of block-based programming in comparison with text-based programming environments in order to teach basic programming concepts to inexperienced high school students [WW17b, WW15]. As a result, using block-based programming, the majority of students gained more programming knowledge, had a better perception of programming, and they were more interested in continuing programming in the future. Furthermore, according to [PHEC17, MGB15], researchers are using Google Blockly to encourage young students to start programming robots [PHEC17] and micro-controllers [MGB15]. Paramasivam et al. [PHEC17] took advantage of "*CustomPrograms*" which is based on Google Blockly to design a BBPE in order to enable young students with disabilities (e.g., Attention Deficit Disorder, Asperger's Syndrome, and other autism spectrum disorders or learning disabilities) to program Clearpath Turtlebot. Likewise, Martínez et al. [MGB15] took advantage of "*BlocklyDuino*" to design a BBPE in order to enable preschool and elementary school children to program and control the behavior of Arduino boards. The results reported by these studies are encouraging, as they were successful in establishing confidence among young students that programming is interesting. Nevertheless, how to best support young female students by BBPEs to develop basic programming skills and encourage them for being interested in programming is still a growing research area, where the open question remains.

6.2.1 Summary

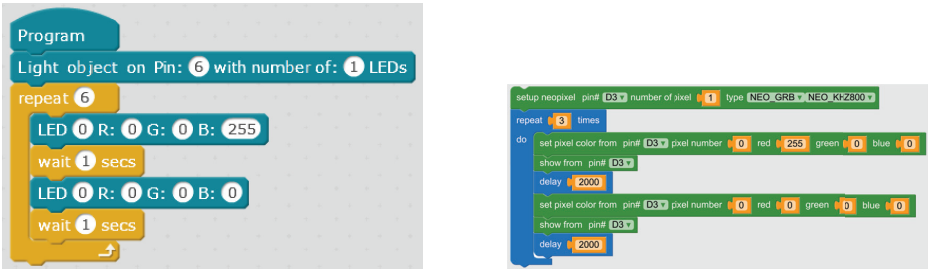
In contrast to a large number of previous studies, we seek to investigate the differences between two widely used block-based programming editors: Scratch and Google Blockly in terms of acquisition of basic programming skills and improving young female students' attitudes towards programming. Similarly, we aim to foster programming skills and interest among the students in order to increase their understanding of the programming side of computer science and motivate them to take part in future computer science education and digital society. To this end, we set up two non-formal programming training sessions independent of the students' regular curriculum and outside their schools. We aim to provide opportunities for young female students in order to begin with basic programming activities and program micro-controllers to control LED lights using two BBPEs which are designed based on Scratch and Google Blockly.

6.3 Overview of Block-based Programming Environments (BBPEs)

In this section, two BBPEs were utilized to enable young female students to learn and create programs for micro-controllers (Arduino or WeMos board) to control LED lights. In that respect, *mBlock* (see Figure 6.1a) and the MpB (see Figure 6.1b) were used in two non-formal programming training sessions. MBlock was used together with Arduino boards, while the MpB was used together with WeMos boards. Different programming language features like variables, data types, control-flow statements, functions, and operators are included as block-shaped elements in both programming environments. Students author programs in these environments by dragging-and-dropping blocks. In addition to the blocks representing programming features, primitive Arduino behavior is wrapped in a set of predefined blocks. Students create programs simply by snapping blocks together. The Arduino code is generated from the blocks in the background (visible in another panel), ready for execution. The possibility of zooming in and out on blocks is given, meaning that the scale of blocks can change by the mouse scroll wheel or the zoom gesture on a track-pad. This enables students to see the whole sequence of blocks if needed. Here, an overview of the design of both programming environments is provided.

6.3.1 MBlock

MBlock is based on Scratch [RMMH$^+$09], and it is primarily designed for inexperienced learners and children to learn and write programs for micro-controllers and



(a) (b)

Figure 6.2: A sample of execution blocks for the (a) *mBlock*, and (b) MpB.

robots. In micro-controller mode (Arduino mode), *mBlock* allows young students to use a visual block-based interface that comprises a full vision of the blocks (Block Panel), block categories (Toolbox), code syntax (Code Panel), and output of the code (Output Panel). Students can track the compile and upload process of the code into the micro-controller as well as the errors and Serial Monitor output in the Output Panel. In addition to blocks representing the programming features, other blocks are designed and developed to enable the students to work with LED lights, which we discuss in the following. The starting block is labeled as `program` that always needs to be the first block. The code syntax which is nested in other blocks only appears in the Code Panel when they are connected to the `program` block. Furthermore, the students can define how many LED lights are connected to a pin via the `light object` block. It is labeled as `light object on Pin <X> with number of <X> LEDs`. The two `<X>` are input field numbers, and they are filled in with default arguments to support the understanding of the block for the students. The color of LED lights can change through the `LED` block that is labeled as `LED <X> R <X> G <X> B <X>`. Similarly, the four `<X>` are input field numbers, and they are filled in with default arguments. The LED number filled in with 0, which always refers to the first LED, and the color is set to be white as default. This block also includes the show LED command in order to colorize the LED light (see Figure 6.2a).

6.3.2 The Micro-Controller Part of BEESM

The MpB is based on *BEESM*, which is built with the Blockly library [Fra14], and it enables inexperienced and young learners to learn and create programs for smart environments, micro-controllers, and mobile robots. The design and additional features of *BEESM* can be found in greater detail in Chapter 3 on this

thesis. For this study, we used the micro-controller part of *BEESM* and manipulated its user interface to enable our target students to have a full vision of the blocks (Block Panel), block categories (Toolbox), code syntax (Code Panel), and output of the code (Output Panel). Similar to *mBlock*, the Output Panel shows the compile and upload process of the code into the micro-controller as well as all return values and errors for debugging purposes. To enable the students to work with LED lights, we designed and developed other blocks in addition to blocks that represent the programming features which are discussed as follows. The first block which is needed in order to define how many LED lights are connected to a pin is called `setup neopixel`. It is labeled as `setup neopixel pin# <X> number of pixel <X>`. The color of LED lights can change through the `set pixel color` block which is labeled as `set pixel color from pin# <X> pixel number <X> red <X> green <X> blue <X>`. Similar to the *mBlock*, all `<X>` values are input field numbers which are filled in with default arguments to support the understanding of the block for the students. The pixel number filled in with 0, which always refers to the first LED, and the color is set to be white as default. Furthermore, in order to colorize the LED light, the show LED command is encapsulated into another block, which is called `show color`, and it is labeled as `show from pin# <X>`. The pin number is always filled in with 1, which refers to the first pin in micro-controllers (see Figure 6.2b).

6.3.3 Main Differences of the Two BBPEs

For a usability analysis, see Holwerda and Hermans [HH18] for a discussion on the differences between the two popular block-based programming editors: Scratch and Blockly. This usability analysis aimed to identify generic aspects of their user interface, and if they effectively fulfill their purpose to facilitate programming for inexperienced young learners. In this respect, the authors mentioned that a larger section for blocks could improve the visibility of finding and reading blocks in the program. However, the Blockly-based tool which is used (ArduBlockly [PA19]) does not support zooming via mouse scroll wheel or the zoom gesture on a trackpad, and it is only possible through the zooming buttons in Block Panel. It is also addressed that dragging a block out of a sequence of blocks will move all other blocks below with it in both editors. This will make manipulating of code structure more difficult for the users. Furthermore, it is suggested to have a search option to enable users looking and finding the right block in both Blockly and Scratch. However, the authors discuss that additional editor features may clutter the interface (both visually and cognitively) for adult novices and more specifically, for young learners in school. The main remaining differences between our two BBPEs are (see also Figure 6.1 and Figure 6.2) as follows:

- *The Block Panel:* the MpB contains a smaller Block Panel and larger Code Panel than *mBlock*.
- *The Code Panel:* the MpB enables students to modify the code that generates from the blocks directly in the Code Panel while in *mBlock* they need to open the code in Arduino IDE in order to modify it.
- *The Toolbox:* the Block Panel in the MpB contains the Toolbox, like a menu, that displays different categories for blocks. A set of blocks within a category is displayed temporary when students click on the category, while in *mBlock*, blocks within a category are displayed lasting when they click on the category.
- *How the blocks are shaped:* the structure of blocks can change using a pop-up panel (e.g., adding an `else-if` to an `if` block) in the MpB while we do not have this feature in *mBlock*.
- *How text codes are encapsulated in different blocks:* for instance, students need to use a `program` block that includes all libraries to start the program in *mBlock*, while in the MpB, necessary libraries are included in the corresponding blocks. Furthermore, in the MpB, a `display` block is needed in order to colorize the LED light, while in *mBlock*, it is nested in an *LED* block.

6.4 Methodology

In order to understand the impact of the two block-based programming editors—namely Scratch and Google Blockly—on young female students' programming skills and their attitudes and perceptions of programming, we conducted two user studies comparing *mBlock* to the MpB. In this respect, the students are welcomed to different research centers to learn programming in non-formal programming training sessions that are outside their school environments and not part of the regular school curriculum. This section begins with the study design and strategy for collecting and analyzing data, then information about the participants is presented; this section concludes with the procedure of the study.

6.4.1 Study Design and Data Collection Strategy

In this study, we use and compare two BBPEs in two non-formal programming training sessions with two groups of young female students (i.e., aged between 10 and 14 years old). The training sessions were held in the premises of the University of Bremen. All used equipment—computers, micro-controllers, and LED lights—were provided by the German Research Center for Artificial Intelligence (DFKI), and the

group of Cognitive Neuroinformatics (CNI) [1]. The BBPEs enable students to focus on programming structures and principles, the main computational concept which was taught and exemplified through the BBPEs. In general, 24 students attended the two programming training sessions (12 students each). In one training session, 12 students used *mBlock* (mBlock-group), and the MpB was used in the second training session (beesm-group) by the other 12 students.

In both programming training sessions, a pre- and post-questionnaire was used in order to collect data with respect to the young female students' attitudes and perceptions of programming, their prior programming experience, and their age group. The acquisition of basic programming skills among the students was assessed, using a pre- and a post-programming question. Pre- and Post-questionnaires, as well as pre- and post-programming questions can be found in the appendix B; all translated from German to English.

In the following, we describe the pre- and post-questionnaire as well as the pre- and post-programming questions in both programming training sessions.

Pre-questionnaire. In each training session, students received a pre-questionnaire. Four attitudinal questions were asked in order to find out the students' attitudes and perceptions of programming. These questions are based on the attitudinal questions which were used in Weintrop and Wilensky [WW17b] and were adapted for the needs of this study. Students' confidence, enjoyment, perceived difficulty, and interest in future programming learning opportunities were evaluated using these questions. In that respect, students were asked to rate the questions "*do you think you are good at programming?*", "*do you think programming is fun?*", "*do you think programming is difficult to understand?*", and "*would you like to learn how to program?*", using a 5-point Likert scale (with 1 "*no, not at all*", 5 "*yes, very much*", and 0 "*I do not know*"). Furthermore, they were asked to determine their prior programming experience with BBPEs using the "*yes*" or "*no*" question "*have you ever worked with a block-based programming environment?*". Then, we asked them to indicate whether they can program on a scale of 1 to 5, with 1 "*no, not at all*", and 5 "*yes, very good*", using the question "*can you program?*".

Post-questionnaire. At the end of each training session, students took the post-questionnaire. It was composed of the same attitudinal questions as the pre-questionnaire, just with different words for two questions; "*do you think programming is difficult to understand?*" changed to "*do you think programming is difficult?*",

[1] We make our training sessions materials available at https://github.com/projekt-smile/Smartes-Stimmungslicht, and at https://github.com/projekt-smile/Smarter-Bilderrahmen

and "*would you like to learn how to program?*" changed to "*would you like to learn better how to program?*".

In addition to the attitudinal questions, five questions were asked in order to measure the students' experiences with the two BBPEs in terms of their ease-of-use. The Students were required to rate the question "*I think the programming environment is easy to use.*", using a 5-point Likert scale (with 1 "*strongly disagree*", 5 "*strongly agree*", and 0 "*I do not know*"). The question "*do you find it easy to program with blocks?*" was also asked, using a 5-point Likert scale (with 1 "*no, not at all*", 5 "*yes, very much*", and 0 "*I do not know*"). They were then asked to rate (i) if they paid attention to the code that is generated matching the blocks, (ii) if they think the function "*edit code*" is helpful to better understand their program, and (iii) if they find the Output Panel helpful to understand their program. The scores for these three questions were calculated, using a 5-point Likert scale (with 1 "*no*", 5 "*yes*", and 0 "*I do not know*").

Two additional questions were added to the post-questionnaire for this study. The students were asked, "*do you think it's helpful if you program a real object? E.g. the LED light and the micro-controller*", to be answered on a 5-point Likert scale (with 1 "*no, not at all*", 5 "*yes, very much*", and 0 "*I do not know*"), and they were asked about their preference of programming with blocks or direct with code syntax, using a 5-point Likert scale (with 1 "*direct with code*", 5 "*with blocks*", and 0 "*I do not know*").

Programming questions. To validate the students' answers with respect to their prior programming experience and to analyze the acquisition of basic programming skills, in both training sessions, students were asked to complete two programming questions. In this respect, a pre-programming question right after the pre-questionnaire, and a post-programming question right after the post-questionnaire were completed. The programming concepts are extended by introducing these programming questions to the students. In each pre- and post-programming question, block-shaped elements were designed independent of the two BBPEs in order to test how well the students acquire the basic programming skills which were taught during the programming training sessions. For instance, see Figure 6.3a, and Figure 6.3b for the block-shaped elements in pre- and post-programming questions in the beesm- and mBlock-group, respectively. However, in each training session, one block (block number 13) was designed similar to what the students saw and used in the corresponding programming environment. In this regard, students needed to use a `Program` block in order to start the program in *mBlock*, and a `display` block in order to colorize the LED light in the MpB (see Figure 6.3).

The pre-programming question in the beesm-group was to program the micro-controller to make one LED light blink in red for 3 times with 2 seconds delay in

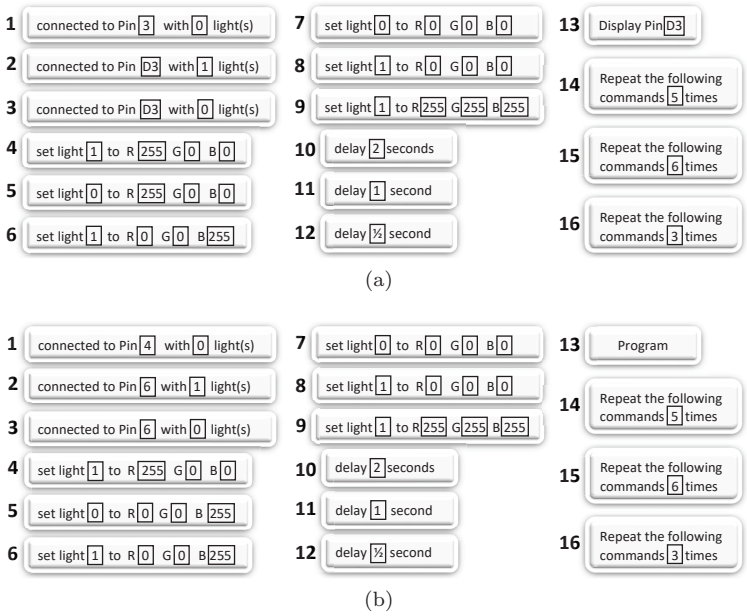


Figure 6.3: Block-shaped elements in (a) pre-programming question in the beesm-group; (b) post-programming question in the mBlock-group; both translated from German to English.

between when the light is connected to the micro-controller on *Pin D3*. The post-programming question in the beesm-group was to program the micro-controller to make one LED light blink in blue for 6 times with 1 second delay in between when the light is connected to the micro-controller on *Pin D3*. In the mBlock-group, apart from the pin number, which is *Pin 6*, similar pre- and post- programming questions were asked from the students. In each programming question, students were asked to select a set of blocks and identify the order of them in a correct logical way based on the question. Students were also notified that some blocks might appear more than once and some may not even be needed in their program.

The pre- and post-programming questions are slightly different from each other in both training sessions. This counterbalance design of questions ensures that students read the questions carefully and identify the order of blocks based on the question. Furthermore, these questions represent realistic programming problems for a micro-controller and an LED light, as colorizing the light is core to the function and use of micro-controller together with one LED light.

For each programming question, we collected the solution made by the student using the blocks and evaluated them by the 10-point grading rubric (see Table 4.1). Each solution was scored by two researchers in order to ensure consistent grading.

6.4.2 Participants

Two user studies were conducted with 24 young female students without any prior programming experience (10 to 14 years old) from several German secondary schools. The schools' headmasters and teachers were contacted and informed about our programming training sessions. Students and their parents were then announced by their school to register for one training session, meaning the students who participated in this study were self-selected and were interested in having programming activities and learning programming.

A total of 12 female students participated in each programming training session. In the mBlock-group, the average age of the participants was 12.67 years ($SD = 0.78$). In the beesm-group, the average age of the participants was 12.58 years ($SD = 1.24$). Although participants in the beesm-group are younger than participants in the mBlock-group, an analysis of variance (ANOVA) showed no significant difference between the groups, $F < 1$.

With respect to the students' prior programming experience, all students indicated that they have no experience in working with any BBPEs. Furthermore, when the students were required to rate whether they can program or not, only two of them in the beesm-group answered, "*no*" and the rest of the students answered, "*no, not at all*". Nevertheless, no significant difference was observed ($F(1,22) = 2.20$, $p = 0.15$). Thus, the level of prior experience was not included in our analyses.

6.4.3 Procedure

In both programming training sessions, we followed the same procedure (see Figure 6.4). The duration of each training session was 130 minutes. At the beginning of both training sessions, each participant was assigned to one computer, one micro-controller (Arduino or WeMos board) and one LED light in order to minimize the distraction of participants. Each computer had an installed version of the corresponding programming environment. In the mBlock-group, Arduino boards were used together with mBlock, and in the beesm-group, WeMos boards were used together with the MpB.

All participants were introduced to micro-controllers, LED lights, and they were shown how to connect them for 10 minutes. Afterward, the participants received the pre-questionnaire; they were asked to determine their prior programming experience and rate their perception of programming using the four attitudinal questions.

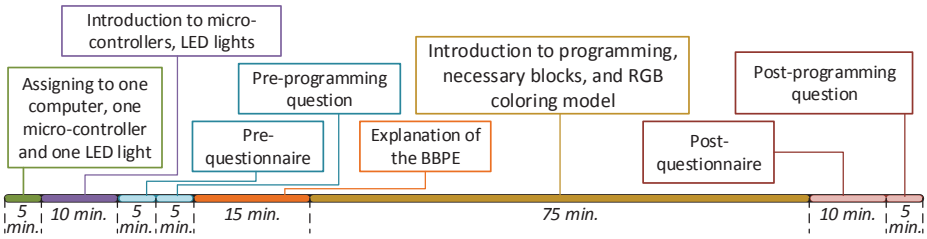


Figure 6.4: Procedure of the programming training sessions.

Each participant was then asked to complete the pre-programming question, which required them to select a set of blocks and write their numbers in a correct logical way based on the question. The participants were then trained according to the interface of each BBPE, different panels, buttons, and their functionalities for 15 minutes. Then, during the allocated time of 75 minutes, all participants were introduced to general features of programming (e.g., variables and loops), necessary blocks to control an LED light, and the RGB coloring model. One female instructor led each training session. In this regard, in each training session, an oral explanation was given, using prepared slides based on each BBPE. Additionally, we used supplementary documents—including an explanation of necessary blocks and RGB coloring model—in order to help our students to work with the corresponding BBPE. This also allows us to minimize and control the instructor effects. Both female instructors have a computer science background and experience in working with young students. During the allocated time, participants were enabled to program their micro-controller, using the corresponding BBPE in three learning steps. These steps respectively were (i) colorizing one LED light with either red, green or blue color, (ii) colorizing one LED light with two arbitrary colors and write down the correct value of red, green and blue colors, and (iii) letting one LED light blink for a random number of times and seconds delay in between. The instructor helped participants at their desk during the 75 minutes to ensure that the participants had faced no major problems. At the end of each training session, the post-questionnaire was given to the participants in order to ask them to rate their perception of programming and to rate their experience with the corresponding BBPEs. Each training session ended with the post-programming question.

6.5 Experimental Evaluation

The results section is divided into three parts. First, results from an analysis of the programming part of the study are presented, reporting the students' performance

on each pre- and post-programming question. Second, results from an analysis of the pre- and post-questionnaires are presented, looking at young female students' confidence, enjoyment, perceived difficulty, and interest in future programming learning opportunities. Finally, we report on results from an analysis of the post-questionnaire, focusing on students' experience with respect to ease-of-use of corresponding BBEPs. Additionally, in this part, students' preferences of programming with blocks or direct code, as well as their thought of being able to see the impacts of their programs on a real object is reported.

The following analyses were computed as one-factor analysis of variance, with the factor *mBlock vs. MpB* (ANOVA). Paired-samples t-test was also used to show the differences within each group of students from the beginning towards the end of each training session. T-test can be used to determine the differences among *Means* of two independent groups, while ANOVA can be used to show the differences between the variation of the *Means* and within each *Mean* (ANOVA is robust to the assumption of normality). With this regard, ANOVA is valuable because it examines the variation between and within the *Means*, while t-test just compares the *Means*. Firstly, we used ANOVA between each group of students and test our data for the normality and transformation. Then, within each group of students and between each test time, we used paired-samples t-test in order to show the *Mean* differences between their answers at the beginning and at the end of the training sessions.

6.5.1 Acquisition of Programming Skills

To understand how students' programming skills are influenced by the two BBPEs (*mBlock* and the MpB) in our non-formal programming training sessions, students' performance on the pre- and post-programming questions were analyzed.

With respect to the programming questions (see Figure 6.5), in the pre-programming question the students in mBlock-group performed better than the students in beesm-group, $M = 2.00 \mid SD = 0.00$, and $M = 1.33 \mid SD = 1.30$, respectively. However, no significant difference occurred, $F(1,22) = 3.14$, $p = 0.09$. In the post-programming question, an ANOVA showed that the students in mBlock-group ($M = 6.83 \mid SD = 1.80$) performed significantly better than the students in beesm-group ($M = 3.17 \mid SD = 1.59$), $F(1,22) = 28.02$, $p < 0.001$. Descriptively, although the students in the mBlock-group indicated a higher level of performance before working with the programming environment, they performed significantly better after working with the programming environment compared to the students who were in the beesm-group.

Focusing on the average performance in pre- and post-programming questions shows that their performance increased in both mBlock-group and beesm-group.

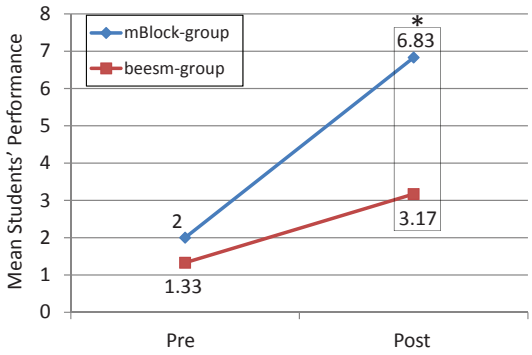


Figure 6.5: Students' performance on the pre- and post-programming questions.

A paired-samples t-test showed that in mBlock-group, students performed significantly better in the post-programming question than in the pre-programming question, $t(12) = 9.30$, $p < 0.001$ | $MD = 4.83$. Similarly, in beesm-group, students' performance significantly increased in the post-programming question compared to pre-programming question, $t(12) = 4.75$, $p = 0.001$ | $MD = 1.83$.

6.5.2 Attitudes and Perceptions of Programming

To understand how students' attitudes and perceptions of programming are affected by the two BBPEs (*mBlock* and the MpB) in our non-formal training sessions, we analyzed scores from the pre- and post-questionnaires. All answers were coded with 1 "*no, not at all*", 5 "*yes, very much*", and 0 "*I do not know*". Please note that students who responded "*I do not know*", are not included in our analysis.

Confidence. Concerning the students' confidence (see Table 6.1), we assessed their responses to the following question in the pre- and post-questionnaires: "*do you think you are good at programming?*". Seven students in each beesm- and mBlock-group responded, "*I do not know*" in the pre-questionnaire. In the post-questionnaire, this number decreased to one in the beesm-group and four in the mBlock-group. In the pre-questionnaire, students in the beesm-group indicated a slightly higher level of confidence compared to the students in the mBlock-group. However, in the post-questionnaire, students in the mBlock-group indicated (descriptively) a higher level of confidence compared to students in the beesm-group.

In both mBlock- and beesm-group, the level of confidence in programming is higher for students in the pre-questionnaire compared to the post-questionnaire.

Table 6.1: Students' Attitudes and Perceptions of Programming

	Pre-questionnaire		**Post-questionnaire**	
Questions	*M (SD)*	*ANOVA Results*	*M (SD)*	*ANOVA Results*
Confidence in mBlock-group	2.40 (0.55)	F <1	3.63 (0.74)	F(1,17) = 3.16, p = 0.09
Confidence in beesm-group	2.60 (1.52)		2.73 (1.27)	
Enjoyment in mBlock-group	4.64 (0.50)	F(1,20) = 1.87, p = 0.19	4.58 (0.51)	F(1,22) = 1.38, p = 0.25
Enjoyment in beesm-group	4.18 (0.98)		4.17 (1.11)	
Difficulty in mBlock-group	3.30 (0.67)	F(1,19) = 3.54, p = 0.08	2.33 (1.15)	F(1,22) = 2.70, p = 0.12
Difficulty in beesm-group	3.91 (0.83)		3.08 (1.08)	
Interest in mBlock-group	4.75 (0.44)	F(1,22) = 1.16, p = 0.29	4.33 (0.49)	F(1,21) = 1.54, p = 0.23
Interest in beesm-group	4.92 (0.29)		4.64 (0.67)	

M: Mean **SD**: Standard Deviation **F**: F-distribution **p**: p-value

With respect to the students who responded, "*I do not know*", we focus on the changes in their level of confidence between the pre- and post-questionnaire. In the beesm-group, six out of seven students had an idea in the post-questionnaire, and in general, they indicated a medium level of confidence, $M = 3.33 \mid SD = 1.21$. In the mBlock-group, four out of seven students had an idea in the post questionnaire, and in general, they indicated a higher level of confidence, $M = 4.00 \mid SD = 0.82$. However, no significant difference was observed, using an ANOVA ($F < 1$).

Enjoyment. With respect to the enjoyment of programming (see Table 6.1), the responses to the following question in the pre- and post-questionnaires were assessed: "*do you think programming is fun?*". Only one student in each of the beesm- and mBlock-group responded "*I do not know*" in the pre-questionnaire. In both pre- and post-questionnaire, the level of enjoyment of programming is slightly higher for the students in the mBlock-group compared to the students in the beesm-group.

Perceived difficulty of programming. The third attitudinal question is whether the students think programming is difficult or not (see Table 6.1). To calculate a measure of the difficulty of programming, students responded to the following question: "*do you think programming is difficult to understand?*" (or "*do you think programming is difficult?*" in the post-questionnaire). Only one student in the beesm-group and two students in the mBlock-group responded "*I do not know*" in the pre-questionnaire. Descriptively, in the pre-questionnaire, the programming difficulty level is higher for students in both mBlock-group and beesm-group. Furthermore, in both pre- and post-questionnaire, the programming difficulty level is higher among students in the beesm-group. We focus on the score of difficulty of programming within the beesm- and mBlock-group, among the students who answered the question in both pre- and post-questionnaires. A paired-samples t-test showed that in the mBlock-group, students found programming significantly less

difficult in post-questionnaire in comparison with pre-questionnaire, $t(9) = -2.70$, $p = 0.024$, $MD = -1.10$. However, no significant results occurred in beesm-group, $t(10) = -1.70$, $p = 0.12$, $MD = -0.82$.

