

*TangibleMike:
Tangible Programming
For High School
Students*

Bachelor's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

*by
Damian Schablowsky*

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Martina Ziefle

Registration date: 22.04.2016
Submission date: 22.08.2016

Eidesstattliche Versicherung

Name, Vorname

Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/
Masterarbeit* mit dem Titel

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum

Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum

Unterschrift

Contents

Abstract	ix
Überblick	xi
Acknowledgements	xiii
Conventions	xv
1 Introduction	1
2 Related work	5
2.1 Learn Computer Programming	5
2.2 Teaching Systems	7
2.3 Tangibles	10
3 Design	15
3.1 Concept	16
3.2 First Iteration	17
3.3 Second Iteration	20

4	Application	25
4.1	Hardware	25
4.1.1	Operating System	26
4.1.2	Tangibles	26
4.1.3	Tabletop	30
4.2	Software	30
4.2.1	Design	31
4.2.2	Architecture	33
5	Evaluation	41
5.1	Study Design	41
5.2	Results	42
5.3	Discussion	44
6	Summary and Future Work	45
6.1	Summary	45
6.2	Future work	46
A	Appendix for the Application Study	49
	Bibliography	57
	Index	61

List of Figures

2.1	Lightbot	8
2.2	Scratch	9
2.3	Osmo	10
2.4	SLAP widget	11
2.5	PUC tangible	12
2.6	PERC tangible	13
3.1	DIA cycle	16
3.2	Drawing for the first paper prototype	18
3.3	First prototype	19
3.4	First loop attempt	20
3.5	Second prototype.	21
3.6	Participant of second evaluation	22
3.7	Different solutions for a single task	23
4.1	Current prototype of the real tangible	28
4.2	Currently used tabletop	31

4.3	UI of current version	32
4.4	All game scenes	34
4.5	First level of the game	35
5.1	Results of the questionnaire's first segment .	43
5.2	Results of the questionnaire's second segment	43
A.1	Level 1	49
A.2	Level 2	50
A.3	Level 3	50
A.4	Level 4	50
A.5	Level 5	51
A.6	Level 6	51
A.7	Level 7	51
A.8	Level 8	52
A.9	Level 9	52
A.10	Level 10	52
A.11	Questionnaire – first page	53
A.12	Questionnaire – second page	54
A.13	Questionnaire – third page	55
A.14	Questionnaire – fourth page	56

Abstract

New technologies are constantly developed, existing companies in the information technology (IT) sector are expanding or new ones are established. As a result, additional to the existing shortage of specialists, the demand for newly graduated IT specialists is constantly growing. To solve this problem, it is important to motivate young students to consider a career in the IT sector.

Though computer programming is a major section of the IT, learning it can be challenging for beginners. We present *TangibleMike*, a software application to teach beginners the fundamentals of computer programming by using tangibles on multi-touch screens instead of programming code.

To create the application, we performed three iterations of the DIA cycle. In the first two iterations we created paper prototypes and performed two studies to evaluate potential ambiguities in the user interface. In the third iteration we implemented the actual application and performed a third study. Initially we planned to compare a tangible version with a non-tangible version in our last study to determine if tangibles have a positive effect on the learning process. Unfortunately, the used tangibles had several technical issues and, therefore, we couldn't use them in a study. Instead, we further concentrated on improving the user interface and the gameplay.

The performed study showed that the user interface was generally intelligible, however, the loop and more importantly the turn-tangible need further refinements in the following iterations.

Überblick

Im Bereich der Informationstechnologie (IT) werden ständig neue Technologien entwickelt, neue Unternehmen gegründet oder bestehende expandiert. Dies hat zur Folge, dass zum bestehenden Mangel an Fachkräften der Bedarf an neuen ständig ansteigt. Um dieses Problem zu lösen, müssen Schüler schon früh dazu motiviert zu werden, eine Karriere in der IT in Betracht zu ziehen.

Die Programmierung gehört zu den Hauptbereichen der IT, jedoch stellt ihr Erlernen eine Hürde für Programmieranfänger dar. In dieser Bachelor Arbeit präsentieren wir die Software *TangibleMike*, welche Programmieranfängern mit Hilfe von greifbaren Objekten, auch Tangibles genannt, und Multi-Touch-Bildschirmen die grundlegenden Konzepte der Programmierung beibringt.

Wir haben zunächst zwei Papierprototypen erstellt und deren Unklarheiten und Schwächen in zwei Studien evaluiert. Anschließend haben wir die Anwendung implementiert. In unserer dritten Studie sollte mit einem Vergleich evaluiert werden, ob Tangibles einen positiven Einfluss auf den Lernprozess haben, aber diese Studie konnte jedoch aufgrund von technischen Problemen der Tangibles nicht durchgeführt werden. Stattdessen lag der Fokus darauf, die Benutzeroberfläche sowie den Spielablauf zu evaluieren und zu verbessern.