Interest. The last attitudinal question is whether the students' interest in learning programming in the future is affected or not (see Table 6.1). In order to calculate a measure of it, students responded to the following question: "*would you like to learn how to program?*" (or "*would you like to learn better how to program?*" in the post-questionnaire). Only one student in the beesm-group responded, "*I do not know*" in the post-questionnaire. Descriptively, in both pre- and post-questionnaire, students in the beesm-group indicated a higher level of interest compared to students in the mBlock-group. Furthermore, in both mBlock-group and beesm-group, students showed a higher level of interest in the pre-questionnaire compared to the post-questionnaire. Focusing on students' interest within the beesm- and mBlock-group, a paired-samples t-test showed that the students' willingness to learn programming dropped in mBlock-group towards the end of the training session; accordingly, it was just barely missed the level of significance, $t(10) = -2.16$, $p = 0.054$, $MD = -0.42$. However, the test showed that the decrease in students' interest in the beesm-group was not significant towards the end of the training session.

6.5.3 Programming Experience

The post-questionnaire included five questions, asking students to reflect on how they perceive the ease-of-use of corresponding BBPEs (see Table 6.2). Furthermore, students were required to answer another question, concerning their preference for programming with blocks or with code. Finally, they were asked to indicate whether being able to see the impacts of their program on a real object is helpful or not. Please note that students who responded "*I do not know*", are not included in our analysis.

With respect to the ease-of-use of the two BBPEs, on average, the students showed broad approval to *mBlock*, while the students were undecided about the MpB. Accordingly, an ANOVA yielded a significant result (see Table 6.2). Only one student in the beesm-group responded, "*I do not know*" to this question. With respect to finding programming easy with blocks, on average, the students found programming with blocks significantly easier in *mBlock* compared to the students in the MpB (see Table 6.2). Similarly, one student in the beesm-group responded, "*I do not know*" to this question. Furthermore, the students were undecided and indicated a low level (especially in mBlock-group) of paying attention to the code which was generated matching the blocks. Six students in the mBlock-group and nine students in the beesm-group responded to this question. The students found

Table 6.2: Students' Experiences of Using Block-based Programming Environments

Questions	**mBlock-group** *M (SD)*	**beesm-group** *M (SD)*	*ANOVA Results*
I think the programming environment is easy to use. (1 = "*strongly disagree*", 5 = "*strongly agree*")	4.58 (0.67)	3.82 (0.98)	F(1,21) = 4.85, **p = 0.039
Do you find it easy to program with blocks? (1 = "*no, not at all*", 5 = "*yes, very much*")	4.00 (0.74)	3.18 (0.75)	F(1,21) = 6.93, **p = 0.016
Did you pay attention to the code that is generated matching the blocks? (1 = "*no*", 5 = "*yes*")	2.83 (2.04)	3.22 (1.56)	F <1
Do you think that the function "*edit code*" is helpful to better understand your program? (1 = "*no*", 5 = "*yes*")	3.43 (0.79)	3.75 (1.26)	F <1
Do you find the error messages in the Output Panel helpful? (1 = "*no*", 5 = "*yes*")	3.33 (0.52)	3.40 (0.89)	F <1

M: Mean **SD**: Standard Deviation **F**: F-distribution **p**: p-value ****p** < **0.05**: Significant Difference

that being able to use the "*edit code*" function and to see error messages in the Output Panel is helpful to understand their own program (see Table 6.2). However, only 11 students answered to each of these two questions. Seven students in the mBlock-group and four students in the beesm-group responded to the question with respect to the helpfulness of being able to use the "*edit code*" function. Six students in the mBlock-group and five students in the beesm-group responded to the question regarding the helpfulness of being able to see error messages in the Output Panel. Although the students in the beesm-group rated all the three questions higher than the students in the mBlock-group, no significant results occurred, all $F < 1$.

Concerning the preference for programming with blocks or directly with code, on average, the majority of students indicated an opinion strongly towards programming with blocks. The students in mBlock-group indicated a slightly higher preference toward blocks ($M = 4.92 \mid SD = 0.29$) compared to the students in beesm-group ($M = 4.44 \mid SD = 0.88$). No significant difference were obtained, using an ANOVA ($F(1,19) = 3.05$, $p = 0.10$). Only three students in the beesm-group responded, "*I do not know*" to this question and they are excluded from our analysis.

With respect to the question of whether or not students think it is helpful to program a real object (e.g., LED light and micro-controller), they found that being able to see the impacts of their program on a real object is helpful. In this regard, students in the beesm-group indicated the higher level ($M = 4.55 \mid SD = 0.52$) compared to students in the mBlock-group ($M = 3.73 \mid SD = 1.27$). However, an ANOVA shows that it just barely missed the level of significance, $F(1,20) = 3,89$, $p = 0.062$. Only two students (one student in each group) responded "*I do not know*" to this question, and thus, they are excluded from our analysis.

6.6 Integration and Discussion

We now discuss the results which were presented in the previous section. In this respect, we begin with our findings based on the research question presented in Section 6.1. Then, we review the limitations and discuss future work of the study that should be taken into account.

6.6.1 Findings

One of the main contributions of this study is showing the potential of using two popular block-based programming editors (Scratch and Google Blockly) in non-formal programming training sessions in order to support the acquisition of basic programming skills among young female students. In this respect, one hypothesis we had in this study was that different BBPEs (*mBlock* and the MpB) have different influence on young female students' performance who have no prior programming experience. Results show that in both mBlock-group and beesm-group, the performance of students was significantly higher in the post-programming question compared with the pre-programming question. This supports the results from prior research that showed by designing introductory programming environments based on block-based programming, we could help young students to have better performance [WW15,MCK17]. Results also show that in both pre- and post-programming questions, the performance of students who worked with *mBlock* (which is based on Scratch) was higher than those who used the MpB (which is based on Google Blockly). Having a closer look into the performance on post-programming question reveals that in mBlock-group, students highly tended to solve the programming question and their performance highly improved in comparison to the students in beesm-group. This finding supports the idea that designing introductory programming environments based on Scratch could help young female students to gain basic programming skills, in particular, when they are indeed new to programming. However, this result could also be caused by the differences in students' socio-economic status and their background in each training session.

This study also reports on young female students' attitudes and perceptions of programming, where the findings were less clear. The results show that before and after the training session, students who used *mBlock* indicated (descriptively) a higher score for the enjoyment of programming. Likewise, students in the mBlock-group showed (descriptively) a higher level of confidence in programming after the training session, while it was slightly lower before the training session compared with the students in the beesm-group. Furthermore, students in the beesm-group indicated (descriptively) a higher level of interest in taking part in the future programming opportunities before and after the training session compared with the

students in mBlock-group. In the mBlock-group, the students' level of interest was decreased after the training session compared with before the training session. With respect to the difficulty of programming, our findings show that the difficulty level of programming was higher (descriptively) before the training session compared with after the training session in beesm-group. In mBlock-group, the difficulty level of programming was significantly dropped after the training sessions compared with before the training session. Our findings also show that in the beesm-group, students found (descriptively) programming harder both before and after the training session.

With respect to the ease-of-use of the two BBPEs, the results showed that the students in mBlock-group found blocks significantly easier to program compared with the students in beesm-group. Furthermore, when asked to indicate how easy was the use of corresponding BBPEs, the students in mBlock-group found *mBlock* significantly easier to use in comparison to the students in beesm-group who used the MpB. This result is in line with results from [HH18] that students found programming easier with Scratch-based environments, as the visibility of finding and reading blocks is higher than Blockly-based environments. In contrast, when the students asked specific questions about the demonstration of error messages in the Output Panel, usage of "*edit code*" function, and paying attention to the generated code syntax matching the blocks, students in beesm-group rated them (descriptively) higher than the students in mBlock-group. This result led us to change the *BEESM* initial user interface in the next two programming workshops (Chapter 7 and Chapter 8). Thus, the Block Panel was extended in order to improve the visibility of finding and reading blocks in the program for the students.

Concerning the subjective questionnaire data, in line with findings from [WW17b], the students in both beesm- and mBlock-group largely prefer working with blocks compared to programming code syntax. Additionally, they found it helpful to code and see the impacts of their program in a real object. This is in line with findings from [Bei05, MCK17] that showed real objects could stimulate students' interest, and motivate them to begin with programming activities.

All in all, the findings show that young female students who used a BBPE based on Scratch (in this case, *mBlock*) performed better on programming questions and showed a higher level of ease-of-use in programming with blocks in *mBlock*. This suggests a Scratch-based design for a productive environment for supporting female students who have no prior programming experience to gain basic programming and computational skills. At the same time, the findings that the students using a BBPE based on Google Blockly (in this case, the MpB) show a higher level of interest in taking future programming opportunities. This indicates a gap between what the students view themselves in programming with different types of BBPEs and the future programming experience with these environments.

6.6.2 Limitations

While we tried to make the conditions across the two non-formal programming training sessions as similar as possible, there were some differences which can be introduced as limitations of this study. For instance, we used *mBlock* together with Arduino boards, but due to technical reasons, the MpB is used together with WeMos boards. Thus, it introduces a difference that may influence the findings of this study. However, there was no evidence that this difference has contributed to significantly differing our target students' experiences and change their level of acceptance for programming.

Another limitation of this study is related to the number of programming tasks and period of each training session. For example, findings are limited to the diversity of the programming tasks that young female students were required to perform to a larger scale of programming activities and computational skills. Using the micro-controllers and LED lights that are used in this study, students can perform larger and more complicated tasks. For example, they can use more LED lights and sensors to create colorful and animated LED picture frames. While we intend on introducing young female students who have no prior programming experience to begin with programming activities in a non-formal learning environment, this is relatively narrow functionality for a micro-controller, and thus, for the programming tasks. In this regard, there is still work to be done to verify the outcome of this study when the programming workshop is longer (including more number of training sessions) and when programming tasks become more diverse and complicated among the students with and without prior programming experience. Thus, in the following chapters of this thesis (Chapter 7 and Chapter 8), we present the results of longer non-formal programming workshops including more diverse programming tasks. To this end, the trajectories of attitudes towards programming and performance is investigated during these workshops among both experienced and inexperienced young learners, especially female students. Furthermore, we explored how Scratch and Google Blockly perform to foster young female students' programming skills and leverage their interest in programming. However, there is still an open question we would like to explore in the future: what are the reasons behind the students' preferences for using Scratch-based environments, and for using Blockly-based environments.

The final limitation of this study relates to the students' prior programming experience, socio-economic status, age, and the number of participants. We would like to emphasize that our results might be affected by a lack of geographic and socio-economic diversity of students. Furthermore, our sample includes 24 female students without any prior programming experience (age between 10 to 14), which is a relatively small sample size to generalize our findings to a larger scale. Thus, we

look at these as a major concern, and we seek to address them in future iterations of this work. A second similar limitation is related to the control group. This is another avenue of future work to find out the impacts of visual block-based programming environments on students' attitudes and programming skills when young male students are targeted for such non-formal programming training sessions.

6.7 Conclusion

As the number of women in higher computer science education and society is lower than the number of men in most western countries, using block-based programming is an active area of research to make the programming side of computer science more interesting and engaging for girls. In this chapter, we presented two non-formal programming training sessions and a comparative study of how different types of BBPEs impact young female students' attitudes towards programming and their programming performance. In that respect, we explored how young female students use *mBlock* and a MpB which are based on Scratch and Google Blockly, respectively. Our findings indicate how young female students' performance, attitudes, and perceptions of programming can be affected by different types of BBPEs. This finding supports the idea of using Scratch in introductory programming environments in order to motivate young female students, in particular, those without prior programming experience to solve programming problems and gain basic programming skills. Furthermore, it shows that those students who used the MpB, which is based on Google Blockly indicated greater interest in future programming learning opportunities. Thus, it supports the claim that different BBPEs have a direct impact on young female students' performance and their attitudes towards programming. By studying under what condition, what type of BBPE has a better influence on young female students, we enhance our understanding to design introductory programming environments for them.

Given the decreasing presence of women in computer science society in most western countries, findings from this study are essential to ensure we are providing exposure to programming activities, preparing female students for future learning programming opportunities and motivating them to join the computer science society in the future. While many questions still remain on how to best introduce programming to female students, the findings of this study can help to inform other researchers and educators about the relationship between BBPEs, programming activities in non-formal learning environments, and young female students' experience and acceptance of programming.

Chapter 7

Students' Attitudes and Skills: Impacts of Smart Objects' Construction

In computer programming education, learning to program tangible objects has become a common way to introduce programming to young students. In an effort to address this intervention, scientific research has been done on the effectiveness of using tangible hardware platforms such as robots and wearable products to teach basic programming concepts to the students. However, there is a lack of research on how young students' attitudes and programming skills are influenced over time, when they learn to program tangible objects and make them smart.

In this chapter, we investigate the impacts of using a tangible everyday object and making it smart on young female students' attitudes towards programming and the acquisition of basic programming skills. During a 4-day non-formal programming workshop with 12 6^{th} grade female students, they were introduced to basic programming concepts, and learned how to apply them to turn a *houseplant* into a smart object. In a pilot study, we took advantage of the introduced block-based programming environment in Chapter 3 (*BEESM*) and analyzed the students' trajectories of attitudes towards programming and performance. In this respect, repeated open-ended qualitative questionnaires and programming questions were used throughout the workshop. The findings of this study contribute to our understanding of how making tangible everyday objects smart can support the development of a positive attitude and keep up of interest throughout a programming workshop among girls.

7.1 Introduction and Motivation

The number of formal and non-formal computer programming courses and workshops that aim to introduce programming and computer science to young students is growing. In computer programming education, the application of computing in reality, tends to be shown to students. In particular, allowing the students to learn the general purpose of programming and write code for tangible hardware platforms such as robots [MCK17, PHEC17, MGB15], smart homes [KD18], and wearable products [QBBD13, KDS09, KLS+14] were considered in previous work. Furthermore, visual block-based programming environments are widely used in the design of introductory programming courses [WW17b, KLS+14, MGB15]. These environments are beneficial for learning to code and starting with programming activities, especially for young students [Wei19].

Despite the growing use of tangible objects and block-based programming, relatively little empirical study has been done to understand the impacts of smart objects together with block-based programming on young students' interest in programming and computer science in general. More specifically, it is not clear how teaching basic programming concepts to young students, and letting them implement these concepts in a tangible object and make it smart can improve their attitudes towards programming over time. Previous research addressed that introducing young students to new technologies supports learning programming and stimulates interest in computer science [MGB15, PHEC17, MCK17]. Although research on teaching programming to young students is vast (e.g., [MCK17, Kaf16, Cof17, Kal15, KM16]), less is known about the students' trajectories of performance and attitudes towards programming in the context of tangible smart objects. In addition, as it is mentioned in previous chapter, most western countries, such as European countries have significant problems with the number of female graduates in the field of computer science. Thus, research is needed to offer insights into the impacts of embedding the construction of smart objects in programming courses, and into female attitudes towards programming and computer science more broadly. This chapter seeks to answer the following question in order to address the gap in previous research:

How do young female students' programming skills and attitudes towards programming change over time in the context of constructing smart everyday objects?

To answer this question, we present the result of a 4-day non-formal programming workshop with 12 6^{th} grade German female students (between 11 and 12 years old). We investigate the influence of using tangible everyday objects and making them smart on the development of a positive attitude towards programming among the students and on the improvement of their programming performance in a pi-

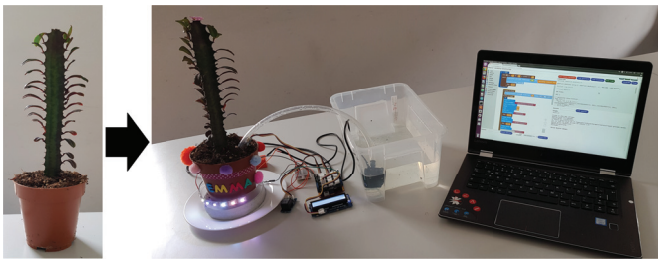


Figure 7.1: An example of one houseplant, at the beginning, and at the end of the workshop.

lot study. A block-based programming environment (*BEESM*) was employed to reduce the complexity of programming and facilitate it for the students to learn basic programming concepts and author programs. A *houseplant* was provided as an appropriate stimulus that enables the students to connect a micro-controller, different sensors (e.g., light, temperature, humidity, sound) and actuators (e.g., LED light, water-pump, mp3-player, RGB LCD) to it. The students can program the sensors and actuators in a way that they react to each other, and construct a *smart houseplant*. For instance, students can use a micro-controller (in this case, Arduino) and connect a humidity sensor, a relay, and a water-pump to it. Then, the students can program the micro-controller using block-based programming in order to enable the water-pump to pump water into the houseplant as soon as the sensor measures the humidity of the flower soil is below a certain degree (see Figure 7.1). It is a special feature of this study that we examine the path of students' attitudes towards programming and their performance based on repeated open-ended qualitative questionnaires and programming questions at the beginning, in the middle and at the end of the workshop.

The chapter begins with a review of previous work and how they employ tangible objects and hardware platforms to teach basic programming concepts together with block-based programming environments (Section 7.2). Then, we describe the study design in Section 7.3. We continue with our findings in Section 7.4, followed by a discussion of the implications of these findings in Section 7.5. The chapter closes with limitations and conclusions 7.6.

7.2 Related Work

The importance of learning computer programming has been shown, and it has been established as an area of research in computer science discipline [WW17b,

WH17, MCK17]. Young students become familiar with the use of technologies (e.g., smartphones, tablets, computers, etc.), while they do not have programming skills [MCK17,Kaf16]. In the CSE community, a large number of studies in the field of computing education highlighted the need to engage young students to learn basic programming concepts [MGB15, WW17b, WW15, MCK17]. In particular, they aim to motivate young female students learning the basics of computer programming and to enable them writing computer code [SB16, MCK17]. However, there is limited consideration of how to improve young female students' attitudes and computer programming performance when it is channeled through appropriate stimuli, such as construction of a smart object.

Tangible objects and block-based programming have been used [B$^+$15,RMB$^+$98], and their benefits in learning programming have been shown, especially for young students [MCK17, SKL16, MI18]. Findings show that their technological confidence benefited from block-based programming environments and tangibles, such as robotic computing platforms [MCK17,MGB15] and computational textiles [KLS$^+$14, QBBD13]. Merkouris et al. [MCK17] explored the benefits of learning to author programs for tangible hardware platforms such as robots and wearable computers in comparison to programming for desktop computers among young students. For this purpose, the authors used similar block-based programming environments (all based on Scratch [MRR$^+$10]) in order to measure attitudes and programming performance in formal classrooms. It was shown that students' performance in learning basic programming concepts were not affected by the tangibles, although they showed a higher intention of learning programming when they program the robots compared with the desktop computers. Concerning gender, the girls performed better in the programming tests than the boys, although they felt less confident than boys. Nevertheless, no information was provided on how the students' performance and attitudes changed over time from the beginning of the course towards the end of it.

The literature also reports that young learners learn better when they are engaged in designing and constructing visible objects such as interactive applications, animations, robots, and computational textiles [MCK17]. With respect to gender, according to [MCK17, BECC08, KDS09] computational textiles activities make use of soft everyday materials (e.g., design bag with colors and LED lights), which are meaningful and give forms of expression to young students who are not primarily interested in technology. In addition, finding shows that girls underestimate their computer abilities and they struggle with assembling and programming materials (e.g., motors and gears) in robotic courses. Therefore, they enter programming courses with less confidence than boys [NHCW04, GC02, GBBB19]. However, Nourbakhsh et al. [NHCW04] found that girls' confidence increased more than boys by the end of the robotic courses. Furthermore, Kelleher and Pausch [KP05] show that performance and interest in programming highly depend on time spent in program-

ming activities and prior programming experiences but not on gender. Nevertheless, research has not yet been conducted on the effectiveness of constructing smart everyday objects together with block-based programming on young female students' programming skills and attitudes towards programming.

7.2.1 Summary

Most interventions to teach computer programming with block-based programming environments and tangible objects achieved high success to establish confidence and engagement among young students. However, it is still required to understand more how the use of these objects during a programming course together with block-based programming fosters programming skills, as well as promotes positive attitudes towards programming and computer science in general. In this study, we investigate the impacts of programming a *houseplant* as a tangible object and making it smart on female students' attitudes, performance, and level of interest in programming over time.

7.3 Methodology

The goal of this study was to experimentally investigate the impacts of tangible objects and block-based programming environments on young female students' programming skills and attitudes towards programming. We conducted a pilot study with 12 6^{th} grade female students (11-12 years old) in a 4-day non-formal programming workshop. *BEESM* as a visual block-based programming environment, and a *houseplant* as the tangible everyday object were used. Three dimensions of students' attitudes were considered: confidence, enjoyment, and interest in future programming learning opportunities [WW17b]. The students' perception of using block-based programming and constructing a smart object was measured with three questionnaires. Furthermore, the performance of the students was assessed with three programming questions. The questionnaires and programming questions were given to students: (i) at the beginning of the workshop (pre questions), (ii) at the end of the second day when the students had learned programming concepts (intermediate questions), and (iii) at the end of the workshop, when the students had implemented their newly learned programming skills in the houseplant and made it smart (post questions).

7.3.1 Study Design and Data Collection Strategy

In a pilot study, we used the micro-controller part of *BEESM* in order to enable students to program a tangible everyday object (see Figure 7.2). We changed the

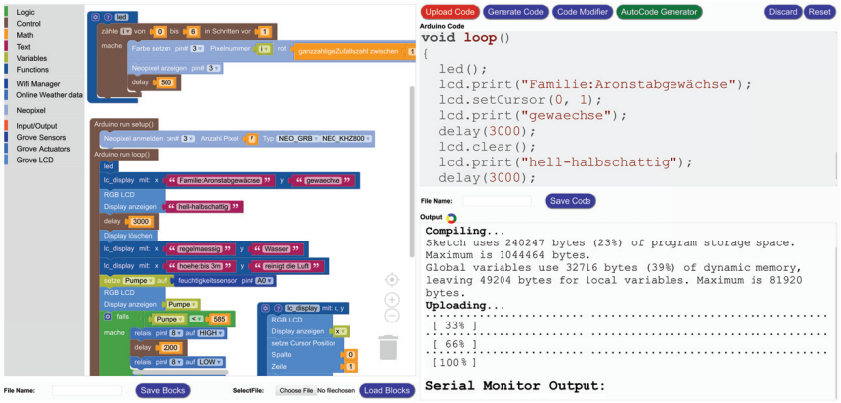


Figure 7.2: Screenshot of the programming environment interface, including the final program for a group.

initial *BEESM* user interface to allow our students to use three different panels and have a full vision of blocks (Block Panel), code syntax (Code Panel), and output of the code (Output Panel). Thus, we removed the 2D Graphical Panel, moved the Output Panel to the right side of the screen and extend the Block Panel in order to improve the visibility of finding and reading blocks in the program. Furthermore, the tangible object in this workshop was a *houseplant*. This was used as an appropriate stimulus to enable the students to connect a micro-controller, as well as different sensors and actuators, program them, and construct a smart object [1].

Pre, intermediate, and post questionnaires were employed to collect data concerning the students' attitudes and perceptions of programming, prior programming experience, and age group. The acquisition of basic programming skills was assessed, using a pre, an intermediate, and a post programming question. All Pre, Intermediate, and Post questionnaires and programming questions can be found in the appendix C; all translated from German to English. In the following, we describe the questionnaires and programming questions.

Pre questionnaire (PreQ). PreQ, which was distributed before the programming activities, consists of five open-ended questions to find out the students' attitudes towards programming and the programming workshop. With this regard, students'

[1] We make our workshop materials available at https://github.com/projekt-smile/Smarte-Pflanze-mit-Charakter

confidence, enjoyment, and interest in future programming learning opportunities were recorded. The students' confidence was asked through, "*how do you rate your programming skills?*" (Q1), and "*do you think you will be successful in this workshop?*" (Q2). The enjoyment was recorded using two questions "*I find programming...* (Q3)", and "*what are you looking forward to in this workshop?*" (Q4). The interest of students in learning programming was asked via the question "*how would you like to learn programming? why?*" (Q5). Furthermore, the students were required to determine their prior programming experience with block-based programming environments using the "*yes*" or "*no*" question "*have you ever worked with a block-based programming environment?*" (Q6).

Intermediate and post questionnaires (IntermediateQ and PostQ). IntermediateQ was distributed after learning basic programming concepts and activities in order to measure the students' attitudes towards programming. The students' perception of using block-based programming and constructing a computer system consists of micro-controller, sensors, and actuators were also considered. This questionnaire was composed of the same questions as the pre questionnaire, just with different words for two questions; Q2 changed to "*do you think you were successful in this workshop?*", and Q4 changed to "*what did you like/dislike about the workshop?*". Furthermore, the students were required to answer two additional questions. These questions were about the block-based programming, "*how do you like programming with blocks?*" (Q7); and programming the sensors and actuators, "*what do you think about programming a computer system? (e.g., sensors and actuators)*" (Q8). In PostQ, Q8 changed to "*what do you think about programming a real smart object? (e.g., smart houseplant)*". All other questions remained the same as they were in IntermediateQ.

Programming questions (PrePQ, IntermediatePQ, and PostPQ). In order to evaluate the students' prior programming experience and measure the acquisition of programming skills, they were asked to perform a pre programming question (PrePQ), an intermediate programming question at the end of the second day of the workshop (IntermediatePQ), and a post programming question at the end of the workshop (PostPQ). In each pre, intermediate and post programming question, block-shaped elements were designed independent of the block-based programming environment in order to test how well the students acquire the basic programming concepts which were taught during the workshop (e.g., see Figure 7.3 for the block-shaped elements in IntermediatePQ).

PrePQ asked to program the micro-controller to get the data from a connected sensor, write the sensor's value into a variable and show it in an RGB LCD for 2 seconds. We added control-flow statements in IntermediatePQ; therefore, we asked

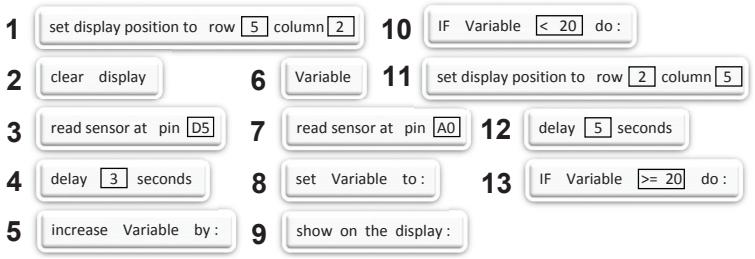


Figure 7.3: Block-shaped elements in the intermediate programming question (IntermediatePQ); translated from German to English.

this time that `if` the value of the sensor is less than 20, `then` the RGB LCD should show the value for 5 seconds in the second row and fifth column. PostPQ contains all previous concepts plus loops. This time, we asked that `if` the value of the sensor is more than 30, `then` the RGB LCD should show the value for 3 seconds in the first row and the fourth column. In addition, the LCD should then blink in green for 3 times with 1 second delay in between.

In each programming question, students were asked to answer the question via selecting a set of blocks and identifying the order of them in a correct logical way. It was noted that some blocks might not be needed and some may appear more than one time in their answers. All programming questions are slightly different from each other, and they are getting more advanced from the beginning towards the end of the workshop. This counterbalance design of questions is to ensure that students read the questions carefully and identify the order of blocks based on the question. Furthermore, these questions represent realistic programming problems for a microcontroller (e.g., Arduino), a sensor (e.g., light, temperature) and an actuator such as RGB LCD. The solution given by the student were collected for each programming question and evaluated by the 10-point grading rubric (see Table 4.1). Each solution was scored independently by two researchers to ensure consistent grading.

7.3.2 Participants

A total of 12 6^{th} grade female students (between 11 and 12 years old) of a German secondary school participated in the study. The school teacher was contacted regarding our programming workshop. Then, students and their parents were informed by their school to register for it. Therefore, the students who participated in this study were self-selected, and interested in learning programming and having programming activities. None of the students had received teaching in program-

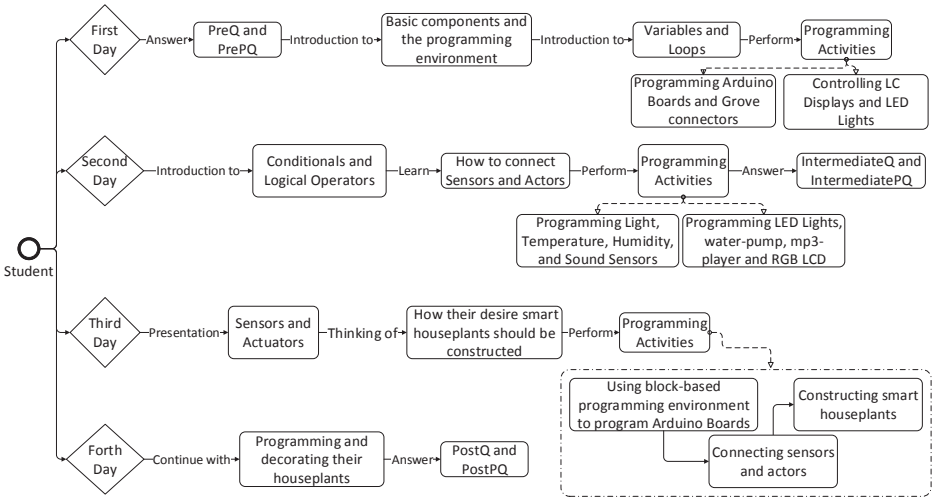


Figure 7.4: Procedure of the programming workshop.

ming as part of their regular school curriculum. However, we employed a question to record their previous programming experiences, and six students indicated that they worked with block-based programming environments and tangibles in the past.