Die Studie hat gezeigt, dass die einzelnen Elemente der Anwendung im Allgemeinen gut verständlich sind, die Loop- und Turn-Tangibles im nächsten Entwicklungsschritt aber verbessert werden müssen.

Acknowledgements

First and foremost, I would like to thank all testers and friends who participated in the several user studies conducted for this thesis. I had a good time observing and talking to you while playing my application. The collected feedback presents an important part of this thesis.

Secondly, I would like to thank Prof. Dr. Jan Borchers and Prof. Dr. Martina Ziefle for giving me the opportunity to write this thesis. You always had an open door and helped wherever you could. Thanks for this.

I would also like to thank Christian Cherek, M.Sc., my supervisor. You offered valuable support whenever I asked for it and it was a pleasure to work together with you.

Special thanks to Dr. Dipl.-Inform. Philipp Brauner for supporting me during the whole thesis. I really appreciated all of your helpful advises, all the help you offered me and the tips you gave when I got stuck.

And big thanks to Agnes, Danielle, Thomas, my whole family and friends for your unconditional and continuous help throughout the whole thesis. Thanks for cheering me up all the time and thank you for keeping your door open.

Thank you all!
Damian

Conventions

Throughout this thesis we use the following conventions.

Names of software or widgets are written in italic text

Source code and implementation symbols are written in typewriter-style text.

`myClass`

The whole thesis is written in American English.

Chapter 1

Introduction

Computers can take several forms like smartphones, tablets, and laptops. People are confronted with them every day and the computer market is growing constantly: new technologies are invented and new companies are established continuously. As a result, the demand for information technology (IT) specialists rises. Unfortunately, there is a huge shortage of IT specialists, especially in Germany. The Bitkom e.V. [2014] reported that about 41,000 IT specialists are missing in Germany. The Bundesagentur für Arbeit [2016] and the Statistisches Bundesamt [2016] also confirmed the shortage of IT specialists. Even though new specialists graduate from universities, their number barely compensate the number of retired specialists. Since the job market is growing constantly, the shortage keeps on increasing. The Verein Deutscher Ingenieure [2012] even assumes that this shortage might compromise Germany's competitiveness regarding the IT sector. The main problem is that too few students plan a career as an IT specialist. It is important to encourage and motivate the students to deal with this area of expertise. Additionally, it is necessary to motivate female students in order to decrease the gender gap [Beede et al., 2011].

Shortage of IT specialists

A wide area of the IT sector deals with software creation or maintaining existing ones. Kelleher and Pausch [2005] stated that learning to program can be a challenge due to the fact that it consists of several parts. The first part con-

A programming language is composed of different parts

tains general concepts like procedures, loops, function calls, recursion, and so forth. The second part consists of the syntax and the semantics. The programming language's syntax can be seen as the language's grammar, answering the question: how do I construct a valid sentence? The semantics, on the other hand, describes the meaning of a sentence and answers the question: is this sentence's meaning valid? The syntax and semantic varies from language to language, although some of them are similar to each other. The next part is adjusting to the programming environment which normally consists of a source code editor and a compiler. The compiler transforms the written code into binary code that represents computer processor instructions. Learning all those parts at the same time can be challenging for a beginner and, therefore, intimidating.

Simple programming environment

To address this problem, we present *TangibleMike*, an application to teach novices the fundamentals of computer programming. As stated by Kelleher and Pausch [2005], programming commands can be seen as blocks that are aligned in a specific order to create an application. Using this metaphor, we created a simple programming environment, where the user creates programs by aligning blocks instead of using programming code. This way, the novice can concentrate on the basics and afterwards move to a general-purpose language.

Two versions of *TangibleMike*

In our application the novice navigates a figure and solves logical tasks. We created two versions of our application. In the first version the blocks are represented by graphical objects on a screen. The user interacts with finger gestures on a multi-touch screen. In the second version the blocks are represented by tangibles (graspable widgets or objects) that can be placed on a multi-touch screen. Using tangibles in education can benefit the learning process [Brauner et al., 2010]. Therefore, in this thesis we want to evaluate if tangibles can motivate High School students to concern themselves more with the IT sector or even to plan a career as an IT specialist. We created two versions in order to evaluate the tangible's benefits over graphical objects.

A novel teaching system

There is already a number of applications that use graphical objects instead of programming code or use tangibles for

educational purpose that will be presented in chapter 2. In *TangibleMike*, we are using tangibles on multi-touch screens to create a novel teaching systems. However, creating this kind of systems comes with three major challenges:

1. Create distinguishable and persistently trackable tangibles.
2. Setup the communication between the application and the tangible.
3. Design a simplified programming environment.

We chose the tangibles created by Voelker et al. [2015] and the MultiTouchKit framework by Linden [2014] in order to meet the first two challenges. Therefore, we could concentrate on the design of a simplified programming environment that harnesses the tangible's potential to motivate High School students.

Focus on the third challenge

This thesis will describe the process of creating the application *TangibleMike*. Initially, in chapter 2 we will present existing teaching system and different tangible variants. Chapter 3 will explain the design process and the first two performed evaluations. The resulting application and used hardware will be explained in detail in chapter 4. Chapter 5 will give a detailed description of our third evaluation and discuss the results, followed by future work in chapter 6.

Chapter 2

Related work

Our application *TangibleMike* interleaves with several other research fields. In this chapter, we will present the big three research fields. The first one is learning computer programming and investigating the problems novices have while learning computer programming. The second research field treats approaches to simplify the learning process of programming by providing special teaching systems. The third research field treats tangibles and we will present the different variants of them.

Three different
research fields

2.1 Learn Computer Programming

Novices who try to learn computer programming, are confronted with several challenges, as described by Kelleher and Pausch [2005]. First, they have to create structured solutions for a given problem. These structured solutions contain (for the novice) unknown programming concepts like sub functions, loops or sequences. Furthermore, the novices have to translate their solutions into syntactically correct statements. The problem here is that most programming languages have a different syntax. Finally, the novices have to understand how programs are executed.

Too much challenges
for a novice

Taxonomy of programming environments and languages	Kelleher and Pausch created a taxonomy of programming environments and languages for novice programmers and described the different methods of how novices can learn computer programming. The mentioned teaching systems in their paper have the purpose to prepare novices for transitioning to general-purpose languages like Java. The teaching systems concentrate on the mechanics of programming and splits into expressing the novice's intentions to the computer and understanding program execution.
Understanding program execution	There are approaches that start with imparting the understanding of program execution rather than how to enter code [Lee et al., 2014]. The novice gets a broken program first and has to debug and fix it before starting to create an own program from scratch. The results of this approach were positive and novices learned programming concepts like conditions or loops in only a couple of hours.
Expressing programs	Another method to teach mechanics of programming is by assisting novices to express their intentions into syntactically correct statements. This can be achieved by either simplifying how to enter code or by finding alternatives to typing programs. Simplifying the code entry can be achieved by simplifying the language itself or by preventing syntax errors. In [McIver and Conway, 1999], the self-proclaimed "pre-language" dissociates from real-world syntax of general-purpose language and uses a smaller language. Simultaneously, the syntax of <i>GRAIL</i> was simplified.
Find alternatives to typing programs	Instead of simplifying the code entry, it is also possible to use an alternative way to typing programs. This can be accomplished by constructing programs with either graphical or physical objects. These teaching systems bypass the syntax by encoding the syntax into the shape of objects. Each of these objects represent a different command and can be assembled in a special area.
Physical objects	Using physical objects (alias tangibles) to learn computer programming has a positive influence. Brauner et al. [2010] tested the effect of tangibles on teaching programming to seventh graders. Additionally, it was tested how the gender influenced the learning process. The pupils had to program

a robot's movements. One group got a virtual robot displayed on a computer screen, the second group interacted with a *LEGO Mindstorms NXT* robot. The results showed that the tangibles had a positive influence on the seventh graders. The tangibles put Science, Technology, Engineering and Mathematic (STEM) topics into a meaningful perspective. But the study also confirmed the huge gender gap with regard to STEM topics. This topic is also important and has to be dealt with.

The general idea of using physical objects to control virtual objects on screens was already introduced in 1995 with *Bricks* [Fitzmaurice et al., 1995]. At this time, teaching systems with tangibles like *AlgoBlock* [Suzuki and Kato, 1995] already existed, but without using a screen. The authors of *Bricks* built also a prototype to further investigate the UI concepts. Back then they concluded that this technology is highly potent and hoped for detailed exploration of it. Over the years, a great number of applications were created based on this idea. In section 2.2, we will present some applications using virtual and physical objects, respectively.

Tangibles on screens

2.2 Teaching Systems

*Lightbot*¹ was created in 2008 and represents a teaching software for learning computer programming, using virtual objects instead of programming code. The game can be played by using a mouse and a touch screen, respectively. In figure 2.1 the UI of *Lightbot* is shown to give an idea of how a teaching software can look like. Programming commands are encoded in each graphical block and user can built programs by assembling them in a programming grid. The created program can be executed and the robot moves according to the used blocks. This game was released for computers first and over the time, different versions of the game found their way to multiple platforms, like smartphones or tables. In the Google Play Store, the first version was downloaded between 100,000 and 500,000 times to the present day and got a very good rating of 4.4/5 stars in 8990