7.3.3 Procedure

The duration of each daily session was five hours, with one hour break. One female and one male instructor led the whole workshop. Both instructors had a computer science background with experience in working with young students. In this study, students worked in pairs on each of the activities. Students with prior programming experience were paired together, and those without experience were paired with each other. All students answered pre, intermediate, and post questionnaires and programming questions individually. The questionnaire was filled first each time, followed by the programming question. Each day, an oral explanation was given, using prepared slides. Additionally, we used supplementary documents—including an explanation of all materials and necessary blocks for programming and activities—in order to help the students, as well as minimize and control the instructor effects. The description of the topics and activities covered over the different days are as follows(see Figure 7.4):

First Day. First, the students filled in PreQ and PrePQ. They were then informed that they are going to have a set of programming activities in each group, and that these activities help them to program and design a smart *houseplant*. Then, pairs of two students (2 experienced or 2 inexperienced) were assigned to one computer. This session was followed by an introduction to the block-based programming environment. The programming concepts introduced in this session were variables, loops, and RGB coloring model, using Arduino and RGB LCD. Students also learned how to show string and numerical values on the RGB LCD in different courser positions, and how to change the LCD color. We asked the students to explore the corresponding blocks in the programming environment.

Second Day. First, we continued with how variables and loops function. Then, students were introduced to sensors (light, temperature, humidity, sound, etc.) and actuators (LED light, water-pump, mp3-player, RGB LCD, etc.). They also learned how to get data from a sensor and put it into a variable. The programming concepts included in this session were the definition of control-flow statements, such as conditions and logical operators. Students were required to execute and understand the blocks. In this respect, they started to program actuators to react to sensors data. At the end of the second day, students filled in IntermediateQ and IntermediatePQ.

Third Day. At the beginning of this session, each group of students was asked to present and share with others how the Arduino, sensors, actuators, variables, loops, and control-flow statements work and are executed. Then, each group chose a *houseplant*. We asked them to give a character to the houseplant and think of how the sensors and actuators in their desire houseplant should communicate and react to each other. This session followed by the implementation of the programming concepts in Arduino and start programming it based on the character of the houseplant.

Fourth Day. In this session, students continued with programming and designing the houseplant. At the end of this session, PostQ and PostPQ were given to the students to find out the changes in their performance and attitudes from the beginning of the workshop towards the end of it. The workshop ended with the presentation of the character and the functionality of each houseplant.

7.4 Experimental Evaluation

All participants filled out PreQ and IntermediateQ, as well as PrePQ and IntermediatePQ—responses to all open-ended questions can be found in the appendix C.7;

Table 7.1: Students' Programming Performance

	Experienced	**Inexperienced**	
Questions	*M (SD)*	*M (SD)*	*ANOVA Results*
PrePQ	5.00 (2.45)	2.00 (1.26)	F(1,10) = 7.11, **p = 0.024
IntermediatePQ	5.33 (2.73)	3.33 (1.03)	F(1,10) = 2.81, p = 0.12
PostPQ	5.67 (2.94)	2.40 (0.89)	F(1,9) = 5.63, **p = 0.042

M: Mean **SD**: Standard Deviation **F**: F-distribution **p**: p-value ****p** < **0.05**: Significant Difference

all translated from German to English. One participant did not show up on the last day, and thus, PostQ and PostPQ were filled by eleven students. Please note, this student showed a negative attitude in PreQ and IntermediateQ. This case is not addressed in the description of results, but it is included in the diagrams and we refer to it in Section 7.5. The written responses to the questionnaires were coded independently by two researchers and then discussed in order to find an agreement on final categories. In each diagram (Figure 7.5 to Figure 7.7), `P1` to `P6` are students with prior experience, and `P7` to `P12` are students without prior experience in programming.

7.4.1 Acquisition of Programming Skills

The students' performance was assessed three times, at the beginning, in the middle and at the end of the workshop (described in Section 7.3). The experienced students performed significantly better than the inexperienced students in PrePQ and PostPQ, and their performance improved (descriptively) from PrePQ towards the PostPQ (see Table 7.1). Furthermore, the performance of inexperienced students improved (descriptively) in IntermediatePQ, where no significant difference was obtained compared with the performance of experienced students. However, their performance dropped in PostPQ (see Table 7.1). No significant difference occurred within each group of experienced and inexperienced students from PrePQ towards PostPQ.

7.4.2 Attitudes and Perceptions of Programming

7.4.2.1 Confidence

Concerning the students' confidence, they were asked to rate their programming skills (`Q1`). After coding their responses, we had the following categories: "*not so good (bad)*", "*not so good but not so bad*", "*improving*", "*good*", "*great (very good)*", and "*others*" (see Figure 7.5a). We saw a positive trend in IntermediateQ for two-third (8) of the students while it remained the same for the other participants. It

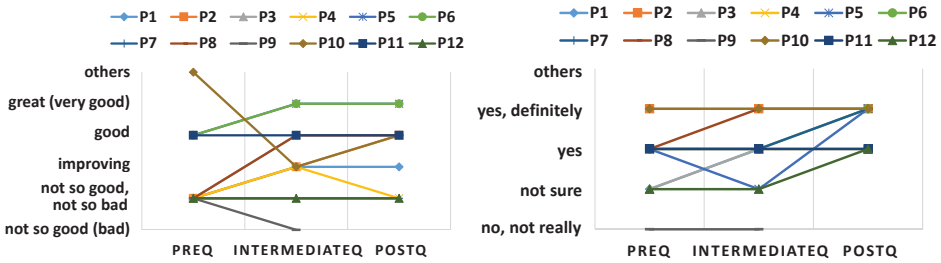


Figure 7.5: (a) Students rate their programming skills (Q1); (b) Students' thoughts on their success in the workshop (Q2).

decreased for one student who did not show up on the last day (P9). No specific difference was observed between inexperienced and experienced students. In PostQ, we saw an increase in confidence for two participants (P2 and P10). One (P4) rated her skills less positive than in IntermediateQ, but equal to PreQ. The remaining participants rated their skills the same as in IntermediateQ.

The students were also asked whether they think that they would be successful in the workshop (Q2). The answers were categorized as "*no, not really*", "*not sure*", "*yes*", "*yes, definitely*", and "*others*" (see Figure 7.5b). At the beginning, three students were unsure (P1, P3, and P12), and one said "*no*" (P9). The confidence increased or remained the same (positively) for all students, except one (P9), towards the end of the workshop. At the end, no unsureness was seen, and all students rated their success with "*yes*" or "*yes, definitely*".

7.4.2.2 Enjoyment and Interest

With respect to the enjoyment of programming, students were asked to indicate how they find programming (Q3). The answers were categorized in "*complicated*", "*fascinating and interesting*", "*hard fun*" (inspired by [Pap02]), "*easy and logical*", "*fun (great)*", and "*others*" (see Figure 7.6a). With respect to the term "*hard fun*", Papert [Pap02] provided a special kind of fun when he saw kids liked hard challenging. Thus, he tried to find a term for these circumstances which could be called "*pleasure*" or "*fun*", but he finally called it "*hard fun*". All participants liked programming in IntermediateQ and PostQ more than in PreQ. However, four students (P2, P7, P10, and P11) changed their mind from IntermediateQ to PostQ; for instance, they realized that the programming is not only "*fun*", but fun and, at the same time, challenging ("*hard fun*"), or just "*interesting*".

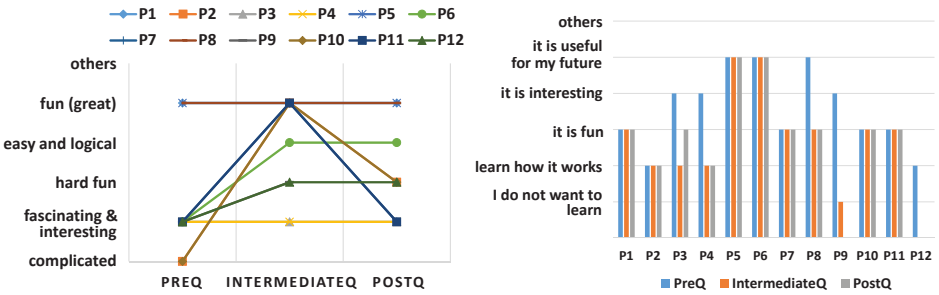


Figure 7.6: (a) How students found programming (Q3); (b) Why students like to learn programming (Q5).

The students were also required to indicate why they would like to learn programming (Q5). This question aimed to find out about their general interest in learning programming. The answers were categorized as "*I do not want to learn*", "*learn how it works*", "*It is fun*", "*It is interesting*", "*It is useful for my future*", and "*others*" (see Figure 7.6b). Four students (P1, P7, P10, and P11) mentioned that programming is "*fun*" from the beginning of the workshop towards the end of it. One student (P3) finally found that the programming is "*fun*". Furthermore, three students found it "*useful for their futures*" (P5, P6, and P8). However, P8 changed her idea and found programming is "*fun*" during the workshop. For seven participants, the reason to learn programming remained the same throughout the workshop. In particular, the motivation of inexperienced students to learn programming was (expecting) "*fun*", while experienced students mentioned the learning of how technical things work.

7.4.2.3 Block-based Programming and Smart Objects

In IntermediateQ and PostQ, participants were required to respond to how they liked programming with blocks (Q7). The answers were categorized as "*exhausting*", "*scope of action (e.g., opportunities to be creative)*", "*hard fun*", "*easy*", "*fun (great)*", "*others*". Here, the answers changed from IntermediateQ to PostQ for half of the participants (see Figure 7.7). Experienced participants often mentioned that "*it is easy*", or appreciated the "*scope of action*", while inexperienced students found it "*fun*" or "*hard fun*". Please note that all experienced students mentioned in Q6 (in PreQ) that they worked with other block-based programming environments before.

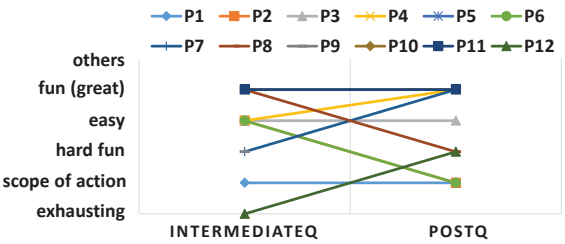
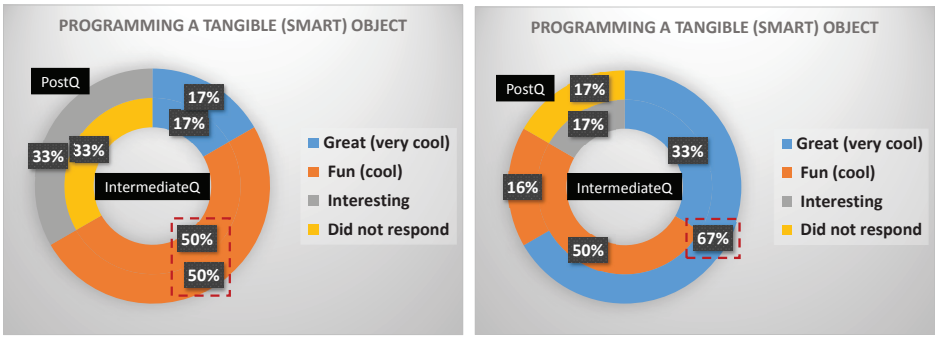


Figure 7.7: How students like to program with blocks (Q7).

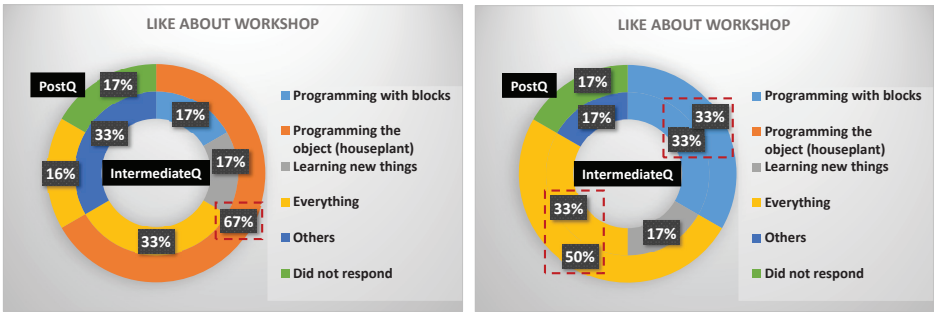


(a) Experienced students (b) Inexperienced students

Figure 7.8: How students like to program a tangible object (Q8).

We also distinctly asked how they liked programming a tangible (smart) object ((Q8) in IntermediateQ and PostQ). Categories were "*great (very cool)*", "*fun (cool)*", "*interesting*", "*no response*" (see Figure 7.8). We saw a difference between experienced and inexperienced students. Half of the experienced students indicated that it was "*fun (cool)*" in both IntermediateQ (P2, P4, and P6) and PostQ (P2, P3, and P6). P4 found it "*interesting*" in PostQ, and P3 did not answer this question in IntermediateQ. Inexperienced students indicated that it was "*great (very cool)*" (P8 and P11) or "*fun (cool)*" (P7, P9 and P10) in IntermediateQ; P12 found it "*interesting*" to program a tangible object in IntermediateQ. Their enthusiasm grew towards the PostQ with a shift of two students (P10 and P12) to "*great (very cool)*". In general, half of the students did not change their minds between IntermediateQ and PostQ.

Students were also asked (Q4) to answer what they look forward to in the workshop (PreQ) and what they liked about the workshop (in IntermediateQ and PostQ).



(a) Experienced students (b) Inexperienced students

Figure 7.9: What students like about the workshop (Q4).

Categories are "*teamwork*", "*learning programming/technical things*", "*program a real object*", "*others*", and "*no response*" in PreQ. In addition, categories in both IntermediateQ and PostQ are "*programming with blocks*", "*programming the object (houseplant)*", "*learning new things*", "*everything*", "*others*", and "*did not respond*" (see Figure 7.9). In PreQ, students were mostly (5) looking forward to "*learning new and technical things*". For inexperienced students, this was about two-third (P7, P8, P10, and P12). However, experienced students were more differentiate, and two of them specified "*programming a tangible object*" (P4 and P6). In IntermediateQ, the distribution of categories changed, and we can see a clear difference between experienced and inexperienced students. With this regard, experienced participants mostly mentioned: "*programming with blocks*" (P2 and P3), and "*everything*" (P1 and P4) in IntermediateQ. In PostQ, two-third (4) of them mentioned: "*programming the object (houseplant)*" (P3, P4, P5, and P6). Inexperienced participants mostly answered, "*everything*" (P11 and P12) and "*programming with blocks*" (P7 and P8) in IntermediateQ. In PostQ, three of them (P10, P11, and P12) mentioned "*everything*", and two of them (P7 and P8) mentioned "*programming with blocks*".

7.5 Integration and Discussion

In this pilot study, the students' performance in the programming questions did not correlate with their confidence concerning perceived programming concepts. Therefore, although girls had some difficulties in understanding the subject, they still felt positive and confident about their programming skills and success. This is in line with the results presented in [Bei05, MCK17, NHCW04] that girls' confidence

level increases by visual programming environments and experience with interactive tangible objects.

With respect to the confidence, enjoyment and interest, a clear difference between experienced and inexperienced students was not observed. However, some differences were seen in the students' responses to the questions regarding the workshop activities and items. The majority of experienced participants mentioned that they liked construction of a smart object in this workshop, while the inexperienced students remained rather vague and mentioned "*everything*" or only "*programming*". The results showed that especially the experienced students appreciated working with the houseplant (as a tangible everyday object) and, due to their prior experience, they were able to articulate clearly what they like and why it was fun for them. Furthermore, the opportunity of applying their programming skills to a tangible object and making it smart was more meaningful and interesting for them. In contrast, the inexperienced students did not mention the tangible everyday object (in this case, houseplant) but they were mostly impressed by programming with a visual block-based programming environment. In addition, the findings showed that programming by itself was interesting for the inexperienced students and applying it to the tangible object did not stand out. We can assume that they did not yet have the terminology to distinguish between programming with and without tangibles. From these results, we can draw implications on designing courses for experienced and inexperienced (female) students. For experienced students, it is indicated that having a meaningful application area for programming such as a tangible everyday object, as well as making it smart is an important area that needs to be taken into account. This supports the results from the programming questions that experienced students performed significantly better than inexperienced students in the PostPQ, while it was not significantly better in IntermediatePQ. This result is in line with findings from [KP05], which showed that experienced students had more benefits from tangible objects and platforms than inexperienced students.

Our initial research question was how programming skills and attitudes change over time in the context of programming and constructing smart everyday objects. This also includes finding out whether using a tangible everyday object and providing the possibility to make it smart changes young female students' attitudes towards programming. Concerning the confidence in programming, the biggest changes were observed between PreQ and IntermediateQ. After the introduction to the block-based programming environment, the confidence in programming increased. Except for one (the dropout student), all students felt that they performed equal or better than what they had expected. The findings showed that the students' confidence in programming was not affected by the implementation of the programming concepts in a tangible everyday object and the experience gained by that time. This is in line with their opinion about programming. After initial

excitement in IntermediateQ, their confidence dropped. This indicates that they realized programming is not just fun but also challenging after programming and constructing the smart object. However, the feeling of being successful increased after working with the tangible object and making it smart. One reason could be that "*smart houseplant*" was the general topic and the objective of the workshop, which our participants felt that it was achieved.

Enjoyment of programming increased in IntermediateQ, and it had a decrease for a few participants after using the tangible object. We assume, one reason is that the complexity of the tasks increased. Nevertheless, working with tangibles did not indicate a distinct effect on enjoyment but helped to keep it up.

As mentioned in Section 7.4, one student dropped out. Although she had excused herself to the workshop instructor in advance that she had another commitment on the last day, she showed negative confidence and attitude in her responses to some questions in PreQ and IntermediateQ. While the results which are obtained from the other students are promising, we would like to highlight that we can probably learn a lot more from dropout participants. Thus, we argue for looking more in-depth into these cases in future iterations of similar studies; for instance, using ethnographic methods.

7.5.1 Limitations

While we tried to provide insights into the relationship between constructing smart objects and improving young female students' attitudes and programming performance, this study has limitations which are addressed in the following.

The first limitation of this study relates to the number of programming tasks and the period of the workshop. For instance, the findings of this study are limited to the diversity of the programming tasks that the students performed and how they speak to more diverse programming activities and computational skills. A second similar limitation is that the intermediate programming question might have had an influence on students' performance in learning basic programming concepts and their motivation. Further work is needed to address these limitations and to generalize the findings beyond the specifics of this study, such as the period of programming workshop and type of programming tasks.

Another limitation of this study is related to the number of participants. We would like to emphasize that the relatively small sample size (12 female students) lowers the power of findings to be generalized on a large scale. Thus, we look at this as a major concern that needs to be addressed in future directions when we expand the scope of this work with a larger sample size.

A final limitation of this study relates to the control group. This is another pathway of future work to find out the impacts of programming courses on students'

attitudes and programming skills over time without using tangible objects as well as when young male students are targeted for such programming courses. This limitation is covered in the next chapter of this thesis (Chapter 8).

7.6 Conclusion

As the presence of women in computer science discipline is lower than men in most western countries, raising girls' interest in the programming side of computer science is an active area of research. In this chapter, we presented a pilot study investigating the impacts of programming and constructing a smart everyday object (smart houseplant) on girls' programming skills and attitudes. Programming performance, enjoyment, interest, and confidence were not only assessed with programming questions and open-ended questionnaires before and after a non-formal programming workshop but also intermediately before starting to implement programming in the tangible everyday object (a houseplant). Our findings indicate the girls' confidence did not match their actual performance. Their confidence increased after introducing the block-based programming environment, and it remained high after construction of the smart object. However, being able to program the houseplant was perceived differently by experienced and inexperienced participants. Data shows that block-based programming was interesting "*enough*" for inexperienced students, while experienced students appreciated more to apply their skills to make the houseplant smart. Furthermore, our study supports the claim that construction of smart objects has a direct impact on young female students' performance and attitude towards programming. By studying the influence of constructing smart everyday objects as an application of computing in reality, we enhance our understanding of designing appropriate programming courses concerning the students' prior experience. While many questions still remain on how to best introduce programming to girls, the findings of this study are essential to inform other researchers and educators about the relation between tangible objects together with block-based programming, and girls' programming performance and attitudes towards programming.

Chapter 8

From Block-based Programming to Construction of Smart Objects

Nowadays, with the advent of graphical programming environments, block-based programming has been utilized to introduce young learners to programming and computer science more broadly. When introducing programming to them via block-based programming, showing the application of programming in reality is a key aspect. Previous work have been done on how to teach basic programming concepts, using block-based programming and tangible objects like robots, smart textiles, wearable products, LED lights, and smart artifacts. Nevertheless, there have been very few studies with a special focus on how young students' attitudes and programming skills changed over time, when they introduce to real life-size smart environments, as well as learn to program and construct a smart tangible object, connecting to the smart environments.

In this chapter, we present findings from a user study, investigating how the use of block-based programming together with a real life-size smart home can leverage young learners' (both female and male) interest in programming over time, and support the acquisition of programming skills. To do this, we use *BEESM* (introduced in Chapter 3) along with constructionist learning theory. In Constructionism, educational designers and researchers tried to turn the table providing learners with greater opportunities to construct a product, and thus, enable them to learn by doing. 28 8^{th} grade students from a German secondary school participated in the study, who programmed and constructed a *smart-lightning object* integrated into the

smart home. Their performance and trajectories of attitude towards programming were assessed, using repeated questionnaires throughout the study. The findings imply that using block-based programming in the context of smart homes can foster young learners' programming skills, and develop a positive attitude towards programming.

8.1 Introduction and Motivation

The importance of computer programming is increasingly growing together with the dispersion of computing applications in our high-tech world. Increasingly, young learners start with programming activities via visual block-based programming environments [Wei19]. These environments are highly used to design introductory programming courses [WW17b, WHHF18], and workshops [MCK17, MGB15]. Furthermore, the application of programming tends to be shown to the learners in reality. In this respect, in addition to allowing them to learn the general purpose of programming and author programs via block-based programming environments, previous work provides possibilities to experience and implement new ideas into tangible objects and real environments [MCK17, PHEC17, MGB15].

The design of introductory programming concepts has resulted in a growing use of tangible objects and block-based programming to enable inexperienced and young learners to program. However, relatively little attention has been given to show potential for using educational block-based programming environments to make state-of-the-art smart technologies accessible for them. More specifically, little work has investigated how introducing young learners to real life-size smart homes, and teaching basic programming concepts via block-based programming can improve their programming performance and attitude towards programming over time. Unfortunately, these smart homes rely on modern and powerful technologies, which make them not fully accessible for young learners. In previous research, block-based programming environments are already employed to introduce the general purpose of programming to young learners in the context of mobile robots [MCK17, PHEC17, MGB15], and smart homes [KD18]. Additionally, it is addressed that introducing young learners to modern technologies fosters their programming skills [MGB15, MCK17] and interest in computer science [MGB15]. While there is much to show for teaching programming to young learners, less is known about how their performance and attitude towards programming are influenced over time in the context of smart homes. Thus, research is needed to show that by using block-based programming along with real life-size smart homes, we are able to not only introduce programming to young learners but also to show computing applications in the aspect of everyday life. In this chapter, we investigate

the learners' trajectories of acquisition of programming skills and their attitudes towards programming in the context of real life-size smart homes by answering the following research questions:

- In what ways can young learners benefit from learning environments that use smart homes as an application area for block-based programming and construction of smart objects?
- Does the construction of a smart interactive object in the context of smart homes have a positive impact on young learners' programming performance and attitude towards programming?
- Are there differences between experienced and inexperienced young learners with respect to their programming performance and attitude towards programming?

To answer this question, we conducted a 2-day non-formal programming workshop with 28 8^{th} grade students (12 to 14 years old). We claim that it is possible to introduce young learners to personally meaningful state-of-the-art smart technologies in the context of smart homes. This enables them to author programs and to see their scope of action to creatively design and control a tangible object embedded in a smart home system. The main goal of using the tangible object is to let the learners apply their new gained programming skills into it, and making it smart. This provides opportunities for them to find out the connection of programming to modern reality, and thus, motivate them to begin with programming activities and develop a greater interest in programming. In order to reduce the complexity of programming and facilitate for the learners to learn basic programming concepts and author programs, a block-based programming environment (*BEESM*) has been employed as a form of visual programming environment. The block-based programming environment integrated functionality for connecting a micro-controller (in this case, WeMos D1 mini Wi-Fi board) to the smart home server in order to read the data generated by different items in the smart home. The smart home is *BAALL*, an approximately 60 m^2 smart living lab apartment which is equipped with various actuators (e.g., doors, toggleable, dimmable and RGB lights), sensors (e.g., lighting, temperature, and thermal cameras), voice recognition system, and smart mirror and fridge. The additional features of *BAALL* can be found in more detail in Section 5.3.2 of this thesis.

In a user study, young learners were enabled to author programs that connect the micro-controller to the smart home server, and read the generated data by different items. The data was then used in order to construct a *smart-lightning object* which consists of *lights* and a *organic light-emitting diode (OLED) display*, both connected

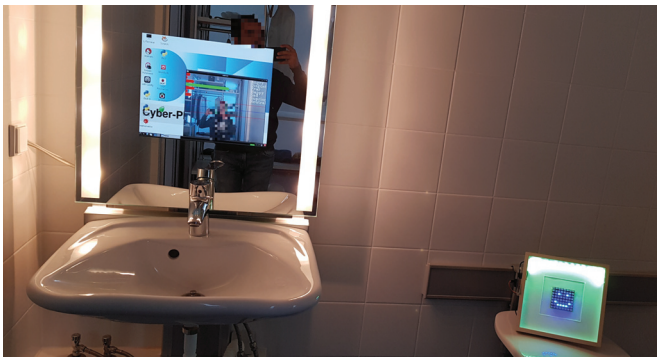


Figure 8.1: A sample of a "*smart-lighting object*", using the smart mirror.

to the micro-controller. For instance, learners can program the micro-controller to show different colors in lights and suitable information on the display as soon as a smart mirror detects someones' face expression who is standing in front of it (see Figure 8.1). It is a special feature of this study that the learners were able to author pieces of code that can be applied to a tangible object like a *lighting object* and make it smart, using the block-based programming and smart homes generated data. The path of learners' attitude towards programming based on repeated quantitative and open-ended qualitative questionnaires was examined (i) at the beginning, (ii) at the middle, and (iii) at the end of the programming workshop. Furthermore, we assessed their programming performance at the beginning and at the end of the workshop, using two programming questions.

The chapter begins with a review of previous work and how they employ smart objects and environments to teach basic programming concepts together with block-based programming environments (Section 8.2). Then, we describe the study design and strategy for collecting and analyzing data in Section 8.3. The evaluation and experimental results are presented in Section 8.4, and they are discussed in Section 8.5. The chapter closes with limitations and conclusions 8.6.

8.2 Related Work

A key issue in learning programming, especially among young learners, is that they do not have a clear idea how programming and computer science are relevant to their daily life [GCNB18, CGN19]. Previous work have been done on how to teach basic programming concepts, using block-based programming and tangible objects like robots [MM19,PHEC17], wearable products [BEE06,GCNB18,KDS09,KDS15],

smart homes [KD18], LED lights, etc. In both CSE and CHI research communities, several scientific studies tried to explore how young learners' performance and attitudes is influenced when they learn basic programming skills based on block-based programming, and then apply them into tangible objects to build an smart object. Nevertheless, there have been very few studies with a special focus on how young students' attitudes and programming skills changed over time, when they introduce to smart environments and learn to program and construct a smart tangible object, using these environments.