Lightbot

¹<https://lightbot.com/>

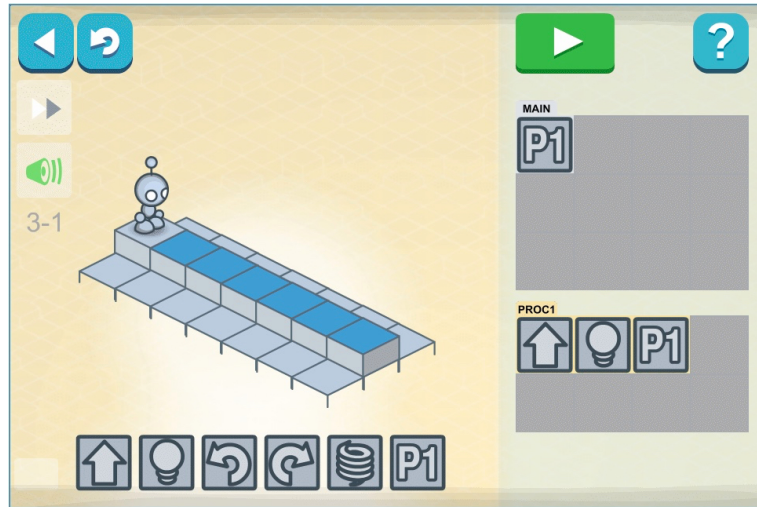


Figure 2.1: The game *Lightbot* uses graphical objects instead of programming code to teach novices the basic programming concepts.

votes. The iOS version was featured several times by Apple, e.g. in the categories like "Best for Learning to Code" or "Best in Hot Educational Games". This does not include the numbers from computer players or the other released versions.

Scratch

Another teaching system was developed at the Massachusetts Institute of Technology (MIT) Media Lab Resnick et al. [2009]. *Scratch* is a visual programming language to easily create small games or animations. This teaching system also uses code blocks instead of programming code. Unlike *Lightbot*, the user can use conditionals and variables. Additionally, drawn pictures (also called sprites) can be included into the program. In figure 2.2 Scratch's UI is shown. The different code blocks are divided into several categories and the user can use them by dragging them into the programming area. Depending on the number of used code blocks, the loop and conditionals automatically resize. Just as *Lightbot*, *Scratch* started with an application for the computer and then rolled out a version for children to the Google Play Store and the Apple App Store with a comparable success.



Figure 2.2: *Scratch* is a visual programming language to create small games or animations.

Lightbot and *Scratch* are only two of many other teaching systems. However, they concentrate only on visual programming with virtual elements. Since physical objects have a proven positive effect on novice programmers, other systems were created using physical elements, too. *Osmo*² for example uses physical blocks to create programs as shown in figure 2.3. The tablet uses its front camera to scan the area right in front of it. After pressing the play button on the tangible, the scanned program is executed on the tablet. Google also created a teaching system called *Project Bloks*³. *Project Bloks* is a modular system for tangible programming. The tangibles can be connected to create a program and since they are modular, they can be customized to fit in every scenario or application. Another system is *Primo*⁴ with a fixed programming area. Children can plug blocks into an interface board and create a movement sequence for a small robot. The developers also provide a small map with tasks.

Teaching systems
with tangibles

²<https://www.playosmo.com/>

³<https://projectbloks.withgoogle.com/>

⁴<https://www.primotoys.com/>



Figure 2.3: The game *Osmo* combines visual programming with tangibles.

2.3 Tangibles

Tangibles on screens

The mentioned teaching systems use screens solely to display a playing field on it or use them for touch inputs. In none of them they are used to directly interact with tangibles like in *Bricks* [1995]. Nevertheless, the idea of using tangibles on screens was further investigated and three major technologies emerged from this idea. The first technology includes visual techniques to detect tangibles using cameras on the back side of a surface. The second one detects magnetic tangibles with an analog Hall-sensor and the last one uses capacitive tangibles on capacitive screens that can be found in smartphones or tablets.

Diffuse Illumination

One of the first technology's approaches is using Silicon Illuminated Active Peripherals (SLAP) [Weiss et al., 2009] widgets. The widgets are made of silicon rubber and acrylic and are detected by the table using a technique called Diffuse Illumination (DI). The table has an acrylic surface and the SLAP widgets have markers attached to the bottom side. Infrared (IR) LEDs flood the acrylic surface from below with IR light and a camera detects the light reflected by the markers. This way the software toolkit detects the different widgets. However, to detect touch input (for example, by a silicon keyboard), a different technique called



Figure 2.4: Acrylic rotary SLAP widget (by Weiss et al. [2009]) on an acrylic surface.

Frustrated Total Internal Reflection (FTIR) is used. With FTIR, the IR light is fed sideways into the acrylic surface. A touch disturbs the light and the camera detects the scattered IR light.

GaussStones uses magnetic fields to detect tangibles on portable screens. Each tangible possesses a magnet surrounded by a galvanized steel shield to eliminate interference between magnetic tangibles. An analog Hall-sensor is placed beneath the screen and is able to detect multiple magnetic tangibles above the screen. The sensor does not require touch screens and therefore can be attached to laptops. Since each magnet creates a magnetic field with a different intensity and polarization, the tangibles can be distinguished.

Shielded magnetic
tangibles

The third major technology involves tangibles on capacitive screens. The capacitive screens can be found, for example, in smartphones or tablet. This technology splits again into three categories. In the first category the tangible has to be touched in order to be detected. The tangibles of the second category are detected by the screen without being touched (so called passive tangibles). Tangibles of the third category (active tangibles) are also detected without being touched and also possesses active elements (battery, micro controller, Bluetooth). With the active elements, the tangible can be persistently tracked by the screen and provide additional functions like switches or buttons.

Capacitive tangibles

Clip-on gadgets [Chang et al., 2012] are part of the first category. All tangibles of this category are only detected by the screen when they are touched by a user. This effect is called

Clip-on gadgets

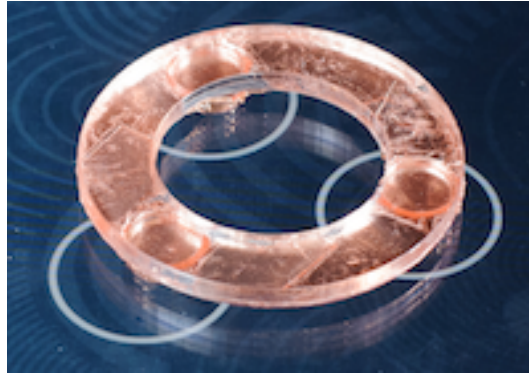


Figure 2.5: *PUCs* (by Voelker et al. [2013]) are detected by a capacitive screen without being grounded or touched by human fingers.

capacitive coupling [Zimmerman et al., 1995]. Capacitive touch screens detect grounded electrical conductors close to them. This grounded conductor can be a human finger. The *Clip-on gadget* consists of conductors that are connected to buttons (conductive rubber). The conductors are connected to the capacitive screen and every time the rubber button is touched by a human finger (or grounded), the capacitance gets changed and the conductor is sensed. The gadgets only need a small area and the screen and does not hide it.

Passive tangibles

Tangibles of the second category can be detected without being grounded or touched by a human finger. Passive Un-touched Capacitive Widgets (*PUCs*) [Voelker et al., 2013] are an example for this category. These tangibles also use the capacitive coupling effect in order to be detected by a capacitive touch screen. But in contrast to tangibles of category one, these tangibles utilize capacitive coupling on a second area. Each area is created by conductive pads at the bottom of the *PUCs* that are electrically connected to each other. Using this construction, active intersections on the screen [Barrett and Omote, 2010] are coupled to inactive intersections. The inactive intersections serve as ground and as a result, the tangible is detected by the screen. But the passive tangibles suffer from the issue that capacitive touch screens filter stationary touches after some time [Voelker et al., 2013] and they are no longer detected.

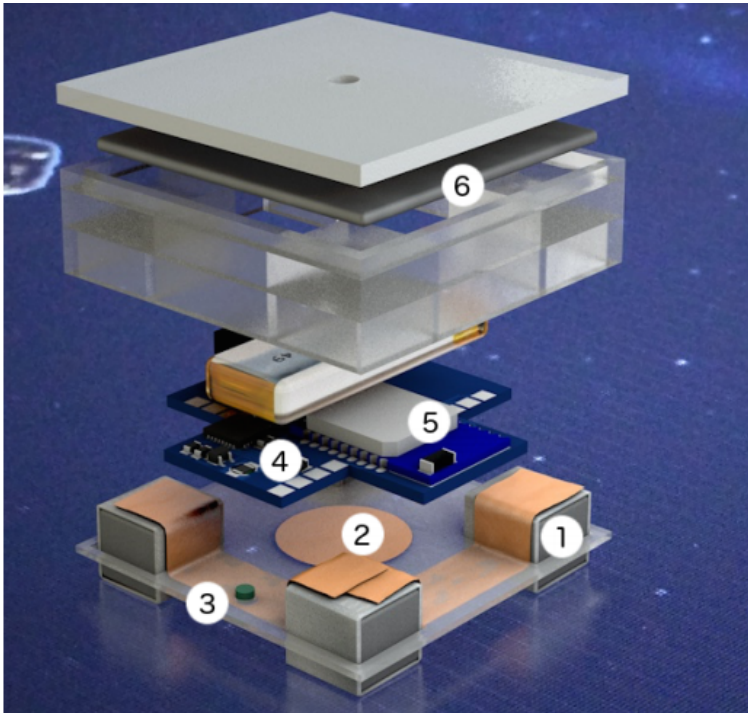


Figure 2.6: Design of a *PERC* tangible by Voelker et al. [2015]: (1) marker pattern, (2) field sensor, (3) light sensor, (4) micro controller, (5) Bluetooth element, and (6) lead plate.