Robots are one of the most common tangible and smart devices which are used for learning and educational purposes. The literature reports that mobile robots are effectively used as a tool to generate positive interest and improve learning among young learners [PHEC17, MGB15]. Paramasivam et al. [PHEC17] explored the use of mobile robots as a tool for suited reflection in elementary school programming courses. Furthermore, Martínez et al. [MGB15] enabled preschool and elementary school learners to program and control the behavior of Arduino boards in the context of N6 robots. The results show that the learners were highly engaged by robot programming, and researchers were successful to establish the confidence that robot programming is interesting. However, no information was provided whether this approach enable them to have a higher intention of learning programming or not.

Another approach is to use smart homes and living labs in order motivate young learners in learning programming and foster their interest in computer science. Katterfeldt and Dittert [KD18] reported on three co-design workshops where young female learners created ideas related to smart homes, and implemented them on a doll house. Still, a key challenge with the use of real life-size smart homes is that young learners can only perform limited actions on the environment and they are not able to make a real application in there. For instance, in Chapter 5, we used real life-size smart homes as a medium to teach basic programming skills to young learners. However, the variety of activities were limited and learners could not really show their creativity when they program different objects within the smart home.

8.2.1 Summary

Irrespective of the concrete approach, the question arises how to support young learners to learn basic programming concepts, and improve their attitude towards programming. The use of visual block-based programming environments can reduce the complexity of programming for the learners. However, block-based programming alone is not sufficient for them to develop a connection between programming and its impacts in their daily life. Having access to smart environments may be useful to show how these environments can react to daily needs. Although these environments are able to show how programming and computer science are relevant

to our daily life, they may not be fully accessible for young learners to build an application for them. To this end, we designed and implemented a 2-day non-formal programming workshop to provide opportunities for young learners to participate and experience modern technologies in the context of smart homes. We analyzed the learners' trajectories of attitude towards programming and their performance based on repeated questionnaires and programming questions from the beginning towards the end of the workshop. In other words, by using block-based programming and letting young learners to program and construct a smart object in the context of smart homes, we not only aim to teach programming but also to leverage their interest in learning programming and understanding the influences of it in their daily life.

8.3 Methodology

This study aims at experimentally investigating the impacts of smart homes and block-based programming environments on young learners' programming skills and attitudes towards programming. We conducted a user study in a 2-day non-formal programming workshop. A visual block-based programming environment (*BEESM*), and a smart home (*BAALL*) were used in order to enable the learners to have a hands-on experience and construct a *smart-lighting object*. We used three questionnaires throughout the workshop, including both 5-point Likert Scale and open-ended questions. The questionnaires were given to learners: (i) at the beginning of the workshop (PreQ), (ii) at the middle of the workshop, before the learners introduced and learned about the smart home (IntermediateQ), and (iii) at the end of the workshop, when the learners implemented their newly learned programming skills to construct the smart-lighting object based on the smart homes generated data (PostQ). Three dimensions of learners' attitude were considered: confidence, enjoyment, and interest in future programming learning opportunities [WW17b]. Furthermore, the learners' performance was assessed, focusing on three basic computational concepts: variables, loops, and control-flow statements (conditions and logical operators) [Lew10]. We applied programming questions at the beginning (PrePQ), and at the end of the workshop (PostPQ).

The main emphasis of this study is to reveal possibilities how block-based programming can be used to make modern and powerful technologies more accessible to young learners. In doing so, an experimental basis for the used of block-based programming in the context of real life-size smart homes was provided. Young learners were enabled to construct a smart tangible object in the context of our smart home, using the block-based programming environment. In this regard, the learners' experience of using the block-based programming application (ease-of-use,

Figure 8.2: Screenshot of the programming environment interface.

ease-of-learning, usefulness and satisfaction) [WAS+18], and constructing a smart tangible object was also measured, using collected data from IntermediateQ and PostQ.

This section begins with the materials used in this study, followed by study design and strategy for collecting and analyzing data. Then, the information about the participants is presented. This section concludes with the procedure of the study.

8.3.1 Study Design and Data Collection Strategy

In this study, we used the micro-controller and smart home parts of *BEESM* in order to program a smart tangible object in the programming workshop (see Figure 8.2). The block-based programming environment enables students to focus on programming structures and principles, the main computational concept which was taught and exemplified through it. We changed the *BEESM* user interface to enable our target students to have a full vision of the blocks (Block Panel), code syntax (Code Panel), output of the code (Output Panel), and a 2D view of the smart home (2D Graphical Panel). A useful feature of the programing environment added to the *BEESM* interface (in this step) is the ability to toggle between the Code and Output Panel by clicking on the "*Output*" button. This enables learners to have a larger section for blocks which is suitable to improve the visibility of finding and reading blocks in the program (addressed by Holwerda and Hermans [HH18] and in

Chapter 6 of this thesis). Furthermore, Arduino code is used to connect to our smart home, and to program the micro-controller (WeMos D1 mini board). The smart home's main educational use in the programming workshop is to allow the learners to connect to its server and read the generated data via the WeMos D1 mini board with Wi-fi functionalities. Then, they can program *lights* and a *OLED display* in a way that they react to the data, and construct a *smart-lighting object*. We chose to work with WeMos D1 mini boards because it has an on-chip Wi-fi Transceiver. It is introduced as a very compact solution for prototyping small smart objects with Wi-fi functionalities, such as connecting to Wi-fi and Web-based interactions (e.g., HTTP GET and POST requests). We also used OLED displays because they are compatible with WeMos D1 mini boards, and they enable learners to show both numerical and string values on them [1].

In the following, we describe the pre, intermediate, and post questionnaires, as well as the pre and post programming questions. Questionnaire were employed to collect data concerning the students' attitudes towards programming, their prior programming experience, their gender, and their age group. The acquisition of basic programming skills among the students was assessed, using the programming questions. All PreQ, IntermediateQ, and PostQ, as well as PrePQ and PostPQ can be found in the appendix D; all translated from German to English.

Pre questionnaire (PreQ). The PreQ at the beginning of the workshop consists of eight (`Q1-Q8`) 5-point Likert scale questions (with 1 "*no, not at all*", and 5 "*yes, very much*"), two (`Q9 and Q10`) "*yes*" or "*no*" questions and two open-ended questions (`Q11 and Q12`). PreQ was used to record the learners' attitude towards programming (confidence, enjoyment and interest), using the 5-point Likert Scale questions. These attitudinal questions were designed based on the questions from [WW17b] with specific questions being added for this study. The learners prior experience with block-based programming environments and micro-controllers were asked via the two "*yes*" or "*no*" questions. Learners were also required to indicate their intentions towards the programming workshop, and their perception of computer programming, using two open-ended questions.

Intermediate questionnaire (IntermediateQ). The IntermediateQ after the first day of programming activities included: (1) one (`Q1`) 5-point Likert scale question to measure the learners' perception of using a tangible object and making it smart; (2) eight (`Q2-Q9`) 5-point Likert scale questions to measure the learners' attitude towards programming (answers for all nine questions categorized as 1 "*no, not at*

[1] We make our workshop materials available at https://github.com/projekt-smile/Smartes-Stimmungslicht

all", and 5 "*yes, very much*"); (3) eight (`Q10-Q17`) 5-point Likert scale questions to measure the learners' experience using the programming environment in terms of its ease-of-use, ease-of-learning, usefulness and satisfaction (with 1 "*strongly disagree*", and 5 "*strongly agree*"); (4) four (`Q18-Q21`) open-ended questions to record learners feedback regarding the programming environment and the workshop. The eight attitudinal questions are largely similar to the questions in the PreQ, just with different words for two questions; "*do you think you will be successful in this workshop?*" changed to "*do you think you were successful in this workshop?*", and "*would you like to learn how to program?*" changed to "*would you like to learn more about programming?*".

Post questionnaire (PostQ). The PostQ at the end of the workshop consists of all questions which are asked in IntermediateQ, and two questions regarding the learners' gender and age. In PostQ, "*do you think it's useful if you program a real object? (e.g., the LEDs light up)*" changed to "*do you think it would be useful if you programmed a real smart object? (e.g., perform actions based on sensor information)*". Furthermore, two open-ended questions "*what did you particularly like/dislike about the 1st workshop day?*" change to "*what did you particularly like/dislike about the workshop?* in PostQ.

Programming questions (PrePQ and PostPQ). We also assessed the learners' performance in basic programming concepts during the study. The PrePQ and PostPQ included programming examination, with two gap-filling and seven open-answer questions. The answer of each question has one point, except for question 6 which has two points as the learners needs to mention both LED colors. The answer given by the learners were collected for each question and evaluated independently by two researchers to ensure consistent grading. The first and sixth questions concern understanding of "*variable*", questions 2, 3 and 9 concern "*control-flow statements*", and questions 4, 5, 7 and 8 concern "*loop*". In this study, PrePQ and PostPQ are slightly different from each other. This counterbalance design of questions is to ensure that learners understand the blocks and read the questions carefully in order to give the correct answers to them.

8.3.2 Participants

In total of 28 8thgrade students from a German secondary school were participated in our workshop (22 boys and 6 girls; ages 12—14, $M = 12.96 \mid SD = 0.33$). Teachers suggested which students could participate in this study and came with them to us. The school and teachers were informed about study protocols. All parents were informed prior to the study by the school teachers. The students performed

and completed all activities after parent consent. All equipment (computers and components), as well as the smart home were provided by the German Research Center for Artificial intelligence (DFKI). All students had prior experience with block-based programming environments as part of their curriculum. In PreQ, we employed two questions to record their previous programming experiences. All girls indicated that in addition to block-based programming, they have worked with micro-controllers (in this case, Arduino boards) and tangibles (in this case, LED lights) in the past as part of an non-formal workshop. In this regard, we will call them *experienced students* throughout this chapter, while 22 boys will be called *inexperienced students*.

8.3.3 Procedure

The duration of each session in each day was six hours, with 90 minutes break. In this study, students worked in a group of three and two on each of the programming activities (10 groups in total). As prior experience and gender factors were considered, girls (they worked with Arduino boards and LED lights before) were grouped together, and boys (they did not work with micro-controllers before) were grouped with each other. However, all students answered PreQ, IntermediateQ, and postQ, as well as PrePQ and PostPQ individually. All students filled the questionnaire first each time, followed by the programming question. Each day, an oral explanation was given, using prepared slides. Additionally, we used supplementary documents, including an explanation of all components and necessary blocks for programming and activities. These materials were used in order to help the students, as well as minimize and control the instructor effects. The description of the topics and activities covered over the first and second day are as follows (see Figure 8.3):

First Day. Each student began by completing the PreQ and PrePQ. The students got an explanation that they are going to use the block-based programming in order to perform a set of programming activities in each group. They were also informed that these activities help them to program and construct a *smart-lighting object*, using the smart homes generated data. The students were then introduced to the block-based programming environment. Each group of students was provided with the basic components (WeMos boards, LEDs, OLED display, cables and mini breadboard) and brief instructions for connection and construction. Students experienced how to control an OLED display, as well as how to show different string and numerical values on the display in different cursor positions. This session was followed by introducing students to LED lights, necessary blocks to control them, and the RGB coloring model to change the color of the lights. We asked the students to explore the corresponding blocks in the programming environment. Then,

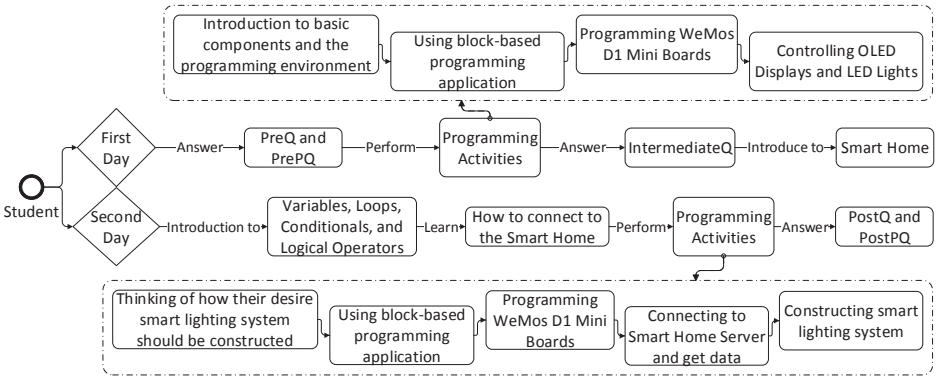


Figure 8.3: Procedure of the programming workshop.

the students were required to fill in the IntermediateQ. This session ended with an introduction to the smart home. All objects (e.g., lights, doors, sensors, etc.) and their functionalities in the smart home were explained to all students to identify different smart items and their functionalities in the smart home.

Second Day. At the beginning of the second day, each group of students was asked to present and share with the others how the WeMos boards, OLED displays, and LED lights worked. Different smart items in the smart home were explained again to the students. In addition to the RGB coloring model, using WeMos boards and OLED displays, the programming concepts introduced in this session were variables, loops, and control-flow statements. The students also learned how to write a program for WeMos boards in order to connect them to the smart home server, read the data, and put it into a variable. In this respect, they started to program the micro-controller to control the OLED display and LED lights in order to react to the smart homes generated data. Students then formed groups of 2–3 students and developed their project ideas by considering how they would design artifacts with LED lights and displays to be embedded in the smart home ecosystem. They were required to think of how the LED lights and displays in their object should communicate and react to the data which is generated by different items in the smart home. They further designed the layout of the artifacts for their *smart-lighting object*, and implemented their project functionality using the programming concepts in WeMos boards. An overview of the smart-lighting objects and number of groups which made each object is shown in Table 8.1. At the end of the second day, students filled in the PostQ and PostPQ. The workshop ended with the presentation of the characters and the functionality of each lighting object.

Table 8.1: Overview of Smart-lighting Objects Constructed by Each Group

Smart-lighting Object	Number of Groups
...communicating with the smart mirror	4
...communicating with the light and temperature sensors	3
...reacting to the status of lights and doors	2
...reacting to the status of TV (TV volume)	1

8.4 Experimental Evaluation

The results section is divided into three parts. First, results from an analysis of the programming part of the study are presented, reporting the students' performance in each PrePQ and PostPQ. Second, results from an analysis of the PreQ, IntermediateQ and PostQ are presented, looking at the students' confidence, enjoyment, and interest in future programming learning opportunities. Finally, we report on results from an analysis of the IntermediateQ and PostQ, focusing on students' experience in terms of ease-of-use, ease-of-learning, usefulness and satisfaction during the use of block-based programming environment. Additionally, in this part, students' perception of being able to construct and see the impacts of their programs on a tangible smart object is reported.

The following analysis was computed based on the factor "*prior experience*". In this study, as the main difference between boys and girls is that the girls previously worked with micro-controllers and tangible objects in addition to block-based programming, the factor "*prior experience*" can also cover the factor "*gender*". Furthermore, no significant difference was observed between the experienced (girls) and inexperienced students (boys) with respect to their age ($F < 1$). In Table 8.2 and Table 8.3, all significant results between the groups and within each group are shown in "*Yellow*" and in "*Orange*", respectively.

Responses to the open-ended questions were coded independently by two researchers and then discussed in order to find an agreement on final categories. We addressed these responses throughout the result section in order to support the quantitative data—responses to all open-ended questions can be found in the appendix D.6; all translated from German to English.

8.4.1 Acquisition of Programming Skills

With respect to the PrePQ, an ANOVA showed that the effect of prior experience was not significant, $F(1,26) = 3.55$, $p = 0.071$. However, concerning the PostPQ, this effect was significant, $F(1,26) = 7.62$, $p = 0.010$. Focusing on the average performance of the students in PrePQ and PostPQ shows that their performance increased

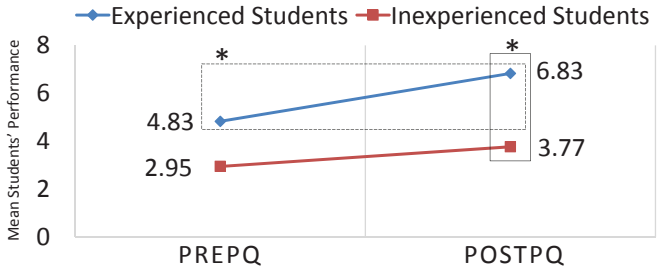


Figure 8.4: Students' performance on the PrePQ and PostPQ.

in both groups; for experienced students (girls), $M = 4.83 \mid SD = 2.23$ in PrePQ, and $M = 6.83 \mid SD = 2.32$ in PostPQ; for inexperienced students (boys), $M = 2.95 \mid SD = 2.15$ in PrePQ, and $M = 3.77 \mid SD = 2.43$ in PostPQ. In this respect, a paired-samples t-test showed that experienced students performed significantly better in PostPQ compared to the PrePQ, $t(6) = 3.87$, $p = 0.012$, $MD = 2.00$. However, no significant result occurred for inexperienced students, $t(22) = 1.71$, $p = 0.10$, $MD = 0.82$. Descriptively, although the experienced students indicated a higher level of performance in PrePQ, their performance was not significantly better than the inexperienced students. However, experienced students showed a greater improvement in PostPQ compared to their performance in the PrePQ, and to the inexperienced students (see Figure 8.4).

8.4.2 Attitudes and Perceptions of Programming

8.4.2.1 Confidence

With respect to students' confidence, we measured their responses to the three questions in PreQ (`Q1-Q2`, and `Q6`), IntermediateQ, and PostQ (`Q2-Q3`, and `Q7`).

Concerning the question whether they think that they are good at programming, inexperienced students (boys) indicated a higher level of confidence in all questionnaires (see Table 8.2). In particular, they showed a significantly higher level of confidence at the beginning of the workshop (PreQ) compared to the experienced students (girls). However, in IntermediateQ and PostQ, no significant results occurred. Furthermore, no significant difference was observed within the group of inexperienced students from the beginning towards the end of the workshop. In contrast, a paired-samples t-test yielded a significant result between the PreQ and the PostQ within the group of experienced students, $t(6) = 2.71$, $p = 0.042$, $MD = 0.83$. It shows that the level of confidence significantly increased among the experienced students at the end of the workshop.

Table 8.2: Students' Attitude Towards Programming

Questionnaires		Confidence: Good at programming	Confidence: Successful in workshop	Confidence: Programming is difficult	Interest: Interested in programming	Interest: Learn how to program	Enjoyment: Programming is fun	Enjoyment: Like programming	Enjoyment: Excited about workshop
PreQ	*Experienced M (SD)*	2.33 (0.82)	3.67 (0.82)	4.00 (0.89)	4.67 (0.52)	4.67 (0.52)	4.50 (0.55)	4.83 (0.41)	4.33 (0.82)
	Inexperienced M (SD)	3.32 (0.84)	3.77 (0.69)	2.91 (1.11)	4.36 (0.85)	4.36 (0.73)	4.64 (0.66)	4.64 (0.58)	3.76 (0.77)
	ANOVA Results	F(1,26) = 6.57, p = 0.017	F <1	F(1,26) = 4.89, p = 0.036	F <1	F <1	F <1	F <1	F(1,25) = 2.52, p = 0.13
IntermediateQ	*Experienced M (SD)*	2.83 (0.98)	3.17 (0.98)	3.67 (1.03)	4.50 (0.84)	4.67 (0.52)	4.67 (0.52)	4.50 (0.84)	3.50 (1.22)
	Inexperienced M (SD)	3.59 (0.85)	4.32 (0.78)	2.71 (1.23)	4.32 (0.89)	4.14 (0.94)	4.64 (0.66)	4.45 (0.91)	4.00 (0.98)
	ANOVA Results	F(1,26) = 3.49, p = 0.073	F(1,26) = 9.23, p = 0.005	F(1,25) = 2.97, p = 0.097	F <1	F(1,26) = 1.73, p = 0.20	F <1	F <1	F(1,26) = 1.11, p = 0.30
PostQ	*Experienced M (SD)*	3.17 (0.41)	3.67 (0.82)	3.67 (1.03)	4.50 (1.22)	4.50 (0.84)	4.83 (0.41)	4.67 (0.82)	4.00 (1.10)
	Inexperienced M (SD)	3.64 (0.79)	4.18 (0.66)	3.05 (1.25)	4.27 (0.88)	4.09 (1.23)	4.50 (0.74)	4.50 (0.74)	4.23 (0.75)
	ANOVA Results	F(1,26) = 1.94, p = 0.18	F(1,26) = 2.58, p = 0.12	F(1,26) = 1.24, p = 0.28	F <1	F <1	F(1,26) = 1.10, p = 0.30	F <1	F <1

M: Mean **SD**: Standard Deviation **F**: F-distribution **p**: p-value

With respect to the question whether they think that they will be/were successful in this workshop, in general, students indicated a medium level of confidence in PreQ. Inexperienced students (boys) showed a significantly higher level of confidence in IntermediateQ, where the confidence of experienced students (girls) dropped. However, in PostQ, no significant result was obtained between the two groups of students (see Table 8.2). No statistical differences were observed within the group of experienced students throughout the workshop. Although, a paired-samples t-test showed a significant result between the PreQ and IntermediateQ within the group of inexperienced students, $t(22) = 2.42$, $p = 0.025$, $MD = 0.55$. Moreover, within this group, the result between the PreQ and PostQ just barely missed the level of significance, $t(22) = 2.00$, $p = 0.059$, $MD = 0.41$.

Concerning the difficulty of programming, on average, the experienced students (girls) found programming more difficult throughout the workshop. In PreQ, the experienced students showed a significantly higher level of difficulty for programming compared to the inexperienced students (boys). However, in IntermediateQ and PostQ, no significant result occurred, where experienced students found programming less difficult and inexperienced students found it more difficult (see Table 8.2). Having a closer look into each group of students, a paired-samples t-test yielded a significant result within the group of inexperienced students between the IntermediateQ and PostQ, $t(21) = 2.12$, $p = 0.047$, $MD = 0.43$.

8.4.2.2 Interest.

In order to calculate a measure of students' interest in programming, they responded to the two questions in PreQ (`Q7-Q8`), IntermediateQ, and PostQ (`Q8-Q9`). On average, all students showed a high tendency (all $M > 4$) towards programming and learning how to program from the beginning until the end of the workshop.

With this regard, no significant result occurred between and within each group of students throughout the workshop (see Table 8.2).

Additionally, in the open-ended question "*I find programming...*" in preQ, 26 students associated programming with a positive attitude, except one who called it "*boring*", and one did not give any answer, both inexperienced (boys). Most of students (16) found it "*very interesting*", "*interesting*", or "*fascinating*"; others responded: "*cool*", "*great*", "*good*", etc. Four students additionally reasoned that programming allows them to be creative; for example "*good and interesting, because you can let your creativity run freely*". These four were all part of the experienced students (girls) group. This question was not asked again in IntermediateQ and PostQ, and it was replaced by the two questions regarding the block-based programming environment.

8.4.2.3 Enjoyment.

With respect to students' enjoyment, we measured their responses to the three questions in PreQ (`Q3-Q5`), IntermediateQ, and PostQ (`Q4-Q6`). In all questionnaires, students indicated a high level of enjoyment for programming in terms of "*programming is fun*", and "*I like programming*", all $M > 4$ (see Table 8.2). There were no statistical differences between and within each group of students throughout the workshop. When the students were asked whether they are excited about this workshop, in PreQ, experienced students (girls) indicated a broad approval, while inexperienced students (boys) were undecided (see Table 8.2). In contrast, a paired-samples t-test showed that in PostQ, inexperienced students indicated that they are excited about the workshop significantly more than what they showed at the beginning, $t(21) = 2.12$, $p = 0.047$, $MD = 0.43$. Experienced students showed lower level of excitement in IntermediateQ compared to the beginning and to the end of the workshop; no significant result occurred within this group of students.

Students were also required to respond to the open-ended question "*what do you think of this workshop?*" in PreQ, and "*what do you particularly like about the first workshop day?*" in IntermediateQ, and "*what did you particularly like about the workshop?*" in PostQ. In preQ, ten inexperienced students (boys) did not answer the question, or wrote that they do not know it. Among the remaining 18 students, three of the experienced students (girls) mentioned that they "*appreciate the workshop offer*"; the other three were "*excited*" or "*interested*" about the workshop. Other inexperienced students (12) addressed different thoughts; for instance, "*the importance of programming for their future*", or "*having insights into smart homes*". Overall, their expectation towards the workshop appeared positive. 25 students answered the question in IntermediateQ. Working with "*LED lights*" or "*displays*" was mentioned by eight students (all inexperienced), and "*programming*" was addressed

by seven of them (one experienced, and seven inexperienced). The scope of action (e.g., opportunities to be creative) was mentioned by three (all experienced); other answers (seven) were "*fun*", "*everything*", or "*the instruction and explanation*". In PostQ, five students mentioned the "*scope of freedom in working*" (two experienced, and three inexperienced), five "*programming*" (two experienced, and three inexperienced), seven "*constructing or decorating the tangible objects*" (one experienced, and six inexperienced), and four addressed the "*smart home*" (one experienced, and three inexperienced). Other students (five) wrote other statements, such as the "*the programming environment*" (two), "*explanation*" (two), or "*everything*" (one); two students did not answer. Furthermore, in IntermediateQ, students were asked "*what did you particularly dislike about the first workshop day?*", and in PostQ they were asked "*what did you particularly dislike about the workshop?*". In IntermediateQ, all experienced students (girls) complained about the "*length*", too much "*repetition*", or what they called "*boring*"; for example, they mentioned that they have done it (working with micro-controllers and LED lights) one time before. Among the inexperienced students (boys), six complained about "*too much explanation*", and that they would like to have "*more experiment*". In PostQ, experienced students mostly complained about the "*missing variety of tasks*"; for example, they indicated that it is better to divide the programming tasks, or provide more tasks. Likewise, two inexperienced students still were not happy with "*too much explanation*". Overall, we can conclude that the opportunity to work on personally meaningful projects within the smart home, as well as programming caused the enjoyment. However, experienced students were not happy with the variety and level of programming tasks in the workshop.

8.4.3 Programming Experience

8.4.3.1 Students' experience with the programming environment

The IntermediateQ and PostQ included eight questions asking students to reflect on how they perceive the ease-of-use (`Q10-Q11`), ease-of-learning (`Q12-Q13`), usefulness (`Q14-Q15`), and satisfaction of the programming environment (`Q16-Q17`).

With respect to the ease-of-use, no significant result occurred with respect to the students' prior experience, neither in the IntermediateQ nor in the PostQ (see Table 8.3). However, within each group of students, the ease-of-use was rated significantly higher in the PostQ compared to the IntermediateQ. A paired-samples t-test showed that both experienced (girls) and inexperienced students (boys) liked to use the programming environment more in the PostQ compared to the IntermediateQ, $t(6) = 3.16$, $p = 0.025$, $MD = 0.67$, and $t(22) = 3.78$, $p = 0.001$, $MD = 0.64$, respectively. Similarly, experienced students thought that the programming environment

Table 8.3: Students' Experience of Using the Block-based Programming Application

Questionnaires		Ease-of-Use		Ease-of-Learning		Usefulness		Satisfaction	
		Like to use it	It is easy to use	It is easy to learn	Written instruction is not needed	Complete tasks quickly	It is useful	It is great	I am satisfied with it
IntermediateQ	*Experienced M (SD)*	4.00 (0.63)	4.00 (0.89)	4.33 (0.82)	4.00 (1.26)	3.33 (0.52)	4.33 (0.52)	4.00 (1.10)	4.33 (0.52)
	Inexperienced M (SD)	3.91 (0.75)	4.05 (0.79)	4.23 (0.81)	4.05 (1.00)	3.95 (0.90)	4.27 (0.77)	4.23 (0.69)	4.27 (0.70)
	ANOVA Results	F <1	F <1	F <1	F <1	F(1,26) = 2.59, p = 0.12	F <1	F <1	F <1
PostQ	*Experienced M (SD)*	4.67 (0.52)	4.83 (0.41)	4.50 (0.55)	3.50 (0.55)	3.66 (1.03)	4.33 (0.82)	4.33 (0.52)	4.67 (0.82)
	Inexperienced M (SD)	4.55 (0.60)	4.41 (0.73)	4.50 (0.67)	4.32 (0.72)	4.27 (0.63)	4.36 (0.66)	4.45 (0.60)	4.45 (0.60)
	ANOVA Results	F <1	F(1,26) = 1.82, p = 0.19	F <1	F(1,26) = 6.69, p = 0.016	F(1,26) = 3.29, p = 0.081	F <1	F <1	F <1

M: Mean **SD**: Standard Deviation **F**: F-distribution **p**: p-value

is significantly easier to use in the PostQ in comparison to the IntermediateQ, $t(6) = 2.71$, $p = 0.042$, $MD = 0.83$. However, no significant difference was observed for inexperienced students from the IntermediateQ towards the PostQ, $t(22) = 1.79$, $p = 0.088$, $MD = 0.36$.