To solve the filtering issue, the *PUCs* were further developed to active tangibles. The Persistently Trackable Tangibles (*PERCs*) [Voelker et al., 2015] have additional active elements (micro controller, battery, Bluetooth) and several sensors integrated. The *PERC* tangible uses a field sensor (figure 2.6) to detect whether it is on a capacitive touch screen or not. This characteristic is sent regularly to the application. Furthermore, using a light sensor and the marker pattern, it is possible to detect the orientation of the tangible. In addition, analog components, like switches or buttons, can be connected to the micro controller and create new applications for the tangible. The tangibles use the integrated Bluetooth element to communicate with other peripherals and to exchange data. The tangible can be connected, for instance, to a computer or a smartphone. Since, in contrast to *GaussStones*, *PERCs* does not require any fur-

Active tangibles

ther peripherals, they can be used on all capacitive touch screens.

MultiTouchKit
framework

In order to benefit from the Bluetooth communication and further features of the tangibles, the MultiTouchKit framework (*MTK*) was created by Linden [2014]. The *MTK* detects active and passive tangibles with the special marker pattern ([Voelker et al., 2015]) and establishes the Bluetooth connection with active tangibles. It serves as the interface between an application and the tangibles and handles all the communication. For every tangible connected to the screen, the application gets an object with all relevant information (position, orientation and so forth) for further processing. Additionally, the framework can process various types of input sources like touches by human fingers, *UITouch* (iOS touch events), JavaScript Object Notation (JSON) objects or mouse clicks. The data from all input sources is converted to so called *MTKTraces* and passed to the application. Several demo applications have already been created with the framework for multiple devices (55" capacitive screen from Microsoft, iPhone, iPad) and confirmed the support for various types of capacitive screens.

Chapter 3

Design

Gould and Lewis [1985] introduced three key principles for designing systems and UIs for people. Since the system and the UI are created for people, it is advisable to involve them into the designing process, which leads to the first principle. The first one states to "Early Focus on Users and Tasks". In other words, the designer has to understand the characteristics of the user and, in general, who the user will be. The system has to be matched to the performed work. The second principle suggests to let the users work with actual prototype. This way, their behavior, reactions, and performance can be documented and analyzed. The last key principle states to use an iterative process, a cycle, to create systems and UIs. Each iteration consists of designing the system, implementing it, measuring data and analyzing them. This can be generalized into three main steps per iteration: design, implement, analyze.

Key principles for designing a UI

The Design–Implement–Analyze (DIA) cycle is widely used to create UIs and applications for people. This process involves the target group already in early stages of projects. The observed behavior, performance, and reactions of the participants can be considered in the next iteration of the cycle to improve the iteration. The benefit of the iterative is that with each additional iteration, the application is further accommodated to the target group and potential design mistakes can be eliminated in early stages of the project.

DIA cycle

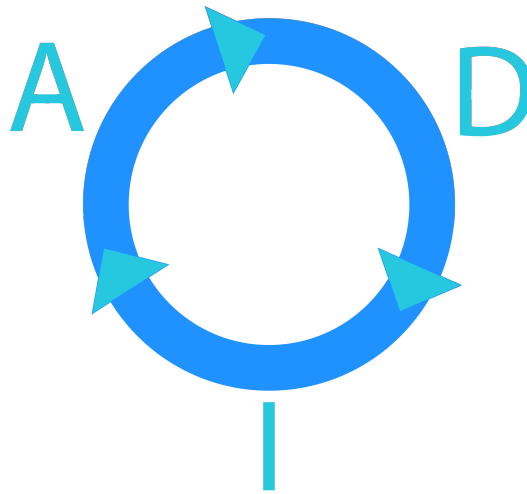


Figure 3.1: The DIA cycle is used to create the application *TangibleMike*. Three iterations were performed.

Three iterations of
the DIA cycle
performed

We made use of the DIA cycle while creating our application *TangibleMike* to benefit from the advantages of the iterative and user involved design. We performed three iterations of the cycle. The results of the first two iterations were paper prototypes and will be described in this chapter. In the third iteration we created the actual application. This iteration will be explained further in section 4.2.

3.1 Concept

General game
concept

In this section we will describe the general concept of our application *TangibleMike*. *TangibleMike* is a game for students in High Schools (secondary schools) and has the purpose to teach them basic programming concepts to them. The game serves as a simple programming environment and renounces complicated and unintuitive syntax.