Concerning the ease-of-learning, in general, students mentioned that they could easily learn how to use the programming environment (all $M > 4$). No significant result was obtained between and within each group of students in both IntermediateQ and PostQ. Concerning the use of the programming environment without instruction, both group of experienced (girls) and inexperienced students (boys) showed a high tendency towards using the environment without instruction in IntermediateQ; no significant result was obtained within each group of students. In PostQ, inexperienced students rated this question significantly higher compared to the experienced students (see Table 8.3). Descriptively, experienced students would prefer to use the environment together with a written instruction more than inexperienced students.

With respect to the usefulness of programming environment, on average, all students found it highly useful (all $M > 4$). No significant differences were obtained between and within each group of students in both IntermediateQ and PostQ. With respect to the question whether they are able to complete the tasks quickly with the programming environment, the inexperienced students (boys) showed (descriptively) broad approval, while the experienced students (girls) were undecided in both IntermediateQ and Post Q. Likewise, no significant difference was observed between and within each group of students, neither in IntermediateQ nor in PostQ (see Table 8.3).

Concerning the students' satisfaction, all students indicated a high level of satisfaction after using the block-based programming environment (all $M > 4$); no significant difference was observed concerning the students prior experience, neither in the IntermediateQ nor in the PostQ. Likewise, no significant result occurred within each group of students from the IntermediateQ towards the PostQ (see Table 8.3).

With respect to the block-based programming environment, the following open-ended questions were also asked in IntermediateQ and in PostQ: "*what do you like about the programming environment?*", and "*what do you dislike about the programming environment?*". In IntermediateQ, being "*easy to use*", or "*easy to understand*" was mentioned by five students as a positive aspect of the programming environment. Seven students appreciated the "*clarity*", or "*arrangement*", two did not answer, and two said "*nothing*". Other (twelve) answers included "*blocks*", "*everything*", and "*not complex*". In PostQ, twelve students mentioned that it is easy to understand, learn, use, or program, five indicated that they liked "*everything*", and three liked the "*clarity*" or "*arrangement*". Other answers (five) included the "*scope of action*" or "*programming*"; three students did not answer this question. Furthermore, with respect to what the students dislike about the programming environment, in IntermediateQ, seven students did not answer the question or wrote that they do not know, and four indicated that it was "*complicated*", or "*difficult*". Five students mentioned "*specific functionality issue*" (e.g., too simple), and six addressed "*usability issues*" (e.g., block labels are sometimes unclear). Others (six) answered "*nothing*" to this question. In PostQ, eleven students did not answer or wrote that they do not know, and seven mentioned "*nothing*". Four students indicated "*specific functionality issue*" (e.g., sometimes it is difficult to find blocks), and three mentioned "*usability issue*" (e.g., the meaning of some block labels are unclear), and three students complained about the "*complexity*". Across the four questions regarding the programming environment, we could not make a clear distinction between the answers given by experienced (girls) and inexperienced students (boys).

The qualitative results support the notion that programming and using blocks were respectively perceived easier and more useful towards the end of the workshop. Although the question was to find out students' experience with the programming environment, these results reflect their overall programming experience and learning. This is in line with their performance in PrePQ and PostPQ — as their performance increased in PostPQ, the programming environment was perceived easier to use and to understand.

8.4.3.2 Programming a tangible and smart object

Concerning programming a tangible object (`Q1` in IntermediateQ), both experienced (girls) and inexperienced students (boys) indicated a high level of usefulness of programming a tangible object, $M = 4.00 \mid SD = 0.63$, and $M = 4.36 \mid SD = 0.79$, respectively; no significant difference was observed. Likewise, students found programming a real smart object useful in the PostQ (`Q1`); $M = 4.50 \mid SD = 0.55$ for experienced, and $M = 4.41 \mid SD = 0.73$ for inexperienced students. No significant

difference occurred neither between nor within each group of students. With respect to these scores, all students in both groups indicated a higher (descriptively) level of usefulness of programming a real smart object in PostQ compared to programming a tangible object in IntermediateQ.

8.5 Integration and Discussion

One of the main contributions of this study is to show how programming a tangible object and then make it smart in the context of smart homes influences young learners' attitude towards programming. With respect to the enjoyment, the result shows that experienced students (girls), did not enjoy (descriptively) the first day of the workshop. It also shows that the introduction to programming and basic components (e.g., micro-controller, LED lights and OLED displays), as well as seeing the impacts of programming in a tangible object reduced their excitement about the workshop. This finding is in line with the outcome of the question regarding their confidence, where their level of confidence in being successful in this workshop also decreased (descriptively). However, the results show that their level of enjoyment and confidence increased at the end of the workshop. The qualitative outcome shows that among the experienced students, feeling "*bored*" affects their enjoyment of the first day of the workshop. However, having an opportunity to construct a *smart-lighting object* in the context of smart homes was enjoyed the most among them during the second day. We can conclude that the implementation of programming concepts into tangibles and the construction of a smart object in the context of smart homes can foster their confidence and increase the level of enjoyment of the workshop.

In contrast, inexperienced students (boys) showed different attitudes regarding confidence and enjoyment. Our findings show that their level of excitement about the workshop, as well as their confidence in being successful in the workshop increased from the beginning towards the end of the workshop. The qualitative results show that improvement in the level of enjoyment among the inexperienced students is due to having an opportunity to program a tangible object during the first day. This improvement continue during the second day, as they could work creatively on constructing the smart-lighting object and apply their newly gained programming skills to it. As there was some evidence in the students' comments, it seems substantial enough to report that the differences between groups are due to their prior experience. In line with this result is the finding by [KP05] that young learners' performance and interest towards programming depend on their prior experience rather than gender. Please note that all students had prior experience with block-based programming environments. Additionally, experienced students (girls)

previously worked with tangibles (in this case, Arduino boards and LED lights), while inexperienced students (boys) did not work with them.

The results obtained with respect to the students' confidence in programming are encouraging, as experienced students (girls) found programming significantly more difficult compared to the inexperienced students (boys), at the beginning of the workshop. Likewise, at the beginning of the workshop, inexperienced students found themselves significantly better in programming compared to the experienced students. Although the level of inexperienced students' confidence in programming was higher from the beginning towards the end of the workshop, no significant results occurred at the middle and at the end of it, where experienced students found themselves significantly better in programming. This finding shows that our study was successful to establish a clearer picture of programming for both groups of students. Furthermore, the slope from beginning to the end of the workshop indicates that inexperienced students found programming significantly more difficult. However, they still rate themselves better (descriptively) in programming at the end of the workshop compared to the beginning. This result is in line with the learners' programming performance that shows a significant difference within the group of experienced students, and between two groups at the end of the workshop, while it was not significant at the beginning. This is supporting the results presented in [Bei05, Sny14, GC02] that girls come to the technical courses with less confidence, but their confidence level increases with experience.

With respect to the students experience while using the block-based programming environment, descriptively, all students found it more useful and showed a higher level of satisfaction at the end of the workshop compared to the beginning of it. Experienced students (girls) indicated that they prefer a written instruction to use the environment at the end of the workshop, while inexperienced students (boys) found it easy to learn, and they indicated that a written instruction is not needed in order to use the environment. Having a closer look to the results obtained from our study, we found that all students found the environment significantly easier to use at the end of the workshop. It shows that experienced students found the environment easy to use and they like to use it, but with a written instruction, while inexperienced students do not need the instruction to use the environment.

In this study, young learners were enabled to use block-based programming to learn basic programming concepts, and implement these concepts into a tangible object in order to construct a personally meaningful *smart-lighting object* in the context of a smart home. Our main take-home message is that by experiencing the application of programming in the context of state-of-the-art smart technologies, the young learners may have a clearer picture of what programming is and what are the capabilities of it to have significant impacts in their daily life. Moreover, while it is not possible to trace our result back to the programming of a real object embedded

in smart home or smart object construction activities, it might be possible that the usage of these had a positive influence on the learners' confidence, enjoyment, and interest in dealing with programming, as well as enable them to link their creativity to the technologies which are available in smart homes. Therefore, we conclude that students can benefit from situating block-based programming and smart object construction activities in the context of real life-size smart homes, concerning their programming experience and attitudes towards programming. Our results suggest that the tight connection of the programming workshop to a real life-size smart home can especially improve the learners' confidence and programming skills.

8.5.1 Limitations

There are several limitations of our study. The first limitation relates to the gender disparity of our participants. As the recruitment for our programming workshop was out of the researchers' control, not much could be done to tackle this issue. A second similar limitation of this study is that all of participants came from the same school and all were interested to learn programming. The lack of diversity in participants' background could have affected our results. Although this can be considered as minor limitation, we seek to address it in our future work. Our sample size (28 8^{th} grade students) is the third similar limitation which is too small to generalize the findings of this study on a large scale. While we tend to expand the scope of this work, having a larger sample size is our major concern which needs to be addressed in future iterations.

The second limitation of this study is related to the number of programming tasks, and the period of the programming workshop. The findings of this study are limited to the diversity of the programming tasks and period of the programming workshop. Thus, it is an interesting approach for future work to find out whether further number of sessions would have any effect on the findings.

The third and final limitation of this study relates to the control group. This is a major concern for future work in order to find out the impacts of block-based programming on young learners' performance and attitude over time without using smart homes. Similarly, using text-based programming environments in the context of smart homes can be another pathway of future work to evaluate the young learners' trajectories of attitude towards programming and performance throughout a programming course.

8.6 Conclusion

With the increasing usage of block-based programming to design introductory programming environments, it is becoming increasingly important to not only introduce

young learners to programming but also show them the application of it in real life. The evaluation of young learners' performance and attitude before and after integrating block-based programming into the hot topic of smart homes allows us to compare their understanding of basic programming concepts and attitude towards programming. With respect to our results, we conclude that the infrastructures provided by smart homes can be beneficial for introductory programming courses and workshops as a meaningful area of application in both CSE and CHI research communities. Through analysis of the learners' responses to repeated questionnaires and programming questions, we are starting to learn how their attitude and performance changed over time when they used block-based programming alone compared to using it to construct a *smart-lightning object* in the context of smart homes. The next step is to apply these findings to design and implement more introductory programming courses and environments, enabling young learners to rapidly prototype and experiment with new applications for smart environments. However, in order to enable the learners to experience new technologies, learn programming and see its impacts in a real tangible object, the present study establishes a sound basis.

Chapter 9

Conclusion

In this thesis, a novel experience-based educational approach was presented. This approach enables researchers and educators to evaluate the engagement of inexperienced and young learners within the computer science discipline from two perspectives: (i) their acquisition of programming skills, and (ii) their perceptions and attitudes towards programming and computer science, more broadly. We designed, developed and implemented several non-formal programming training sessions (from 2-hour to 4-day) in order to foster the learners' programming skills and improve their attitudes towards programming.

The former approach provides young learners with a computer programming area that requires them to experience smart tangible objects (e.g., toy robots and doll houses) but not smart life-size environments (e.g., smart homes and living labs). Thus, in case of the existence of a motivating context where the learners can actively participate and experience latest research efforts on a real-world environment and learn about the future, it is the only approach. We showed how visual block-based programming enables inexperienced and young learners to learn and author programs which is applied in real life-size smart environments, mobile robots and micro-controllers. Later in this thesis, we introduced a methodology to analyze the learner's trajectories of attitudes towards programming and their programming performance over time. The required data is collected via repeated questionnaires and programming questions from the beginning towards the end of each experience.

In comparison to the previous work, the main advantages of the proposed approach are as follows:

- Showing the potential for using educational block-based programming environments to make modern and powerful technologies accessible for inexperienced and young learners.

- Using the programming environments in order to help the learners to solve programming problems in the context of real-world environments and tangible objects.
- Development, implementation, and evaluation of non-formal programming training sessions, with the goal to teach basic programming concepts to the learners, and to arouse their interest in programming and computer science.
- Showing how the learners' programming performance and attitude towards programming change over time, when they apply their new gained programming skills in a tangible object and make it smart in the context of real life-size smart homes.
- Studying the construction process of smart and tangible objects as interactive artifacts with physical computing material. These artifacts are integrated into real life-size smart homes in order to engage the learners with a focus on their programming performance and attitude.

Apart from the aforementioned advantages, we introduced a block-based rapid programming tool for educators and researchers to help learners to have a short time span between the development of ideas and their implementation in real-world environments. Using the proposed tool leverages the interest of learners for programming and helps them by adequate computational supports. Apart from the development and implementation of this tool , its effectiveness and ease of use were evaluated, thereby comparing two helping features (supplementary documents) to support learners, namely presenting examples and explaining procedures.

The information related to the comparison of Scratch with Google Blockly in fostering the learners' programming skills and improving their attitudes towards programming was presented. These two block-based programming editors have been chosen as (i) Scratch is popular in current educational use of block-based programming, and (ii) Google Blockly is used in design and development of BEESM. Thus, it can be used by future researchers and educators as a starting point to describe under what circumstances a given block-based programming editor is a better choice to foster interest and programming skills among inexperienced and young learners, especially girls.

The findings from an investigation of how implementation of programming in a real object and make it smart can leverage the young learner' interest in programming, and, at the same time, support the acquisition of programming skills were presented. During longer (2- and 4-day) periods of programming training sessions, learners were introduced to basic programming concepts based on block-based programming, and learned how to implement them in a real object to make it smart. This explicit intervention allows educators and researchers to quickly adopt their

training sessions to provide more opportunities for their learners to learn programming and apply new gained skills to construct a smart tangible object.

In the case of evaluating inexperienced and young learners' programming performance and attitudes towards programming (which is the primary goal of this thesis) and in comparison to the existing approaches, our approach has four main advantages as follows:

(1) All programming questions are slightly different from each other in all training sessions to ensure that learners read all questions carefully and identify the correct solution based on the question.

(2) In addition to prepared slides (for oral explanation), we used supplementary documents (including an explanation of necessary blocks and hardware components) in order to help the learners to work with the programming environment and components, and minimize and control the instructor effects.

(3) In addition to the quantitative data (which is collected through close-ended questions), we collected and analyzed qualitative data using open-ended questions to support the quantitative data.

(4) Learners' attitudes, programming performance and experience were evaluated in the context of smart homes, smart tangible artifacts and a combination of both of them, using block-based programming environments.

Therefore, concerning the main emphasis of this thesis, an extensive educational block-based programming approach was presented which extends the application area of designing smart objects to the state-of-the-art area of real-world smart environments (in Chapter 3). This approach shows the potential for using block-based programming to make state-of-the-art smart technologies accessible for inexperienced and young learners. Furthermore, regarding the primary goal of this thesis, we introduced the learners to personally meaningful state-of-the-art smart technologies in the context of smart homes. This motivates them to begin with programming activities and to take part in learning programming in the future.

With respect to gender and prior experience, the results (in Chapter 5) showed that inexperienced girls performed better in order to solve programming problems. However, inexperienced boys showed a higher interest in learning programming. In contrast, experienced girls both performed better and showed a hight tendency towards programming learning opportunities than experienced boys. Furthermore, we had a closer look (in Chapter 7) into the programming performance and attitude towards programming among both experienced and inexperienced girls. Experienced girls showed higher interest in the construction of smart objects, while inexperienced

girls were impressed by learning basic programming concepts with block-based programming. In addition, programming performance increased among inexperienced girls when they showed a higher level of confidence after working with the block-based programming environment. Similarly, when it came to construction of a smart object which is integrated into the smart homes (in Chapter 8), experienced students (girls) did not show a high level of enjoyment and confidence when they only worked with the programming environments. However, when they were enabled to apply their gained programming skills into a tangible object and make it smart, they showed a higher level of confidence and enjoyment. In general, we conclude that programming performance and attitude towards programming is more based on young learners' prior experience than gender difference.

Being well aware that many researchers and educators do not have immediate access to real life-size smart environments, we aim to introduce our learners to an innovative, unique environment which provides possibilities of how computer science is connected to real-world environments and to their daily life. We support learners and motivate them to learn general purposes of programming via a visual block-based programming tool. This enables them to program and see its impacts on real life-size smart homes, mobile robots and micro-controllers. The results show that with this tool, learners are able to program successfully in the context of smart and tangible objects and environments by themselves. In this respect, visual block-based programming environments are suitable to simplify programming and to remove difficulties for inexperienced and young learners. Additionally, we can conclude that smart objects and homes provide a motivating and fascinating context for the learners to begin with programming activities and experience new technologies.

We would also like to emphasize that in our studies, we directly asked school teachers to come to us with their interested students. In this respect, teachers need to allocate a time slot in order to come with a group of students, limiting the possible number of students attending our training sessions and working with the programming environment.

Future studies with a larger sample size should focus on possible influences of young learners' gender and competence levels (e.g., prior programming knowledge) with respect to block-based programming and tangible smart objects and environments. In future work, young learners should be enabled to practice self-paced and to perform more creative tasks in a longer period of time, following their own ideas in the smart tangible objects and environments (i.e., formal programming training sessions as part of the students' regular school curriculum). Including self-paced and creative tasks will also help to better adapt the training sessions to different levels of students' prior knowledge, and it becomes easier to capture the interest and performance of the individual learners. In future iterations of this work, we need control groups, including students with different gender, prior programming

experience, socio-economic status and from different reigns in order to generalize the findings. Including a control group will help us to have two groups of participants in each phase of the research. For instance, one group starts with programming activities, using block-based programming and smart tangible objects and environments, while the other group uses neither block-based programming nor smart tangible objects and environments. Another question for future work is to ascertain when young learners can move from block-based programming environments to pure programming IDEs, using traditional text-based code syntax. More detailed knowledge about learners' performance and their attitudes is necessary in order to adapt the learning environment to individual learning prerequisites in an optimal way. However, in order to enable young learners to experience new technologies, foster their interest in learning programming and see its impacts in real life-size smart environments and objects, this thesis establishes a sound basis.

Bibliography

[ADRW00] Atkinson, Robert K. ; Derry, Sharon J. ; Renkl, Alexander ; Wortham, Donald: Learning from examples: Instructional principles from the worked examples research. In: *Review of educational research* 70 (2000), Nr. 2, S. 181–214

[AH16] Aivaloglou, Efthimia ; Hermans, Felienne: 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*, 2016, S. 53–61

[AMWW15] Ashrov, Adiel ; Marron, Assaf ; Weiss, Gera ; Wiener, Guy: A use-case for behavioral programming: an architecture in JavaScript and Blockly for interactive applications with cross-cutting scenarios. In: *Science of Computer Programming* 98 (2015), S. 268–292

[B+15] Blikstein, Paulo u. a.: Computationally enhanced toolkits for children: historical review and a framework for future design. In: *Foundations and Trends® in Human–Computer Interaction* 9 (2015), Nr. 1, S. 1–68

[Bau15] Bau, David: Droplet, a blocks-based editor for text code. In: *Journal of Computing Sciences in Colleges* 30 (2015), Nr. 6, S. 138–144

[BBDP15] Bau, David ; Bau, D A. ; Dawson, Mathew ; Pickens, C S.: Pencil code: block code for a text world. In: *Proceedings of the 14th International Conference on Interaction Design and Children*, 2015, S. 445–448

[BBE+09] Bruckman, Amy ; Biggers, Maureen ; Ericson, Barbara ; McKlin, Tom ; Dimond, Jill ; DiSalvo, Betsy ; Hewner, Mike ; Ni,

Lijun ; Yardi, Sarita: " Georgia computes!" improving the computing education pipeline. In: *ACM SIGCSE Bulletin* 41 (2009), Nr. 1, S. 86–90

[BECC08] Buechley, Leah ; Eisenberg, Mike ; Catchen, Jaime ; Crockett, Ali: The LilyPad Arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2008, S. 423–432

[BEE06] Buechley, Leah ; Elumeze, Nwanua ; Eisenberg, Michael: Electronic/computational textiles and children's crafts. In: *Proceedings of the 2006 conference on Interaction design and children*, 2006, S. 49–56

[BEE07] Buechley, Leah ; Eisenberg, Mike ; Elumeze, Nwanua: Towards a curriculum for electronic textiles in the high school classroom. In: *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, 2007, S. 28–32

[Bei05] Beisser, Sally R.: An examination of gender differences in elementary constructionist classrooms using Lego/Logo instruction. In: *Computers in the Schools* 22 (2005), Nr. 3-4, S. 7–19

[Ben12] Benitti, Fabiane Barreto V.: Exploring the educational potential of robotics in schools: A systematic review. In: *Computers & Education* 58 (2012), Nr. 3, S. 978–988

[BGK+17] Bau, David ; Gray, Jeff ; Kelleher, Caitlin ; Sheldon, Josh ; Turbak, Franklyn: Learnable programming: blocks and beyond. In: *Communications of the ACM* 60 (2017), Nr. 6, S. 72–80

[BLV+17] Broll, Brian ; Lédeczi, Akos ; Volgyesi, Peter ; Sallai, Janos ; Maroti, Miklos ; Carrillo, Alexia ; Weeden-Wright, Stephanie L. ; Vanags, Chris ; Swartz, Joshua D. ; Lu, Melvin: A visual programming environment for learning distributed programming. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 2017, S. 81–86

[BN84] Birrell, Andrew D. ; Nelson, Bruce J.: Implementing remote procedure calls. In: *ACM Transactions on Computer Systems (TOCS)* 2 (1984), Nr. 1, S. 39–59

[Bra17] Bray, Tim: The javascript object notation (json) data interchange format. 2017. – Forschungsbericht

[BS11] Barr, Valerie ; Stephenson, Chris: Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? In: *Acm Inroads* 2 (2011), Nr. 1, S. 48–54

[CDP00] Cooper, Stephen ; Dann, Wanda ; Pausch, Randy: Alice: a 3-D tool for introductory programming concepts. In: *Journal of computing sciences in colleges* 15 (2000), Nr. 5, S. 107–116

[CGN19] Chu, Sharon L. ; Garcia, Brittany ; Nam, Beth: Understanding Context in Children's Use of Smartwatches for Everyday Science Reflections. In: *Proceedings of the 18th ACM International Conference on Interaction Design and Children*, 2019, S. 83–93

[Che18] CherryPy: *CherryPy Homepage*. 2018. – Retrieved January 1, 2020 from https://cherrypy.org/

[CLKL14] Charters, Polina ; Lee, Michael J. ; Ko, Andrew J. ; Loksa, Dastyni: Challenging stereotypes and changing attitudes: the effect of a brief programming encounter on adults' attitudes toward programming. In: *Proceedings of the 45th ACM technical symposium on Computer science education*, 2014, S. 653–658

[Cof17] Coffey, John W.: A study of the use of a reflective activity to improve students' software design capabilities. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 2017, S. 129–134

[Coh02] Cohoon, J M.: Recruiting and retaining women in undergraduate computing majors. In: *ACM SIGCSE Bulletin* 34 (2002), Nr. 2, S. 48–52

[Com17] Committee, Computer Science Framework S.: *K-12 computer science framework*. 2017. – Retrieved January 10, 2020 from http://www.k12cs.org

[Cor17] Corporation, Microsoft: *Why Europe's girls aren't studying STEM*. 2017. – Retrieved January 10, 2020 from http://hdl.voced.edu.au/10707/427011

[DHB+04] Dey, Anind K. ; Hamid, Raffay ; Beckmann, Chris ; Li, Ian ; Hsu, Daniel: a CAPpella: programming by demonstration of context-aware applications. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2004, S. 33–40

[DLY+06] Davidoff, Scott ; Lee, Min K. ; Yiu, Charles ; Zimmerman, John ; Dey, Anind K.: Principles of smart home control. In: *International conference on ubiquitous computing* Springer, 2006, S. 19–34

[DSK05] Dagdilelis, Vassilios ; Sartatzemi, Maya ; Kagani, Katerina: Teaching (with) robots in secondary schools: some new and not-so-new pedagogical problems. In: *Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05)* IEEE, 2005, S. 757–761

[ELP17] Ertl, Bernhard ; Luttenberger, Silke ; Paechter, Manuela: The impact of gender stereotypes on the self-concept of female students in STEM subjects with an under-representation of females. In: *Frontiers in psychology* 8 (2017), S. 703

[Fet18] Fette, Ian: *The websocket protocol*. 2018. – Retrieved January 10, 2020 from https://tools.ietf.org/html/rfc6455

[Fra14] Fraser, Neil: *Google blockly-a visual programming editor*. 2014. – Retrieved January 10, 2020 from https://developers.google.com/blockly/

[FSK+13] Flannery, Louise P. ; Silverman, Brian ; Kazakoff, Elizabeth R. ; Bers, Marina U. ; Bontá, Paula ; Resnick, Mitchel: Designing ScratchJr: Support for early childhood learning through computer programming. In: *Proceedings of the 12th International Conference on Interaction Design and Children*, 2013, S. 1–10

[FSR+10] Frey, Jochen ; Stahl, Christoph ; Röfer, Thomas ; Krieg-Brückner, Bernd ; Alexandersson, Jan: The DFKI competence center for ambient assisted living. In: *International Joint Conference on Ambient Intelligence* Springer, 2010, S. 310–314

[GBBB19] Gorbacheva, Elena ; Beekhuyzen, Jenine ; Brocke, Jan vom ; Becker, Jörg: Directions for research on gender imbalance in the IT profession. In: *European Journal of Information Systems* 28 (2019), Nr. 1, S. 43–67

[GC02] Gürer, Denise ; Camp, Tracy: An ACM-W literature review on women in computing. In: *ACM SIGCSE Bulletin* 34 (2002), Nr. 2, S. 121–127

[GCNB18] Garcia, Brittany ; Chu, Sharon L. ; Nam, Beth ; Banigan, Colin: Wearables for learning: examining the smartwatch as a tool for situated science reflection. In: *Proceedings of the 2018 CHI conference on human factors in computing systems*, 2018, S. 1–13

[GSH+18] Gutierrez, Francisco J. ; Simmonds, Jocelyn ; Hitschfeld, Nancy ; Casanova, Cecilia ; Sotomayor, Cecilia ; Peña-Araya, Vanessa: Assessing software development skills among K-6 learners in a project-based workshop with scratch. In: *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)* IEEE, 2018, S. 98–107

[HA17] Hermans, Felienne ; Aivaloglou, Efthimia: Teaching software engineering principles to k-12 students: a mooc on scratch. In: *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)* IEEE, 2017, S. 13–22

[HC15] Huang, Justin ; Cakmak, Maya: Supporting mental model accuracy in trigger-action programming. In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015, S. 215–225

[HCB14] Hicks, Andrew ; Cateté, Veronica ; Barnes, Tiffany: Part of the game: Changing level creation to identify and filter low quality user-generated levels. In: *FDG*, 2014

[HH12] Hwang, Amy ; Hoey, Jesse: Smart home, the next generation: Closing the gap between users and technology. In: *2012 AAAI Fall Symposium Series*, 2012

[HH18] Holwerda, Robert ; Hermans, Felienne: A usability analysis of blocks-based programming editors using cognitive dimensions. In: *2018 IEEE symposium on visual languages and human-centric computing (VL/HCC)* IEEE, 2018, S. 217–225

[HHKM17] Heimgaertner, Florian ; Hettich, Stefan ; Kohlbacher, Oliver ; Menth, Michael: Scaling home automation to public buildings: A

distributed multiuser setup for OpenHAB 2. In: *2017 Global Internet of Things Summit (GIoTS)* IEEE, 2017, S. 1–6

[HLC16] Huang, Justin ; Lau, Tessa ; Cakmak, Maya: Design and evaluation of a rapid programming system for service robots. In: *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)* IEEE, 2016, S. 295–302