Scenario of
TangibleMike

In this game, the students have to solve levels by navigating a builder called Mike. Mike starts at a certain point on a street and has to reach the end of the road. On his way to the finish, he approaches several dirty streets that were left

behind after a construction site. All dirty streets have to be cleaned in order to solve the level.

The user can control the main character by aligning blocks in special programming grids. Each blocks presents a different action and all actions can be split into two categories: movement actions and programming concepts. Blocks of the first category directly manipulate the movement of Mike (move forward, turn, clean dirty road). Blocks of the second category present different programming concepts that can be used (sub functions and loops). In later levels, the different kind of blocks have to be combined in order to solve them.

Navigate Mike with blocks

There will be two versions of our game. The first one will use graphical objects as blocks. The blocks can be used with touch gestures on a multi-touch tabletop. The second version will use tangibles, to be more precise *PERCs*, as blocks. The user can place them in the programming grid and create programs with them.

Graphical objects and tangibles as blocks

3.2 First Iteration

Since in the first iteration of the application there was no basic construct for the UI, we based our first design on the game *Lightbot*, presented in chapter 2. These kinds of games can improve the learning experience of programming languages [Kazimoglu et al., 2012] and looking at the number of downloads and rating, *Lightbot* is a good representative for this game category. At this point of the process, we still lacked a suitable scenario for our application and therefore temporarily adopted the scenario from *Lightbot*. Figure 3.2 shows the first drawing of our prototype. Mark (1) refers to the virtual version of the tangibles. Those tangibles can be assigned to particular cells in the main grid (alias the main function) (2). The second grid (3) can be used as a function call in order to execute repeatedly used commands. The buttons (4) act as the controls to execute the created program, stop the execution, and abort the game. (5) illustrates the playing field and exposes the assigned task to the user. The goal of every level is to navigate a virtual figure. This

The DIA cycle starts with paper prototypes

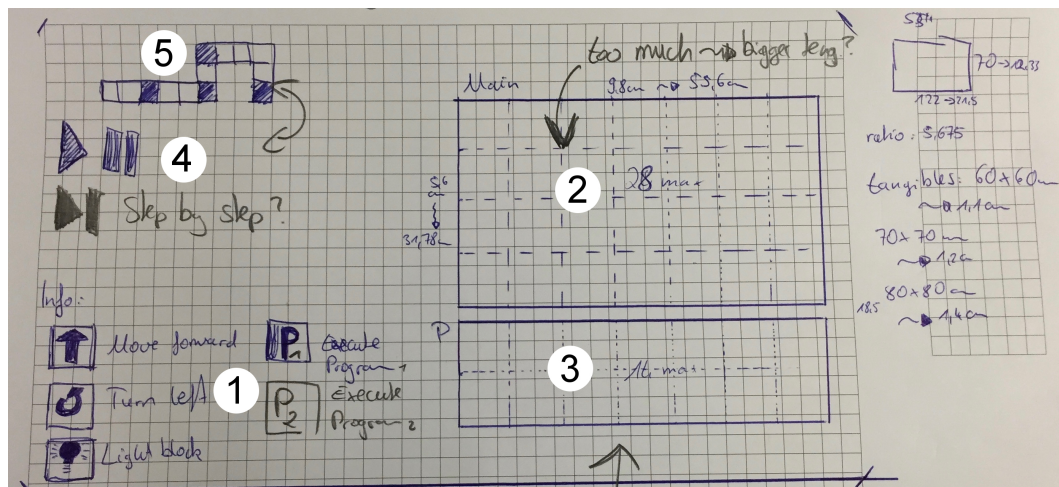


Figure 3.2: First basic UI of our application based on *Lightbot*. (1) tangibles, (2) main function, (3) sub function, (4) controls, (5) playing field.

figure has to reach the finish and perform one or several tasks on its way.

Implementation of the first prototype

Having a first idea of how the application should look like, the first prototype was built. The prototype in figure 3.3 illustrates the tangible version of the application. The graspable tangibles (1) are made of cardboard, since it is faster to create new ones or alter them. The drawn tangibles (2) just represent an explanation for every type of tangible. In this version, the information should be displayed permanently, in further versions as a non-permanent element triggered by a button tap. The first attempt of a loop looks like a clock (3). The user can place real or drag virtual tangibles onto this virtual element and adjust the number iterations of the respective tangibles. In this early version of the prototype, we didn't decide yet how to implement loops in our solution. Therefore, we initially allowed only function calls P1 and P2 to be repeatedly executed, as shown in figure 3.4. The playing field (4) and the controls (5) were adopted from the sketch with the exception that both switched places.