[HM10] Harvey, Brian ; Mönig, Jens: Bringing "no ceiling" to Scratch: Can one language serve kids and computer scientists. In: *Proc. Constructionism* (2010), S. 1–10

[HMW12] Harel, David ; Marron, Assaf ; Weiss, Gera: Behavioral programming. In: *Communications of the ACM* 55 (2012), Nr. 7, S. 90–100

[Inv18] Inventor, MIT A.: *MIT App Inventor Homepage*. 2018. – Retrieved January 7, 2020 from http://appinventor.mit.edu/explore/

[JKGM18] Jimenez, Yerika ; Kapoor, Amanpreet ; Gardner-McCune, Christina: Usability Challenges that Novice Programmers Experience when Using Scratch for the First Time. In: *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* IEEE, 2018, S. 327–328

[JLY04] Jiang, Li ; Liu, Da-You ; Yang, Bo: Smart home research. In: *Proceedings of 2004 international conference on machine learning and cybernetics (IEEE Cat. No. 04EX826)* Bd. 2 IEEE, 2004, S. 659–663

[KAB+11] Ko, Andrew J. ; Abraham, Robin ; Beckwith, Laura ; Blackwell, Alan ; Burnett, Margaret ; Erwig, Martin ; Scaffidi, Chris ; Lawrance, Joseph ; Lieberman, Henry ; Myers, Brad u. a.: The state of the art in end-user software engineering. In: *ACM Computing Surveys (CSUR)* 43 (2011), Nr. 3, S. 1–44

[Kaf16] Kafai, Yasmin B.: From computational thinking to computational participation in K–12 education. In: *Communications of the ACM* 59 (2016), Nr. 8, S. 26–27

[Kal15] Kalelioğlu, Filiz: A new way of teaching programming skills to K-12 students: Code. org. In: *Computers in Human Behavior* 52 (2015), S. 200–210

[KB13] Kafai, Yasmin B. ; Burke, Quinn: The social turn in K-12 programming: moving from computational thinking to computational participation. In: *Proceeding of the 44th ACM technical symposium on computer science education*, 2013, S. 603–608

[KD18] Katterfeldt, Eva-Sophie ; Dittert, Nadine: Co-designing Smart Home Maker Workshops with Girls. In: *Proceedings of the Conference on Creativity and Making in Education*, 2018, S. 100–101

[KDS09] Katterfeldt, Eva-Sophie ; Dittert, Nadine ; Schelhowe, Heidi: EduWear: smart textiles as ways of relating computing technology to everyday life. In: *Proceedings of the 8th International Conference on Interaction Design and Children*, 2009, S. 9–17

[KDS15] Katterfeldt, Eva-Sophie ; Dittert, Nadine ; Schelhowe, Heidi: Designing digital fabrication learning environments for Bildung: Implications from ten years of physical computing workshops. In: *International Journal of Child-Computer Interaction* 5 (2015), S. 3–10

[KH04] Koenig, Nathan ; Howard, Andrew: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)* Bd. 3 IEEE, 2004, S. 2149–2154

[Kit18] Kit, Smart H.: *LittleBits Homepage*. 2018. – Retrieved January 5, 2020 from https://littlebits.com/blog/introducing-the-smart-home-kit/

[KLPK15] Kam, Hyeong R. ; Lee, Sung-Ho ; Park, Taejung ; Kim, Chang-Hun: RViz: a toolkit for real domain data visualization. In: *Telecommunication Systems* 60 (2015), Nr. 2, S. 337–345

[KLR16] Khan, Nazish Z. ; Luxton-Reilly, Andrew: Is computing for social good the solution to closing the gender gap in computer science? In: *Proceedings of the Australasian Computer Science Week Multiconference*, 2016, S. 1–5

[KLS+14] Kafai, Yasmin B. ; Lee, Eunkyoung ; Searle, Kristin ; Fields, Deborah ; Kaplan, Eliot ; Lui, Debora: A crafts-oriented approach to computing in high school: Introducing computational concepts, practices, and perspectives with electronic textiles. In: *ACM Transactions on Computing Education (TOCE)* 14 (2014), Nr. 1, S. 1–20

[KM06] Ko, Andrew J. ; Myers, Brad A.: Barista: An implementation framework for enabling new tools, interaction techniques and views in code editors. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2006, S. 387–396

[KM16] Kereki, Inés F. ; Manataki, Areti: "Code Yourself" and "A Programar": a bilingual MOOC for teaching Computer Science to teenagers. In: *2016 IEEE Frontiers in Education Conference (FIE)* IEEE, 2016, S. 1–9

[KMA04] Ko, Andrew J. ; Myers, Brad A. ; Aung, Htet H.: Six learning barriers in end-user programming systems. In: *2004 IEEE Symposium on Visual Languages-Human Centric Computing* IEEE, 2004, S. 199–206

[KP05] Kelleher, Caitlin ; Pausch, Randy: Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. In: *ACM Computing Surveys (CSUR)* 37 (2005), Nr. 2, S. 83–137

[Lab18] Lab, Human-Centered R.: *A standalone Blockly programming application, integrated with ROS*. https://github.com/hcrlab/code_it. Version: 2018

[Lag05] Lagesen, Vivian A.: *Extreme Make-over?: The Making of Gender and Computer Science*. STS-report 71/2005. Trondheim: Centre for Technology and Society, Norwegian University of Science and Technology, 2005

[Lag08] Lagesen, Vivian A.: A cyberfeminist utopia? Perceptions of gender and computer science among Malaysian women computer science students and faculty. In: *Science, Technology, & Human Values* 33 (2008), Nr. 1, S. 5–27

[Lav08] Lavrakas, Paul J.: *Encyclopedia of survey research methods*. Sage Publications, 2008

[Lew10] Lewis, Colleen M.: How programming environment shapes perception, learning and goals: logo vs. scratch. In: *Proceedings of the 41st ACM technical symposium on Computer science education*, 2010, S. 346–350

[Lin19] Lin, Fred: *A web-based visual programming editor for Arduino*. https://github.com/BlocklyDuino/BlocklyDuino. Version: 2019

[LK14] Lye, Sze Y. ; Koh, Joyce Hwee L.: Review on teaching and learning of computational thinking through programming: What is next for K-12? In: *Computers in Human Behavior* 41 (2014), S. 51–61

[LPKW06] Lieberman, Henry ; Paternò, Fabio ; Klann, Markus ; Wulf, Volker: End-user development: An emerging paradigm. In: *End user development*. Springer, 2006, S. 1–8

[Mak18] MakeCode: *MakeCode Homepage*. 2018. – Retrieved January 10, 2020 from https://www.microsoft.com/en-us/makecode

[Mak19] Makeblock: *mBlock - The educational programming software*. 2019. – Retrieved January 17, 2020 from http://www.mblock.cc

[Mat04] Mataric, Maja J.: Robotics education for all ages. In: *Proc. AAAI Spring Symposium on Accessible, Hands-on AI and Robotics Education*, 2004

[MB11] Millner, Amon ; Baafi, Edward: Modkit: blending and extending approachable platforms for creating computer programs and interactive objects. In: *Proceedings of the 10th International Conference on Interaction Design and Children*, 2011, S. 250–253

[MCK17] Merkouris, Alexandros ; Chorianopoulos, Konstantinos ; Kameas, Achilles: Teaching programming in secondary education through embodied computing platforms: Robotics and wearables. In: *ACM Transactions on Computing Education (TOCE)* 17 (2017), Nr. 2, S. 1–22

[MFM16] Mahadevan, Anand ; Freeman, Jason ; Magerko, Brian: An interactive, graphical coding environment for EarSketch online using Blockly and Web Audio API. (2016)

[MGB15] Martinez, Cecilia ; Gomez, Marcos J. ; Benotti, Luciana: A comparison of preschool and elementary school children learning computer science concepts through a multilanguage robot programming platform. In: *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, 2015, S. 159–164

[MGG+14] McLaren, Bruce M. ; Gog, Tamara van ; Ganoe, Craig ; Yaron, David ; Karabinos, Michael: Exploring the assistance dilemma: Comparing instructional support in examples and problems. In:

International Conference on Intelligent Tutoring Systems Springer, 2014, S. 354–361

[MI18] Melcer, Edward F. ; Isbister, Katherine: Bots & (Main) frames: exploring the impact of tangible blocks and collaborative play in an educational programming game. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, S. 1–14

[MKP+08] Myers, Brad A. ; Ko, Andrew J. ; Park, Sun Y. ; Stylos, Jeffrey ; LaToza, Thomas D. ; Beaton, Jack: More natural end-user software engineering. In: *Proceedings of the 4th international workshop on End-user software engineering*, 2008, S. 30–34

[ML18] Milne, Lauren R. ; Ladner, Richard E.: Blocks4All: overcoming accessibility barriers to blocks programming for children with visual impairments. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, S. 1–10

[MM19] Michaelis, Joseph E. ; Mutlu, Bilge: Supporting Interest in Science Learning with a Social Robot. In: *Proceedings of the 18th ACM International Conference on Interaction Design and Children*, 2019, S. 71–82

[MPA19] Marikyan, Davit ; Papagiannidis, Savvas ; Alamanos, Eleftherios: A systematic review of the smart home literature: A user perspective. In: *Technological Forecasting and Social Change* 138 (2019), S. 139–154

[MRR+10] Maloney, John ; Resnick, Mitchel ; Rusk, Natalie ; Silverman, Brian ; Eastmond, Evelyn: The scratch programming language and environment. In: *ACM Transactions on Computing Education (TOCE)* 10 (2010), Nr. 4, S. 1–15

[MRT03] Montemerlo, Michael ; Roy, Nicholas ; Thrun, Sebastian: Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)* Bd. 3 IEEE, 2003, S. 2436–2441

[MSABA11] Meerbaum-Salant, Orni ; Armoni, Michal ; Ben-Ari, Mordechai: Habits of programming in scratch. In: *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, 2011, S. 168–172

[MSS+13] Mubin, Omar ; Stevens, Catherine J. ; Shahid, Suleman ; Al Mahmud, Abdullah ; Dong, Jian-Jie: A review of the applicability of robots in education. In: *Journal of Technology in Education and Learning* 1 (2013), Nr. 209-0015, S. 13

[MWW12] Marron, Assaf ; Weiss, Gera ; Wiener, Guy: A decentralized approach for programming interactive applications with javascript and blockly. In: *Proceedings of the 2nd edition on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions*. 2012, S. 59–70

[Mye86] Myers, Brad A.: Visual programming, programming by example, and program visualization: a taxonomy. In: *ACM sigchi bulletin* 17 (1986), Nr. 4, S. 59–66

[Mye90] Myers, Brad A.: Taxonomies of visual programming and program visualization. In: *Journal of Visual Languages & Computing* 1 (1990), Nr. 1, S. 97–123

[NHCW04] Nourbakhsh, Illah R. ; Hamner, Emily ; Crowley, Kevin ; Wilkinson, Katie: Formal measures of learning in a secondary school mobile robotics course. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004* Bd. 2 IEEE, 2004, S. 1831–1836

[PA19] Pereira Atencio, Carlos: *Ardublockly*. 2019. – Retrieved January 15, 2020 from https://ardublockly.embeddedlog.com

[Pap80] Papert, Seymour: *Mindstorms: Children, computers, and powerful ideas*. Basic Books, Inc., 1980

[Pap02] Papert, Seymour: *Hard Fun*. 2002. – Retrieved January 8, 2020 from http://www.papert.org/articles/HardFun.html

[PB15] Price, Thomas W. ; Barnes, Tiffany: Comparing textual and block interfaces in a novice programming environment. In: *Proceedings of the eleventh annual international conference on international computing education research*, 2015, S. 91–99

[PEH07] Powers, Kris ; Ecott, Stacey ; Hirshfield, Leanne M.: Through the looking glass: teaching CS0 with Alice. In: *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, 2007, S. 213–217

[PH07] Parsons, Dale ; Haden, Patricia: Programming osmosis: Knowledge transfer from imperative to visual programming environments. In: *Procedings of The Twentieth Annual NACCQ Conference*, 2007, S. 209–215

[PHEC17] Paramasivam, Vivek ; Huang, Justin ; Elliott, Sarah ; Cakmak, Maya: Computer science outreach with end-user robot-programming tools. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 2017, S. 447–452

[PP03] Player ; Projects, Stage: *GAZEBO Homepage*. 2003. – Retrieved January 21, 2020 from http://gazebosim.org/

[PR12] Przybylla, Mareen ; Romeike, Ralf: My Interactive Garden–A Constructionist Approach to Creative Learning with Interactive Installations in Computing Education. In: *Constructionism: Theory, Practice and Impact. Proceedings of Constructionism* 2012 (2012), S. 395–404

[PR14] Przybylla, Mareen ; Romeike, Ralf: Physical Computing and Its Scope–Towards a Constructionist Computer Science Curriculum with Physical Computing. In: *Informatics in Education* 13 (2014), Nr. 2, S. 241–254

[QBBD13] Qiu, Kanjun ; Buechley, Leah ; Baafi, Edward ; Dubow, Wendy: A curriculum for teaching computer science through computational textiles. In: *Proceedings of the 12th international conference on interaction design and children*, 2013, S. 20–27

[QCG+09] Quigley, Morgan ; Conley, Ken ; Gerkey, Brian ; Faust, Josh ; Foote, Tully ; Leibs, Jeremy ; Wheeler, Rob ; Ng, Andrew Y.: ROS: an open-source Robot Operating System. In: *ICRA workshop on open source software* Bd. 3 Kobe, Japan, 2009, S. 5

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

[RA10] Reddy, Y M. ; Andrade, Heidi: A review of rubric use in higher education. In: *Assessment & evaluation in higher education* 35 (2010), Nr. 4, S. 435–448

[RI06] Repenning, Alexander ; Ioannidou, Andri: What makes end-user development tick? 13 design guidelines. In: *End user development*. Springer, 2006, S. 51–85

[RMB+98] Resnick, Mitchel ; Martin, Fred ; Berg, Robert ; Borovoy, Rick ; Colella, Vanessa ; Kramer, Kwin ; Silverman, Brian: Digital manipulatives: new toys to think with. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1998, S. 281–287

[RMH14] Ruf, Alexander ; Mühling, Andreas ; Hubwieser, Peter: Scratch vs. Karel: impact on learning outcomes and motivation. In: *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, 2014, S. 50–59

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

[RMSS96] Resnick, Mitchel ; Martin, Fred ; Sargent, Randy ; Silverman, Brian: Programmable bricks: Toys to think with. In: *IBM Systems journal* 35 (1996), Nr. 3.4, S. 443–452

[Rom07] Romeike, Ralf: Applying creativity in CS high school education: criteria, teaching example and evaluation. In: *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research-Volume 88*, 2007, S. 87–96

[SB16] Sullivan, Amanda ; Bers, Marina U.: Girls, boys, and bots: Gender differences in young children's performance on robotics and programming tasks. In: *Journal of Information Technology Education: Innovations in Practice* 15 (2016), S. 145–165

[SDP+15] Spyropoulou, Natalia ; Demopoulou, Gerasimoula ; Pierrakeas, Christos ; Koutsonikos, Ioannis ; Kameas, Achilles: Developing a computer programming mooc. In: *Procedia Computer Science* 65 (2015), S. 182–191

[SKL16] Shim, Jaekwoun ; Kwon, Daiyoung ; Lee, Wongyu: The effects of a robot game environment on computer programming education for elementary school students. In: *IEEE Transactions on Education* 60 (2016), Nr. 2, S. 164–172

[SKR+10] Salden, Ron J. ; Koedinger, Kenneth R. ; Renkl, Alexander ; Aleven, Vincent ; McLaren, Bruce M.: Accounting for beneficial effects of worked examples in tutored problem solving. In: *Educational Psychology Review* 22 (2010), Nr. 4, S. 379–392

[Sna19] Snap4Arduino: *Snap4Arduino Homepage*. 2019. – Retrieved January 6, 2020 from http://snap4arduino.rocks

[Sny14] Snyder, Thomas D.: Mobile Digest of Education Statistics, 2013. NCES 2014-086. In: *National Center for Education Statistics* (2014)

[Str09] Strecker, Kerstin M.: *Informatik für Alle-wie viel Programmierung braucht der Mensch?*, University of Göttingen, Diss., 2009

[SVMP98] Sweller, John ; Van Merrienboer, Jeroen J. ; Paas, Fred G.: Cognitive architecture and instructional design. In: *Educational psychology review* 10 (1998), Nr. 3, S. 251–296

[SWYM17] Sentance, Sue ; Waite, Jane ; Yeomans, Lucy ; MacLeod, Emily: Teaching with physical computing devices: the BBC micro: bit initiative. In: *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, 2017, S. 87–96

[TPO12] Teclaw, Robert ; Price, Mark C. ; Osatuke, Katerine: Demographic question placement: Effect on item response rates and means of a veterans health administration survey. In: *Journal of Business and Psychology* 27 (2012), Nr. 3, S. 281–290

[UMPYHL14] Ur, Blase ; McManus, Elyse ; Pak Yong Ho, Melwyn ; Littman, Michael L.: Practical trigger-action programming in the smart home. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, S. 803–812

[WAS+18] Weintrop, David ; Afzal, Afsoon ; Salac, Jean ; Francis, Patrick ; Li, Boyang ; Shepherd, David C. ; Franklin, Diana: Evaluating CoBlox: A comparative study of robotics programming environments for adult novices. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, S. 1–12

[Wei19] Weintrop, David: Block-based programming in computer science education. In: *Communications of the ACM* 62 (2019), Nr. 8, S. 22–25

[WH17] Weintrop, David ; Holbert, Nathan: From blocks to text and back: Programming patterns in a dual-modality environment. In: *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, 2017, S. 633–638

[WHHF18] Weintrop, David ; Hansen, Alexandria K. ; Harlow, Danielle B. ; Franklin, Diana: Starting from Scratch: Outcomes of early computer science learning experiences and implications for what comes next. In: *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018, S. 142–150

[WL15] Woo, Jong-bum ; Lim, Youn-kyung: User experience in do-it-yourself-style smart homes. In: *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*, 2015, S. 779–790

[WR08] Wittwer, Joerg ; Renkl, Alexander: Why instructional explanations often do not work: A framework for understanding the effectiveness of instructional explanations. In: *Educational Psychologist* 43 (2008), Nr. 1, S. 49–64

[WW15] Weintrop, David ; Wilensky, Uri: Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. In: *ICER* Bd. 15, 2015, S. 101–110

[WW17a] Weintrop, David ; Wilensky, Uri: Between a block and a typeface: Designing and evaluating hybrid programming environments. In: *Proceedings of the 2017 conference on interaction design and children*, 2017, S. 183–192

[WW17b] Weintrop, David ; Wilensky, Uri: Comparing block-based and text-based programming in high school computer science classrooms. In: *ACM Transactions on Computing Education (TOCE)* 18 (2017), Nr. 1, S. 1–25

[Zim17] Zimmerman, Bryant: Programming education for students of any age: LEGO® education-mindstorms® EV3 robotics. In: *Journal of Computing Sciences in Colleges* 33 (2017), Nr. 1, S. 42–43

[ZLP18] Zhi, Rui ; Lytle, Nicholas ; Price, Thomas W.: Exploring instructional support design in an educational game for K-12 computing education. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 2018, S. 747–752

Appendix A

Questionnaires and Instructional Materials

A.1 Pre-questionnaire

Since we want to ask you both before and after the course, we need a code to assign your answers. This is necessary so that the survey can be done anonymously.

What is the first letter of your mother's first name?	☐
What is the second letter of your father's first name?	☐
What is the third letter of your first name?	☐
Which day is your birthday? (for example, 27, if you were born on 27/03/2000)	☐

Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

	Yes				**No**
Do you think that computer science is difficult to understand?	☐	☐	☐	☐	☐
Would you like to learn how to program?	☐	☐	☐	☐	☐
	1	2	3	4	5
What grade would you give to your programming skills?	☐	☐	☐	☐	☐

A.2 Post-questionnaire

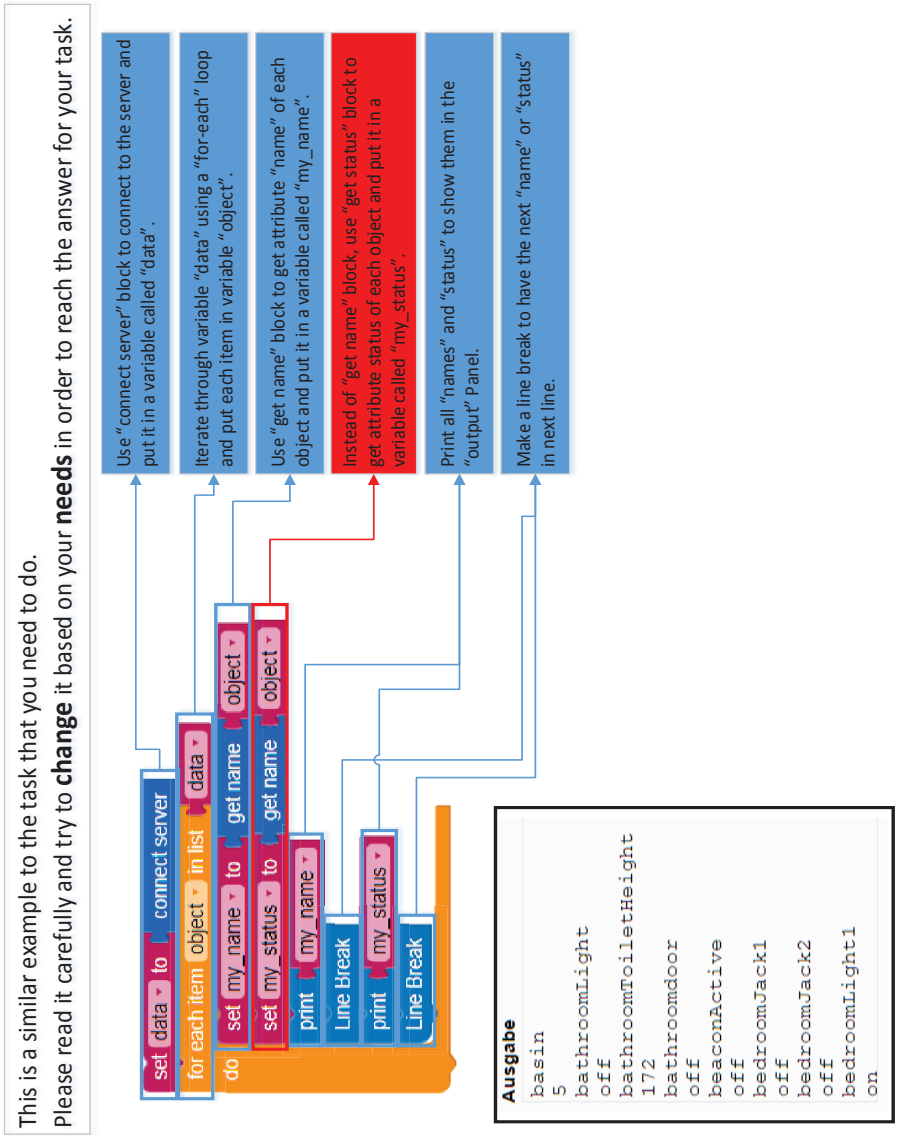
In order to conduct the survey anonymously, we need your code again:

What is the first letter of your mother's first name?	☐
What is the second letter of your father's first name?	☐
What is the third letter of your first name?	☐
Which day is your birthday? (for example, 27, if you were born on 27/03/2000)	☐

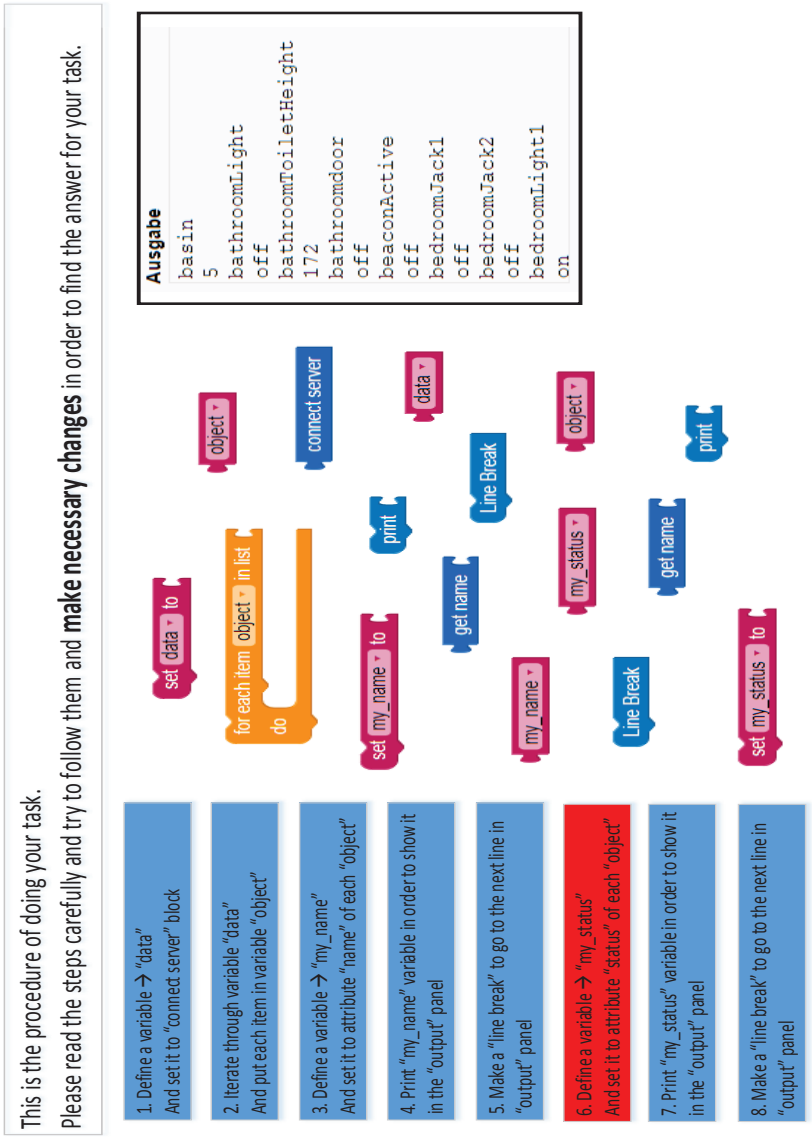
Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

	Yes				**No**
Is it easy to program with blocks?	☐	☐	☐	☐	☐
Do blocks help you to easily understand programs?	☐	☐	☐	☐	☐
Do you think that it is helpful to be able to see directly in reality whether the program works as desired?	☐	☐	☐	☐	☐
Do you think computer science is difficult to understand?	☐	☐	☐	☐	☐
Would you like to learn how to program?	☐	☐	☐	☐	☐
	Block				**Code**
Do you prefer to program with block or directly with code syntax?	☐	☐	☐	☐	☐
Are you a boy or a girl?	☐ Girl ☐ Boy				
How old are you? ..... years					

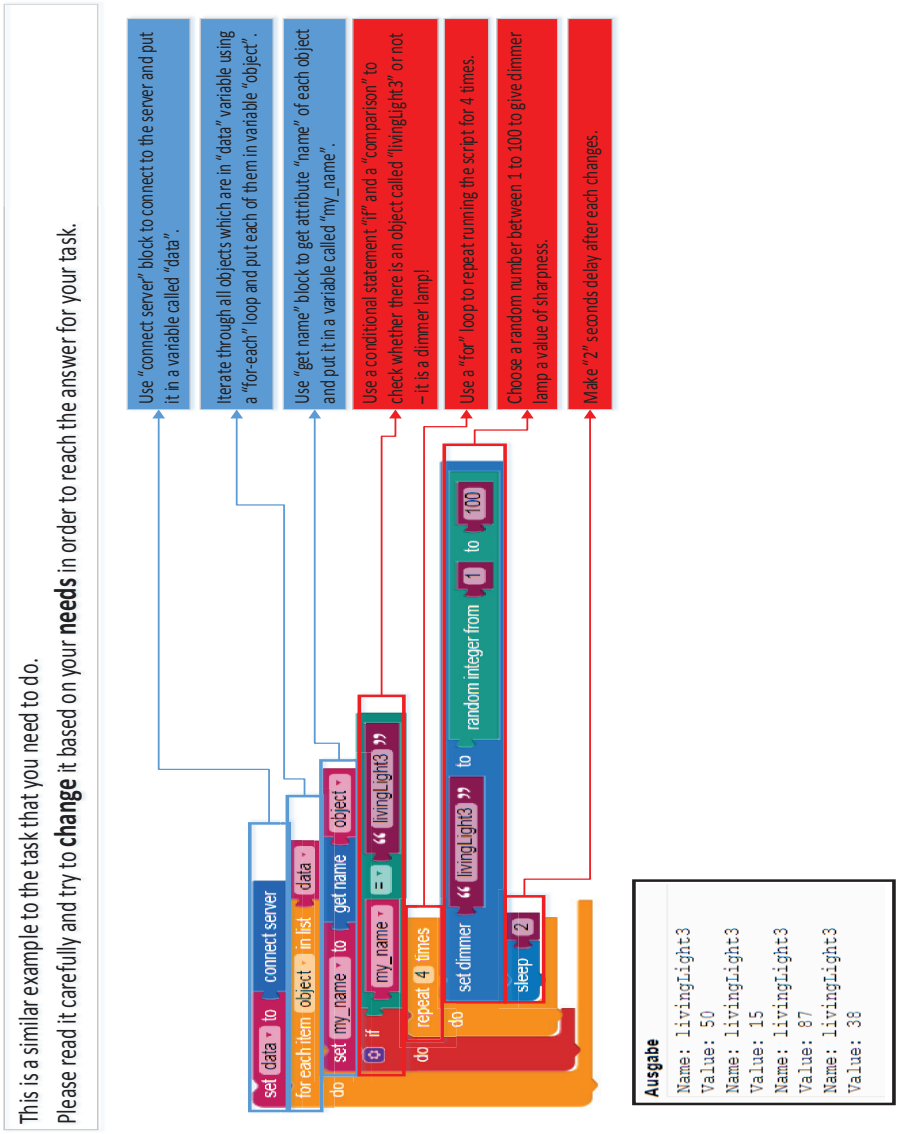
A.3 Worked Example for Programming Task 1



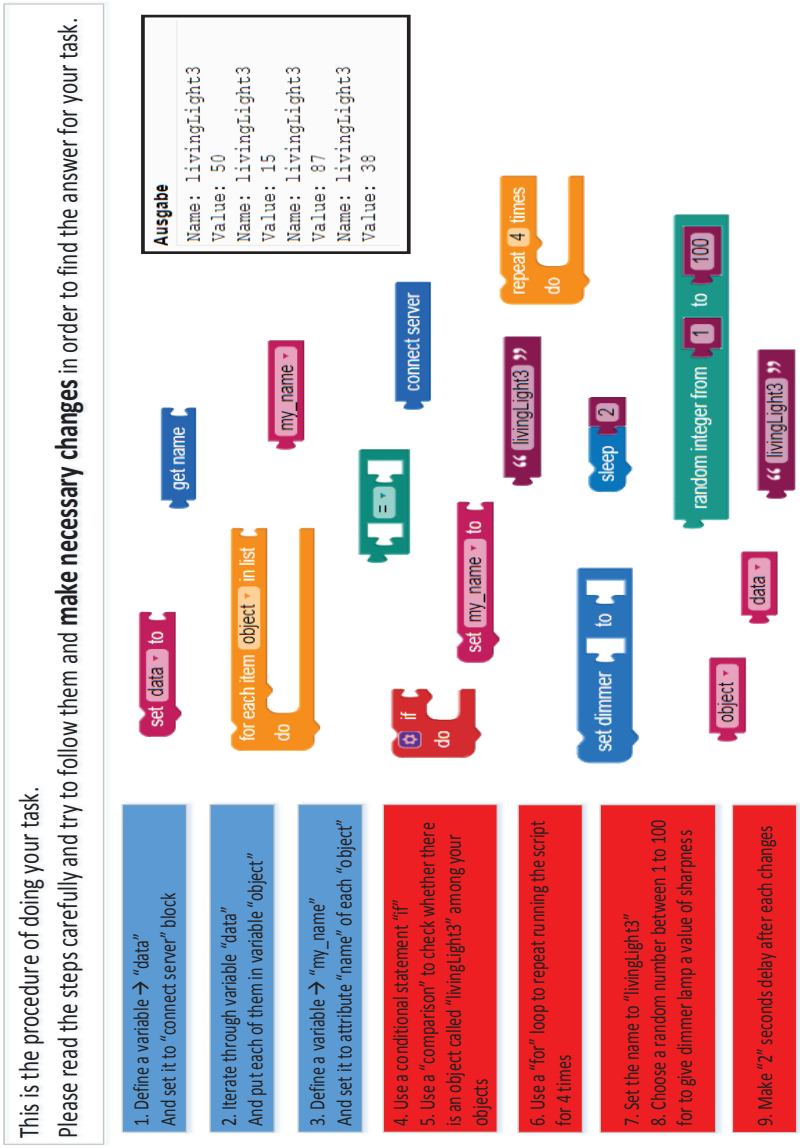
A.4 Instructional Procedure for Programming Task 1



A.5 Worked Example for Programming Task 2



A.6 Instructional Procedure for Programming Task 2



Appendix B

Questionnaires and Programming Questions

B.1 Pre-questionnaire in the beesm-group

Since we want to ask you both before and after the course, we need a code to assign your answers. This is necessary so that the survey can be done anonymously.

What is the first letter of your mother's first name?	☐
What is the second letter of your father's first name?	☐
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	☐

Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

	Yes, very				No, not at all	Don't know
Do you think programming is fun?	☐	☐	☐	☐	☐	☐
Do you think you are good at programming?	☐	☐	☐	☐	☐	☐
Do you think programming is difficult to understand?	☐	☐	☐	☐	☐	☐
Would you like to learn how to program?	☐	☐	☐	☐	☐	☐

Here you can see two examples of block-based programming:

	Yes	No
Have you ever worked with a block-based programming environment?	☐	☐

	Yes, very good				No, not at all
Can you program yet?	☐	☐	☐	☐	☐

B.2 Post-questionnaire in the beesm-group

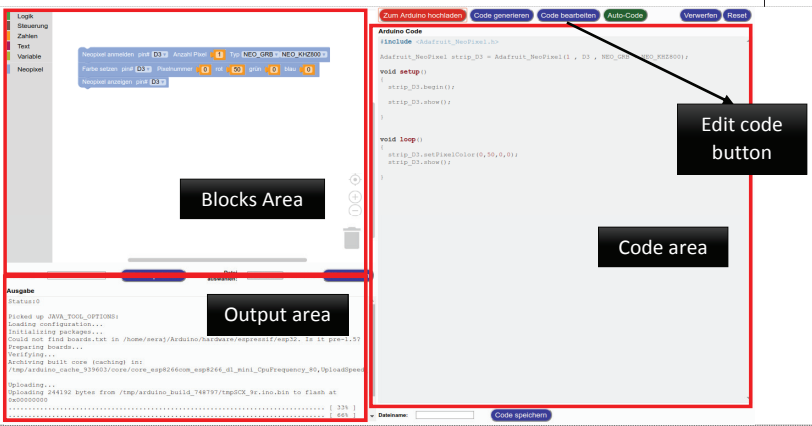
In order to conduct the survey anonymously, we need your code again:

What is the first letter of your mother's first name?	☐
What is the second letter of your father's first name?	☐
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	☐

Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

	Yes, very				No, not at all	Don't know
Do you think programming is fun?	☐	☐	☐	☐	☐	☐
Do you think you are good at programming?	☐	☐	☐	☐	☐	☐
Do you think programming is difficult?	☐	☐	☐	☐	☐	☐
Would you like to learn better how to program?	☐	☐	☐	☐	☐	☐
Do you find it easy to program with blocks?	☐	☐	☐	☐	☐	☐
Do you think it's helpful if you program a real object? E.g. the LED light and the micro-controller.	☐	☐	☐	☐	☐	☐

	Yes				No	Don't know
Did you pay attention to the code that is generated "matching" the blocks?	☐	☐	☐	☐	☐	☐
Do you think that the "edit code" function is helpful to better understand your own program?	☐	☐	☐	☐	☐	☐
Do you think that the "Output" area (console) is helpful to understand your own program?	☐	☐	☐	☐	☐	☐



	With blocks				Direct with code	Dont't Know
Do you prefer programming with block blocks or directly in code?	☐	☐	☐	☐	☐	☐
	Strongly agree				**Strongly disagree**	**Don't know**
I think the programming environment is easy to use.	☐	☐	☐	☐	☐	☐
How old are you? ..... years						

B.3 Pre-questionnaire in the mBlock-group

Since we want to ask you both before and after the course, we need a code to assign your answers. This is necessary so that the survey can be done anonymously.

What is the first letter of your mother's first name?	☐
What is the second letter of your father's first name?	☐
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	☐

Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

	Yes, very				No, not at all	Don't know
Do you think programming is fun?	☐	☐	☐	☐	☐	☐
Do you think you are good at programming?	☐	☐	☐	☐	☐	☐
Do you think programming is difficult to understand?	☐	☐	☐	☐	☐	☐
Would you like to learn how to program?	☐	☐	☐	☐	☐	☐

Here you can see two examples of block-based programming:

	Yes	No
Have you ever worked with a block-based programming environment?	☐	☐

	Yes, very good				No, not at all
Can you program yet?	☐	☐	☐	☐	☐

B.4 Post-questionnaire in the mBlock-group

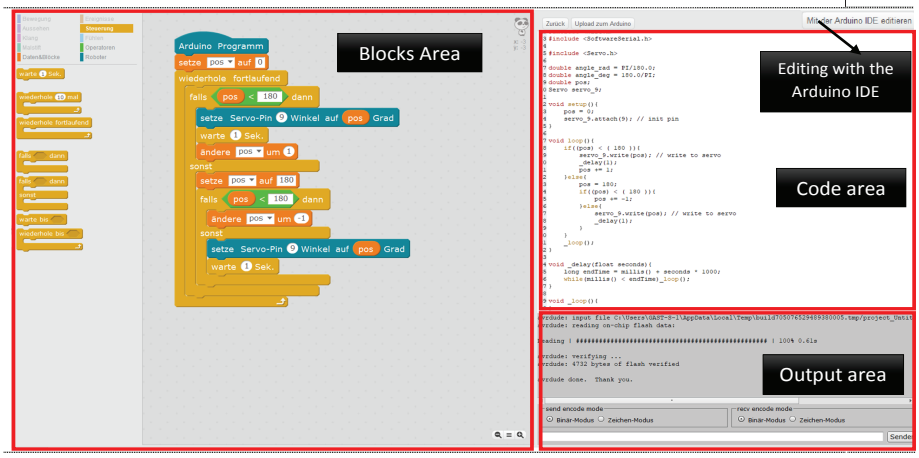
In order to conduct the survey anonymously, we need your code again:

What is the first letter of your mother's first name?	☐
What is the second letter of your father's first name?	☐
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	☐

Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

	Yes, very				No, not at all	Don't know
Do you think programming is fun?	☐	☐	☐	☐	☐	☐
Do you think you are good at programming?	☐	☐	☐	☐	☐	☐
Do you think programming is difficult?	☐	☐	☐	☐	☐	☐
Would you like to learn better how to program?	☐	☐	☐	☐	☐	☐
Do you find it easy to program with blocks?	☐	☐	☐	☐	☐	☐
Do you think it's helpful if you program a real object? E.g. the LED light and the micro-controller.	☐	☐	☐	☐	☐	☐

	Yes				No	Don't know
Did you pay attention to the code that is generated "matching" the blocks?	☐	☐	☐	☐	☐	☐
Do you think the "Edit with the Arduino IDE" feature is helpful to better understand your own program?	☐	☐	☐	☐	☐	☐
Do you think the "Output" area (console) is helpful to understand your own program?	☐	☐	☐	☐	☐	☐



	With blocks				direct with code	Don't know
Do you prefer programming with block blocks or directly in code?	☐	☐	☐	☐	☐	☐
	Strongly agree				**Strongly disagree**	**Don't know**
I think the programming environment is easy to use.	☐	☐	☐	☐	☐	☐
How old are you? ..... years						

B.5 Pre-programming Question in the beesm-group

What is the first letter of your mother's first name?	
What is the second letter of your father's first name?	
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	

Task: On our small computer we have connected a small lamp to pin D3. Now we want to let the light flash red 3 times with a 2 second break in between.

Imagine that you write a program for this. Put the available boxes in the correct logical order. Just write down all numbers of the boxes in your desired order.

Note: Some blocks may appear more than once in your program and some blocks you may not need in your program.

1 Connected to pin [3] with [0] light(s)

2 Connected to pin [D3] with [1] light(s)

3 Connected to pin [D3] with [0] light(s)

4 set light [1] to R [255] G [0] B [0]

5 set light [0] to R [255] G [0] B [0]

6 set light [1] to R [0] G [0] B [255]

7 set lamp [0] to R [0] G [0] B [0]

8 set lamp [1] to R [0] G [0] B [0]

9 set light [1] to R [255] G [255] B [255]

10 delay [2] Seconds

11 delay [1] Second

12 delay [½] Second

13 Display Pin [D3]

14 Repeat the following commands [5] times

15 Repeat the following commands [6] times

16 Repeat the following commands [3] times

Solution:

.....

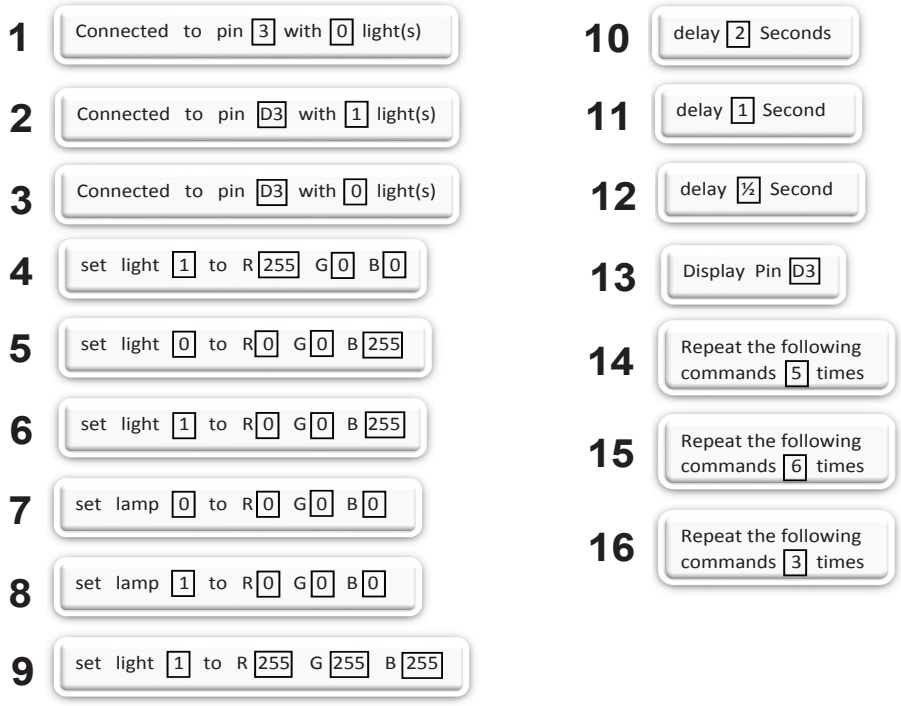
B.6 Post-programming Question in the beesm-group

What is the first letter of your mother's first name?	
What is the second letter of your father's first name?	
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	

Task: On our small computer we have connected a small lamp to pin D3. Now we want to let the light flash blue 6 times with a 1 second break in between.

Imagine that you write a program for this. Put the available boxes in the correct logical order. Just write down all numbers of the boxes in your desired order.

Note: Some blocks may appear more than once in your program and some blocks you may not need in your program.



Solution:

.....

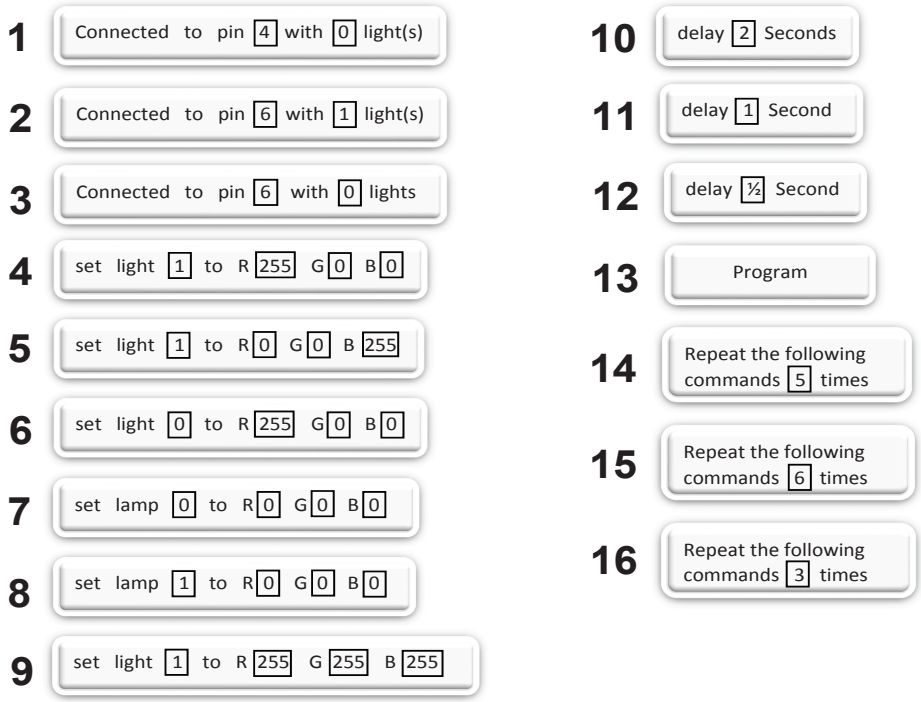
B.7 Pre-programming Question in the mBlock-group

What is the first letter of your mother's first name?	
What is the second letter of your father's first name?	
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	

Task: On our small computer we have connected a small lamp to pin 6. Now we want to let the light flash red 3 times with a 2 second break in between.

Imagine that you write a program for this. Put the available boxes in the correct logical order. Just write down all numbers of the boxes in your desired order.

Note: Some blocks may appear more than once in your program and some blocks you may not need in your program.



Solution:

.....

B.8 Post-programming Question in the mBlock-group

What is the first letter of your mother's first name?	
What is the second letter of your father's first name?	
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	

Task: On our small computer we have connected a small lamp to pin 6. Now we want to let the light flash blue 6 times with a 1 second break in between.

Imagine that you write a program for this. Put the available boxes in the correct logical order. Just write down all numbers of the boxes in your desired order.

Note: Some blocks may appear more than once in your program and some blocks you may not need in your program.

1 Connected to pin [4] with [0] light(s)

2 Connected to pin [6] with [1] light(s)

3 Connected to pin [6] with [0] lights

4 set light [1] to R [255] G [0] B [0]

5 set light [1] to R [0] G [0] B [255]

6 set light [0] to R [0] G [0] B [255]

7 set lamp [0] to R [0] G [0] B [0]

8 set lamp [1] to R [0] G [0] B [0]

9 set light [1] to R [255] G [255] B [255]

10 delay [2] Seconds

11 delay [1] Second

12 delay [½] Second

13 Program

14 Repeat the following commands [5] times

15 Repeat the following commands [6] times

16 Repeat the following commands [3] times

Solution:

.....

Appendix C

Questionnaires and Programming Questions

C.1 Pre Questionnaire (PreQ)

Since we want to ask you both before and after the course, we need a code to assign your answers. This is necessary so that the survey can be done anonymously.

What is the first letter of your mother's first name?	☐
What is the second letter of your father's first name?	☐
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	☐

Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

How do you rate your programming skills?

.....

Do you think you will be successful in this workshop?

.....

I find programming .....

.....

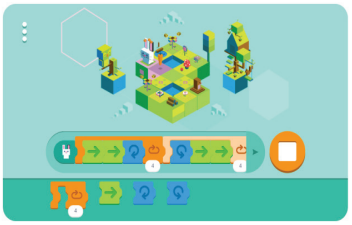
What are you looking forward to in this workshop?

.....

How would you like to learn programming? Why?

.....

Here are two examples of block-based programming:



	Yes	No
Have you ever worked with a block-based programming environment?	☐	☐

C.2 Intermediate Questionnaire (IntermediateQ)

In order to conduct the survey anonymously, we need your code again:

What is the first letter of your mother's first name?	
What is the second letter of your father's first name?	
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	

Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

What do you think about programming with blocks?

.....

What do you think about programming a real smart system? e.g., sensors and actuators.

.....

How do you rate your programming skills?

.....

Do you think you were successful in this workshop?

.....

I find programming .....

.....

How would you like to learn programming? Why?

.....

What did you like about the workshop?

.....

What **did not** you like about the workshop?

.....

Anything else you want to tell us?

.....

C.3 Post Questionnaire (PostQ)

In order to conduct the survey anonymously, we need your code again:

What is the first letter of your mother's first name?	☐
What is the second letter of your father's first name?	☐
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	☐

Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

What do you think about programming with blocks?

.....

what do you think about programming a real smart object? e.g., smart houseplant.

.....

How do you rate your programming skills?

.....

Do you think you were successful in this workshop?

.....

I find programming .....

.....

How would you like to learn programming? Why?

.....

What did you like about the workshop?

.....

What **did not** you like about the workshop?

.....

Anything else you want to tell us?

.....

How old are you? ..... years

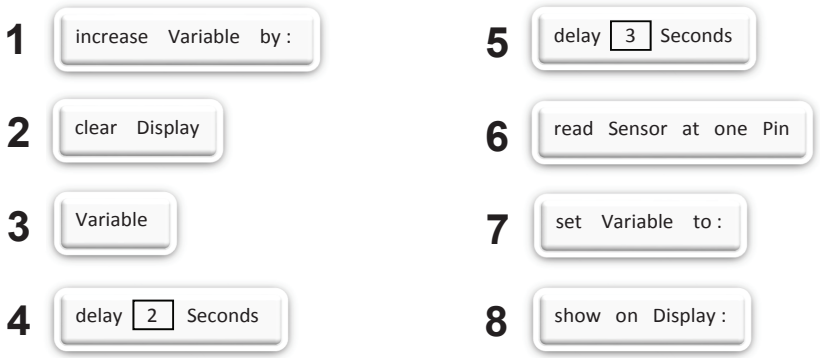
C.4 Pre Programming Question (PrePQ)

What is the first letter of your mother's first name?	
What is the second letter of your father's first name?	
What is your birthday? (e.g. 27, if you were born on 27.03.2000)	

Task: We have connected an LC display to our small computer (Arduino). We also connected a SENSOR (light, distance, temperature, etc.) to a pin and wrote its VALUE into a VARIABLE. Now we want the LC Display to show the VALUE for 2 seconds.

Imagine that you write a program for this. Put the available blocks in the correct logical order. Just note down all numbers of the blocks in your desired order.

Note: Some blocks may appear more than once in your program and some blocks you may not need in your program.



Solution:

.....

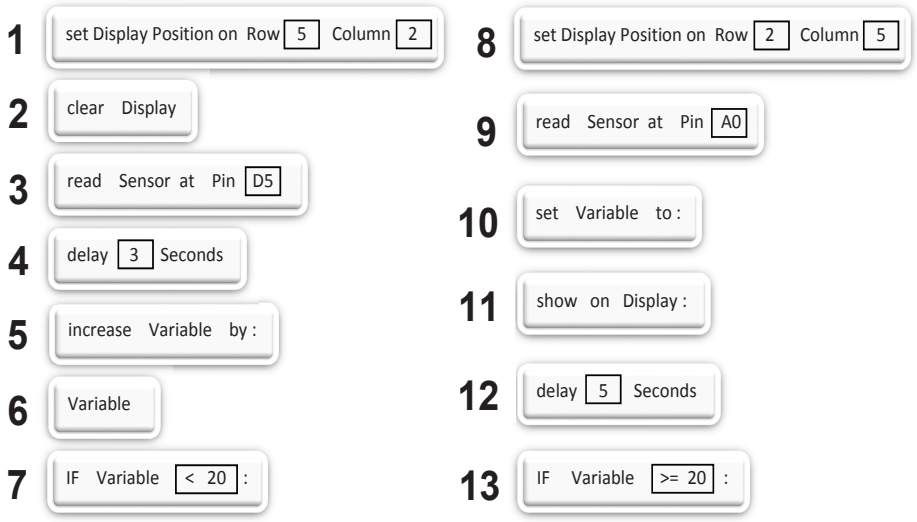
C.5 Intermediate Programming Question (IntermediatePQ)

hat is the first letter of your mother's first name?	
hat is the second letter of your father's first name?	
hat is your birthday? (e.g. 27, if you were born on 27.03.2000)	

Task: We have connected an LC display to our small computer (Arduino). We also connected a SENSOR (light, distance, temperature, etc.) to an analog pin and wrote its VALUE to a VARIABLE. IF the VALUE of the SENSOR is less than 20, THEN we want the LC Display to show the VALUE for 5 seconds in line 2 and column 5.

Imagine that you write a program for this. Put the available blocks in the correct logical order. Just note down all numbers of the blocks in your desired order.

Note: Some blocks may appear more than once in your program and some blocks you may not need in your program.



Solution:

.....

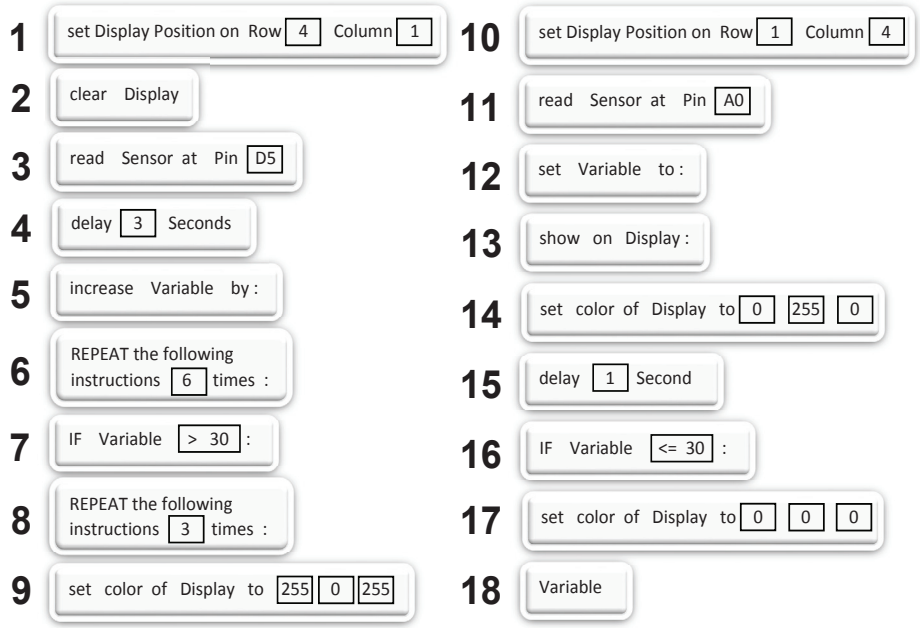
C.6 Post Programming Question (PostPQ)

hat is the first letter of your mother's first name?	
hat is the second letter of your father's first name?	
hat is your birthday? (e.g. 27, if you were born on 27.03.2000)	

Task: We have connected an LC display to our small computer (Arduino). We also connected a SENSOR (light, distance, temperature, etc.) to an analog pin and wrote its VALUE to a VARIABLE. IF the VALUE of the SENSOR is greater than 30, THEN we want the LC Display to show the VALUE for 3 seconds in line 1 and column 4, and additionally the LC Display should blink green 3 times with a pause of 1 second.

Imagine that you write a program for this. Put the available blocks in the correct logical order. Just note down all numbers of the blocks in your desired order.

Note: Some blocks may appear more than once in your program and some blocks you may not need in your program.



Solution:

.....

C.7 Learners' Responses to the Open-ended Questions

P1-P6 with experience | p7-P12 without experience	PreQ	IntermediateQ	PostQ - P9 did not attend
Q1 - How do you rate your programming skills?	P1. I don't program much, so not perfect P2. I would understand, if I do it with same programming environment that I used before P3. I do not know exactly, I think that I am not so bad, but I am not so good P4. Average, I can create simple Scratch programs P5. Good P6. Not perfect, because I forger it --- P7. Not so good, but not so bad P8. I think that they are "average" P9. Not so good P10. Since I have not done anything in this field yet, I don't know. However, I would like to learn it P11. I think that it is fine P12. Not so good	P1. They get better and better P2. I learned a lot in a very short time, but I will have a hard time applying them to a real object like "plant" P3. Good, it works fine P4. I have a little experience, but it is expandable P5. Really great P6. Very good and I understand everything --- P7. Average P8. Good to very good P9. Bad P10. Actually, I understand a lot, but I still need help from time to time P11. I think that I am good P12. It is okay, can be better or worse	P1. They get better and better P2. Fine, but I still would not be able to do it without help P3. Good P4. Minor prior knowledge P5. Really great P6. Very good --- P7. Average P8. Good P9. ---------- P10. From 0 to 10, 7.5 P11. Good P12. Average
Q2 - Do you think you will be successful in this workshop? / were successful in this workshop? In IntermediateQ and PostQ	P1. Depends on what we will do P2. I will definitely know more than before P3. I am not sure, I think so P4. Yes, since I have done the workshop before P5. I think so P6. Yes --- P7. Yes, I think so P8. Yes, I think so P9. Not really, but I will try it P10. I believe that I will learn a lot here and I will have fun P11. Yes P12. A little, but I do not know	P1. Yes P2. Yes, I think that I know a lot about programming now P3. Yes P4. Yes, I think that I am good in solving the programming tasks until now P5. Not sure P6. Yes --- P7. Yes, I think like this P8. Yes, definitely P9. No, not really P10. Yes, I have learned a lot and worked with friends P11. Yes P12. A little bit	P1. Yes, with my new partner P2. Yes, I learned a lot, had a lot of fun, and we have a great "plant" now P3. Yes P4. I could program a bit more P5. Yes P6. Yes --- P7. Yes, I learned a lot P8. Yes, definitely P9. ---------- P10. I think that I have learned a lot and have gained a lot of experience P11. Yes P12. Yes
Q3 - I find programming...	P1. Fun, if you understand it	P1. It is fun and give you time to think and create	P1. To have fun P2. Exciting, I like to work on something for long time until it works

	P2. Very complicated, if you make a mistake at the beginning, you have to do everything again P3. Exciting and important in life P4. Exciting and interesting P5. Great, because it is so much fun P6. Interesting and logical --- P7. Interesting and important for the future P8. Great because you learn it and it's fun P9. Interesting P10. Exciting and possibly something demanding P11. Really interesting P12. Exciting	P2. Easier and more fun than I have expected, especially when you work in a team P3. Interesting P4. Logical and exciting P5. Fun, great and interesting P6. Easy, logical and it feels good --- P7. Exciting and fun to do P8. Great P9. Exhausting, difficult, but sometimes it's cool P10. Very cool, you have to combine many things well and logically, This is fun P11. Cool P12. Exhausting, complicated but also good and interesting	P3. Interesting P4. Interesting and exciting P5. Great, because it is so fun P6. Easy, logical and it feels good --- P7. Interesting P8. Really great P9. ---------- P10. Super cool but a bit demanding P11. Interesting, you can always try something P12. Good, practical, interesting, but complicated and needs efforts
Q4 - What are you looking forward to in this workshop?	P1. We can do what we like P2. Working in a team to achieve a great result P3. ---------- P4. Working with the object (plant) P5. I will have fun while programming P6. We work with sensors again --- P7. Programming P8. I am looking forward to know what programming skills do I have P9. To work with new people P10. Learning how to deal with technical things (e.g. computers, etc.) P11. We will program the object (Plant) P12. I will get to know new and interesting things, especially in this area		
Q5 - How would you like to learn programming? Why?	P1. Yes, because it is fun to create things P2. I do not necessarily want to learn programming, I want to know where programming can be applied to P3. Because it is interesting P4. I like mathematics and general science and I like to learn it P5. Very gladly, because I have and I will have a lot to do in my life with CS	P1. Very much, because it is fun P2. I learned the basics and I am happy with that P3. Because you can program many different things P4. I am a MINT fan and mainly technical stuff P5. Very much, because it is important for my future	P1. Because it is fun P2. I have already learned a little bit P3. Because it is fun P4. I like MINT subject P5. Very much, because it is important for my future P6. It could be useful for my later profession ----------

	P6. I think that it would be useful for my future career --- P7. Very gladly, because it is fun P8. Very much, because it can help me in my professional life P9. I like to learn but I do not think that I need it in the future P10. I think you can have a lot of fun with programming and be creative P11. Very much, because I think that it will be fun P12. I like to do it simply because I want to know how to do it and how something like that works	P6. It could be useful for my later profession --- P7. Very much, because you need it in the future and it is fun P8. I can do it nicely and it is great P9. Not at all P10. From 0 to 10, 7. It's fun, but sometimes it's exhausting P11. Very gladly, because it is fun P12. ----------	P7. Very gladly, because it is fun P8. I learned it and it is great P9. ---------- P10. From 0 to 10, 7. it is very demanding P11. Very gladly, because it is fun P12. ----------
Q6 - have you ever worked with a block-based programming environment?	P1. Yes P2. Yes P3. Yes P4. Yes P5. Yes P6. Yes	P7. No P8. No P9. No P10. No P11. No P12. No	
**********	**IntermediateQ**	**PostQ**	
Q7 - What do you think about programming with blocks?	P1. Good, because they give you a lot of ways to program correctly P2. Much easier than I have expected (if you understood it) P3. It is not so hard P4. Good, It is easier to understand P5. Pretty cool P6. Very easy and cool --- P7. It is fun, but it is a little bit complicated P8. I find it exciting and great P9. Exhausting but it is OKAY P10. It is really fun and you can try a lot of things P11. Awesome P12. Exhausting	P1. Cool, because they give you different ways to program correctly P2. Logical and sometimes you need to explore by yourself (self-exploratory) P3. Simple P4. It is great P5. Great P6. Easy and creative --- P7. I have a lot of fun with it and you learn a lot through it P8. I think that it is cool to try P9. ---------- P10. It is a lot of fun P11. I enjoyed it P12. Good and interesting but also complicated	
Q8 - What do you think about programming a real system? (Sensors and Sctuators) in IntermediateQ / a real smart object? (smart houseplants)in PostQ	P1. -------- P2. I think it's good to work with sensors and "ordinary" actuators, but playing with LEDs and LCDs is even more fun	P1. Interesting P2. Cool, especially "plant" which is funny P3. Cool	

	P3. ----------	P4. Interesting and exciting
	P4. Quite exciting	P5. Pretty cool
	P5. Great	P6. Really easy and fun
	P6. It was a lot of fun	----------
	----------	P7. exciting and you can try something new
	P7. I think that it is good	P8. Awesome
	P8. Very cool	P9. ----------
	P9. Cool	P10. I think it's cool that we do something great and exciting
	P10. Very exciting and cool	P11. Really great
	P11. Awesome	P12. Super good
	P12. Interesting and good	
Q4 - What did you like about the workshop?	P1. There are good helps given	P1. Everything
	P2. I have learned so much in a short time	P2. ----------
	P3. You can program different things in a team	P3. Programming and decorating the real object (plant)
	P4. We make everything step by step	P4. It was very creative
	P5. Everything so far	P5. Programming the object (plant)
	P6. Everything	P6. We can program the object (plant) in a way which is our desire
	----------	----------
	P7. Programming	P7. Programming
	P8. BEESM	P8. Working with the blocks
	P9. The breaks	P9. ----------
	P10. We had fun and learned something	P10. Cooperation with others and learning new things
	P11. All	P11. All
	P12. Almost everything so far	P12. Actually everything
Q4 - What did not you like about the workshop?	P1. Everything is partnership work	P1. My partner has done nothing and left me alone
	P2. We have not worked with the real object (plant)	P2. ----------
	P3. ----------	P3. We often fill the questionnaire
	P4. Nothing	P4. Nothing
	P5. Nothing so far	P5. Nothing
	P6. Nothing	P6. Nothing
	----------	----------
	P7. Many questionnaires	P7. Too many questionnaires
	P8. Nothing	P8. Nothing
	P9. It's too warm but it goes until 1:30	P9. ----------
	P10. Sometimes expectations come that can be better	P10. ----------
	P11. Nothing	P11. Nothing
	P12. It is too hot and it will be 1.30	P12. Nothing

Appendix D

Questionnaires and Programming Questions

D.1 Pre Questionnaire (PreQ)

Since we want to ask you both before and after the course, we need a code to assign your answers. This is necessary so that the survey can be done anonymously.

What is the first letter of your mother's first name?	☐
What is the second letter of your father's first name?	☐
What is your birth-day? (e.g. 27, if you were born on 27.03.2000)	☐

Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

	yes, very much				**no, not at all**
1. Do you think you're good at programming?	☐	☐	☐	☐	☐
2. Do you think you will be successful in this workshop?	☐	☐	☐	☐	☐
3. Do you think programming is fun?	☐	☐	☐	☐	☐
4. Do you like programming?	☐	☐	☐	☐	☐
5. Are you excited about this workshop?	☐	☐	☐	☐	☐
6. Do you think programming is difficult?	☐	☐	☐	☐	☐
7. Are you interested in programming?	☐	☐	☐	☐	☐
8. Would you like to learn how to program?	☐	☐	☐	☐	☐

Here are two examples of **block-based programming** and a **micro-controller**:

	Yes	**No**
9. Have you ever worked with a **micro-controller** (e.g. Arduino)?	☐	☐
10. Have you ever worked with a **block-based programming** environment?	☐	☐

11. I find programming .....

.....

12. What do you think of this workshop? .....

.....

D.2 Intermediate Questionnaire (IntermediateQ)

In order to conduct the survey anonymously, we need your code again:

What is the first letter of your mother's first name?	
What is the second letter of your father's first name?	
What is your birth-day? (e.g. 27, if you were born on 27.03.2000)	

Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

	yes, very much				**no, not at all**
1. Do you think it is useful to program a real object? (e.g., the LEDs light up)	☐	☐	☐	☐	☐
2. Do you think you are good at programming?	☐	☐	☐	☐	☐
3. Do you think you were successful in this workshop?	☐	☐	☐	☐	☐
4. Do you think programming is fun?	☐	☐	☐	☐	☐
5. Do you like programming?	☐	☐	☐	☐	☐
6. Are you excited about this workshop?	☐	☐	☐	☐	☐
7. Do you think programming is difficult?	☐	☐	☐	☐	☐
8. Are you interested in programming?	☐	☐	☐	☐	☐
9. Would you like to learn more about programming?	☐	☐	☐	☐	☐
	strongly agree				**strongly disagree**
10. I like to use [the programming environment].	☐	☐	☐	☐	☐
11. I think [the programming environment] is easy to use.	☐	☐	☐	☐	☐
12. It was easy to learn how to use [the programming environment].	☐	☐	☐	☐	☐
13. I can use [the programming environment] without written instructions.	☐	☐	☐	☐	☐
14. I am able to complete tasks and exercises quickly with [the programming environment].	☐	☐	☐	☐	☐
15. [The programming environment] is useful.	☐	☐	☐	☐	☐
16. [The programming environment] is great.	☐	☐	☐	☐	☐
17. Overall, I am satisfied with [the programming environment].	☐	☐	☐	☐	☐

18. What do you **like** about [the programming environment]? .....

.....

19. What do you **dislike** about [the programming environment]? .....

.....

20. What did you particularly **like** about the first workshop day? .....

.....

21. What did you particularly **dislike** about the first workshop day? .....

.....

D.3 Post Questionnaire (PostQ)

In order to conduct the survey anonymously, we need your code again:

What is the first letter of your mother's first name?		
What is the second letter of your father's first name?		
What is your birth-day? (e.g. 27, if you were born on 27.03.2000)		

Please answer the following questions. There are no right or wrong answers. Your personal opinion is important.

	yes, very much				**no, not at all**
1. Do you think it's useful if you program a real smart object? (e.g., perform actions based on sensor information)	☐	☐	☐	☐	☐
2. Do you think you are good at programming?	☐	☐	☐	☐	☐
3. Do you think you were successful in this workshop?	☐	☐	☐	☐	☐
4. Do you think programming is fun?	☐	☐	☐	☐	☐
5. Do you like programming?	☐	☐	☐	☐	☐
6. Are you excited about this workshop?	☐	☐	☐	☐	☐
7. Do you think programming is difficult?	☐	☐	☐	☐	☐
8. Are you interested in programming?	☐	☐	☐	☐	☐
9. Would you like to learn more about programming?	☐	☐	☐	☐	☐
	strongly agree				**strongly disagree**
10. I like to use [the programming environment].	☐	☐	☐	☐	☐
11. I think [the programming environment] is easy to use.	☐	☐	☐	☐	☐
12. It was easy to learn how to use [the programming environment].	☐	☐	☐	☐	☐
13. I can use [the programming environment] without written instructions.	☐	☐	☐	☐	☐
14. I am able to complete tasks and exercises quickly with [the programming environment].	☐	☐	☐	☐	☐
15. [The programming environment] is useful.	☐	☐	☐	☐	☐
16. [The programming environment] is great.	☐	☐	☐	☐	☐
17. Overall, I am satisfied with [the programming environment].	☐	☐	☐	☐	☐

18. What do you **like** about [the programming environment]? .....
.....

19. What do you **dislike** about [the programming environment]? .....
.....

20. What did you particularly **like** about the workshop? .....
.....

21. What did you particularly **dislike** about the workshop? .....
.....

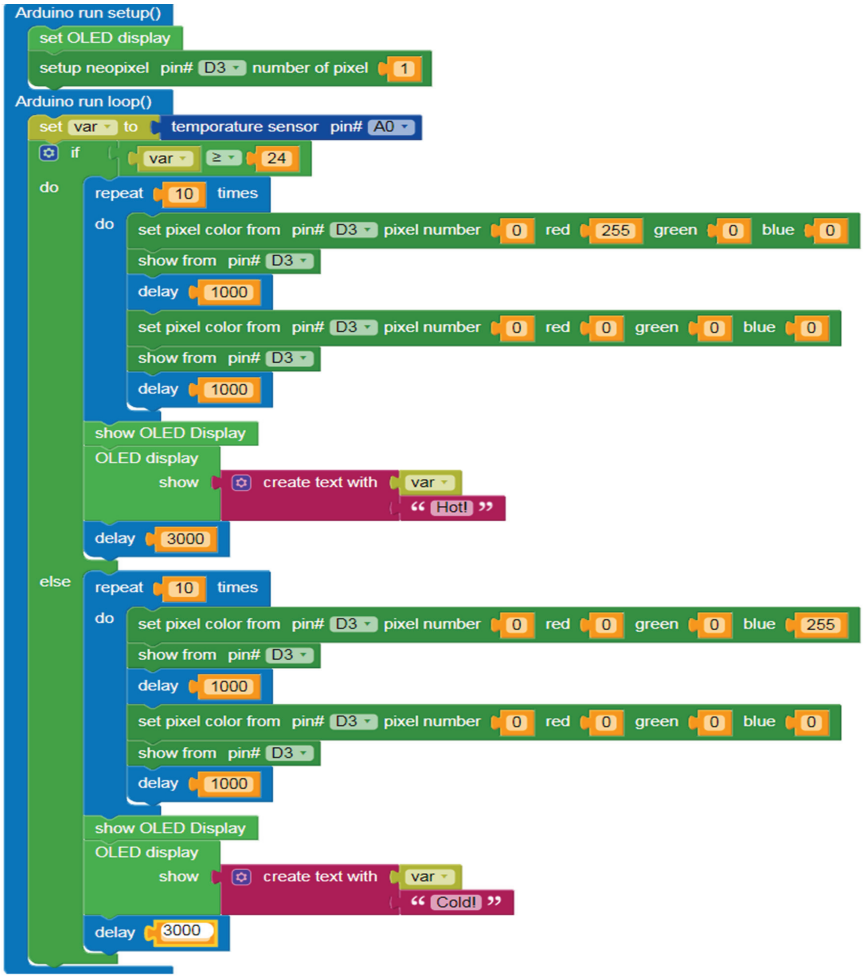
Are you a girl or a boy? ☐ Girl ☐ Boy

How old are you? ..... years old

D.4 Pre Programming Question (PrePQ)

What is the first letter of your mother's first name?	
What is the second letter of your father's first name?	
What is your birth-day? (e.g. 27, if you were born on 27.03.2000)	

***Programming Task**: please answer the following questions about this program.*



1. What is the value of "var", when this program is executed?
   .....

2. If "var" is 27, what is shown on the display?
   .....

3. If "var" is 20, what is shown on the display?
   .....

4. How often does the "LED" flash, **each time** the loop is executed?
   .....

5. The "LED" flashes for ..... seconds, **each time** the loop is executed.

6. What color is the "LED", when the program is executed?
   .....

7. The value of "var" is shown on the display for ..... seconds, **each time** the loop is executed.

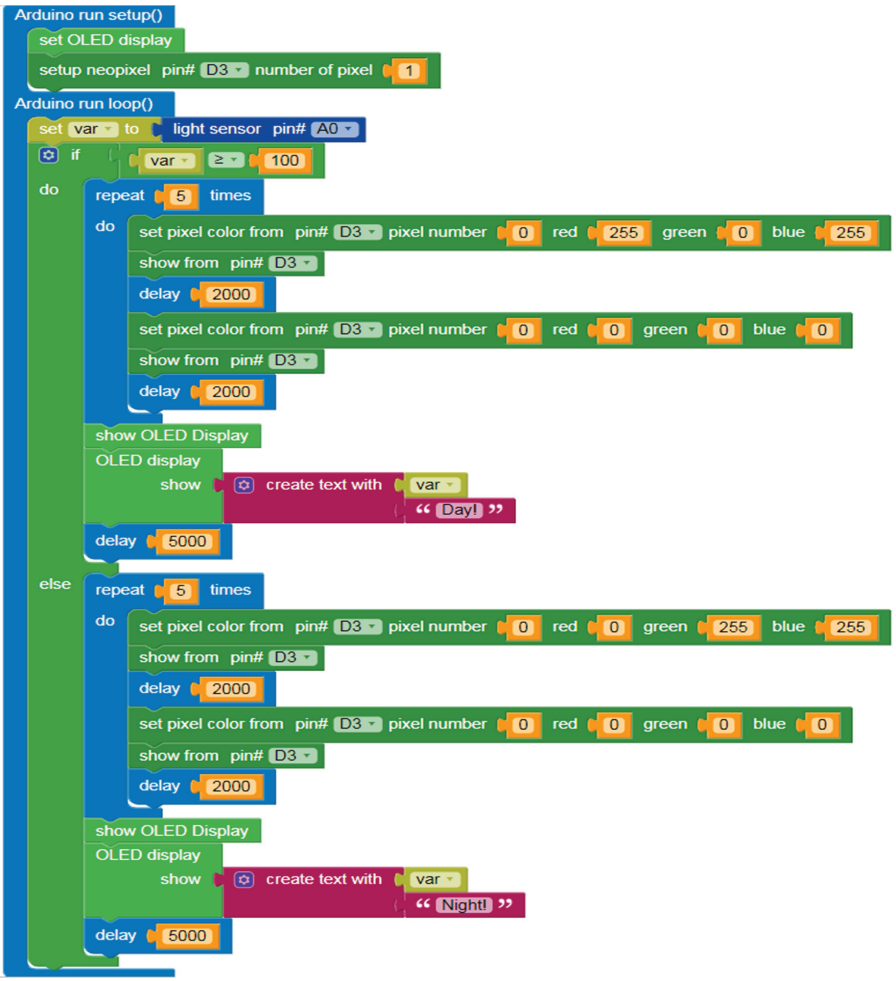
8. How often is the comparison "var >= 24" tested during the execution of this program? .....

9. Under what condition is the word "Cold!" shown on the display, when this program is executed? .....

D.5 Post Programming Question (PostPQ)

What is the first letter of your mother's first name?	
What is the second letter of your father's first name?	
What is your birth-day? (e.g. 27, if you were born on 27.03.2000)	

***Programming Task**: please answer the following questions about this program.*



1. What is the value of "var", when this program is executed?
   .....

2. If "var" is 130, what is shown on the display?
   .....

3. If "var" is 80, what is shown on the display?
   .....

4. How often does the "LED" flash, **each time** the loop is executed?
   .....

5. The "LED" flashes for ..... seconds, **each time** the loop is executed.

6. What color is the "LED", when the program is executed?
   .....

7. The value of "var" is shown on the display for ..... seconds, **each time** the loop is executed.

8. How often is the comparison "var >= 100" tested during the execution of this program? .....

9. Under what condition is the word "Night!" shown on the display, when this program is executed? .....

D.6 Learners' Responses to the Open-ended Questions

P1-P6 with experience (girls) | p7-P28 without experience (boys)		
PreQ – I find Programming ...	P1. Super exciting, because you have to try a lot and think logically P2. Good and interesting because you can let your creativity run freely P3. Interesting P4. An interesting thing, because you're creative P5. Very difficult and very interesting because you can program so many different things, and you can be creative P6. Exciting and I would like to know more about it because it is very creative P7. Very cool P8. Very Interesting P9. Very Interesting P10. Very Interesting P11. Great P12. --- P13. Very interesting and wants to try it myself P14. Interesting	P15. Great and exciting P16. Exciting P17. Interesting and cool P18. Great P9. Interesting P20. OK! P21. Good! P22. Exciting P23. Interesting, because it is something very new P24. Very interesting and easy P25. Sometimes it is exciting but generally I find it boring P26. Interesting P27. Cool P28. Very good
PreQ - What do you think of this workshop?	P1. I think it's good that this is being offered, and it's going to be great P2. I'm sure it'll be a lot of fun and I think it's good that something like this is offered P3. I'm on it P4. I'm very excited about what we're doing P5. I think it'll be interesting P6. I think it's good that such a thing is offered, because you can learn more about it P7. It sounds interesting to me P8. I think it's good P9. I think it's really good P10. I think it's good P11. I had a lot of this workshop P12. --- P13. I do not know that what we will do but I will enjoy it most likely P14. I think the idea is good because computer science is becoming increasingly important	P15. --- P16. --- P17. I hope that you get an insight into the smart living here P18. --- P9. I do not know P20. It is very cool P21. --- P22. I do not know P23. I hope that will be fine, because I feel this before it starts P24. --- P25. Until now it is OK, but I cannot give my opinion, cause the workshop is not started yet P26. --- P27. I do not know but I am happy to be here P28. I think it is a good workshop

IntermediateQ - What do you like about [the programming environment]	P1. It is arranged and construct properly P2. It is well arranged and very clearly structured P3. --- P4. Easy arrangement P5. It is clearer than other programs P6. I think it's very practical, because it allows beginners to learn how to program P7. It is easy to use P8. It's clear P9. It's great and it's easy to understand P10. It is easy to use and understand P11. Everything P12. --- P13. blocks P14. It is not so complex	P15. The possibility that you can try by yourself P16. The simple clear blocks P17. It is easy to use P18. That it is logical P19. --- P20. --- P21. Clear and fun P22. Everything P23. It works with blocks P24. It's easy P25. Not much but the programming of display and led are pretty cool P26. It is not so complicated P27. Everything is well arranged P28. That is it visual
IntermediateQ - What do you dislike about [the programming environment]	P1. Zoom In / Out P2. That sometimes the words are not quite clear P3. --- P4. Sometimes a little unclear P5. Some things are also very complicated P6. Partly the programs did not work P7. I do not know P8. Nothing P9. Nothing P10. Nothing P11. Nothing P12. --- P13. Something is not understandable P14. ---	P15. "delay" is not easy to understand P16. --- P17. The errors are not very understandable me P18. I found no "if then" block P19. --- P20. Nothing P21. Some very difficult P22. Nothing P23. That you don't have the ability to enter codes P24. Too simple P25. All of the blocks have complicated labels P26. --- P27. Zoom in / out P28. That it is a little bit complicated
IntermediateQ - What did you particularly like about the first workshop day?	P1. Experimenting with "BEESM" yourself - super to operate P2. Actually everything P3. That we can watch P4. Creative self-test P5. You've learned more things P6. That we could program without much instruction P7. The task with the lights P8. Programming P9. Programming P10. Programming P11. Program the lights	P15. --- P16. LEDs P17. Display P18. Almost everything P19. --- P20. That the tasks were not so difficult and not too easy P21. Programming with blocks P22. The explanation P23. That it was clear P24. I could run and test new program P25. The display

	P12. --- P13. Display P14. Tasks with the light		P26. Programming P27. It made a lot of fun P28. The programming of the LEDs
IntermediateQ - What did you particularly dislike about the first workshop day?	P1. Became lengthy with time (better to divide tasks) P2. It is sometimes boring as I have done it one time before P3. A lot of repetition P4. Almost the same thing I've already done P5. I have already done working with LEDs P6. We've done it before in school P7. That we only discussed the solutions verbally P8. Nothing P9. Nothing P10. Short Breaks P11. I want to program with codes more P12. --- P13. That we are not allowed to take anything with us P14. ---		P15. --- P16. We have tasks to do mostly and we could not experiment by our-self P17. --- P18. That we've been allowed to work so short and that so much has been explained P19. --- P20. Nothing P21. That we are not allowed to take anything with us P22. --- P23. That you can't do so many things you choose to do by yourself P24. I would have liked to do more experiments P25. Long introduction P26. --- P27. I could not get all the tasks finished P28. Nothing
PostQ - What do you like about [the programming environment]	P1. Blocks are easy to understand and it is easy to understand P2. That everything is clear and easy to find P3. It is clearly arranged and you can understand it well P4. That it is clear and easy to understand P5. It is clearer than other programs P6. It is easier to learn the programming steps P7. Clarity - well arranged P8. It's easy to understand P9. It's great and it's easy to use P10. Everything P11. That it is easy for me P12. Programming P13. That you can do a lot P14. It is easy for people to start programming		P15. --- P16. Actually everything P17. Easy to use P18. Almost everything P19. --- P20. That you can easily program P21. Everything P22. It is easy P23. --- P24. It's easy P25. Not really P26. It is very easy to understand and use P27. Actually everything P28. That it is easy
PostQ - What do you dislike about [the programming environment]	P1. Zoom IN/Out P2. Some labels are not clear (their meaning) P3. --- P4. You have to work a lot with it to understand it P5. There are many complicated things P6. Sometimes uploading doesn't work		P15. "delay" is an unusual term P16. --- P17. Zoom IN/Out P18. --- P19. --- P20. Nothing

	P7. I do not know	P21. Nothing
	P8. Nothing	P22. ---
	P9. I actually like everything	P23. ---
	P10. I do not know	P24. ---
	P11. BEESM is not with code	P25. Not really
	P12. Nothing	P26. ---
	P13. Nothing	P27. Sometimes it is difficult to find blocks
	P14. ---	P28. There is a lot to pay attention to
PostQ - What did you particularly like about the workshop?	P1. No tight schedules and the BAALL	P15. Programming
	P2. Programming	P16. Programming
	P3. When we were allowed to do something by ourselves	P17. To build the lamp
	P4. Creative work (independent)	P18. That we were allowed to work freely
	P5. To handle something about intelligent systems	P19. Experience freely
	P6. That we could relatively program	P20. That with the display and BAALL
	P7. The task with the lights	P21. Programming and decorating
	P8. The one with the lights	P22. The explanation
	P9. Programming	P23. ---
	P10. Programming	P24. Build the lamp
	P11. Everything we have programmed and done alone	P25. The lamp
	P12. ---	P26. Programming and BAALL
	P13. BEESM	P27. I could get to know BEESM
	P14. BAALL tour	P28. Everything
PostQ - What did you particularly dislike about the workshop?	P1. There should be more tasks. it got lengthy	P15. ---
	P2. I did most things before	P16. ---
	P3. Was a lot of repetition	P17. ---
	P4. Often no variety (boring)	P18. The long explanations
	P5. Some things were not so useful yet	P19. Long explanations
	P6. ---	P20. Nothing
	P7. Only discussed the solutions verbally	P21. Nothing
	P8. Short breaks	P22. ---
	P9. Short breaks	P23. ---
	P10. Questions and short breaks	P24. ---
	P11. ---	P25. Programming
	P12. ---	P26. ---
	P13. That we cannot take anything with us	P27. I did not find that programming the lamp is exciting
	P14. ---	P28. Nothing