First evaluation of the prototype

After preparing 20 cardboard-tangibles with the size of 50x50mm and two different levels, we let four users make their first experience with our prototype. In this study we concentrated on the appearance and handling of our UI to-

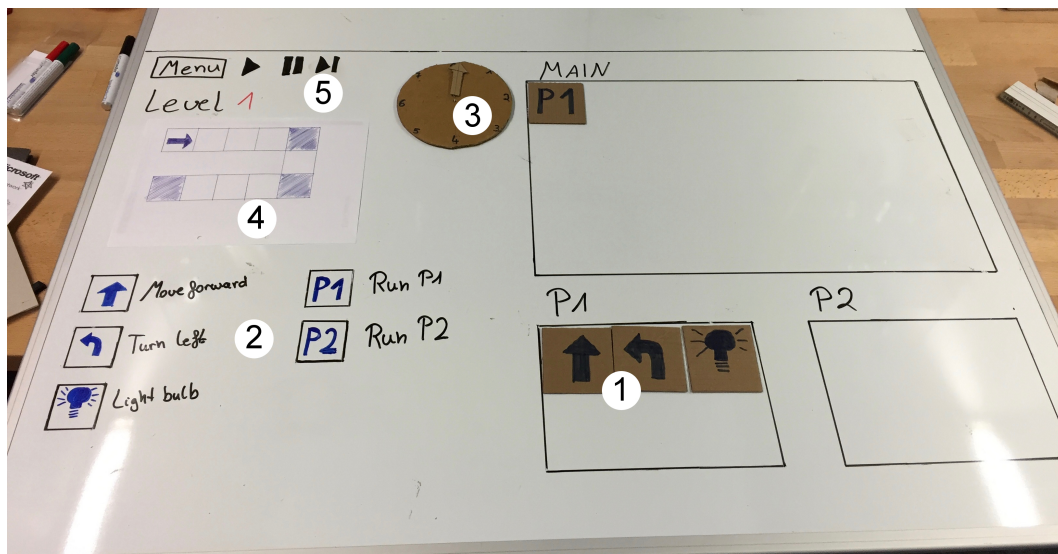


Figure 3.3: First prototype based on the drawing in figure 3.2. (1) cardboard tangibles, (2) tooltips, (3) loop area, (4) playing field, (5) controls.

gether with the comprehension of our application. In the first two studies we didn't prepare a questionnaire. Instead, we annotated the behavior, reactions and performance of the participants and asked them afterwards about their general opinion

The first two users studied computer science and therefore had experiences in programming. Both of them understood the basic concepts of programming represented by the tangibles and had several suggestions how to improve the tangibles. The other two users were total beginners and had struggle understanding the concepts of programming, especially with function calls and loops. With a little assistance, all managed to solve the first two levels of the game.

Participants differed in experience

The first evaluation showed that our loop attempt had to be improved. It was not clear how to create loop with several tangibles. Proposal was to create a further tangible specially for loops together with the possibility to mark all tangibles that are inside the loop. Another misunderstanding involved the turn-tangible. This tangible was either interpreted as turn movement or as the turn movement with a following forward movement. We initially planned to just make a turn without any further movements. Additionally,

Results of the first evaluation

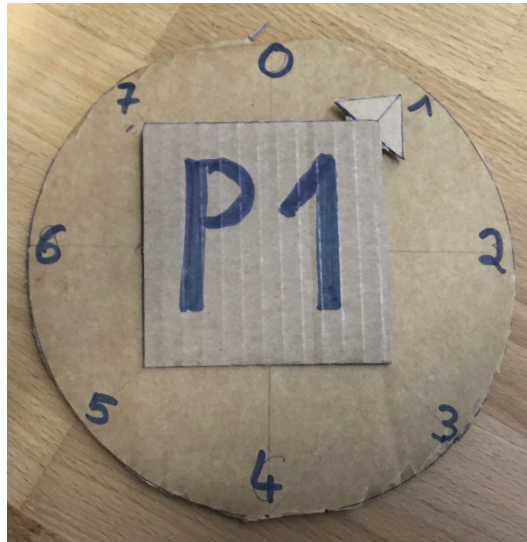


Figure 3.4: First implementation of a loop. The tangible has to be placed or dragged on this element to adjust the number of iterations.

we got the suggestion to hide the tooltips and show them only if requested. The other elements, like the forward-tangible, were positively evaluated. The participants had no struggle to understand and use them. With the helpful suggestions of the participants, we decided to make a second prototype before we started to create the application.

3.3 Second Iteration

Changes in the
second paper
prototype

With a second paper prototype, we intended to improve the UI further and clear the confusion about the tangibles. We took all the suggestions from the first evaluation into account and started to create the second paper prototype as shown in figure 3.5. The first improvement was the scenario of the application. We discarded the scenario of *Lightbot* and created our own, where a builder called Mike has to cross streets and arrive at the finish. On his way, he approaches several broken streets. In order to pass them, the broken streets have to be repaired. The light-up-tangible was replaced with the repair-tangible.

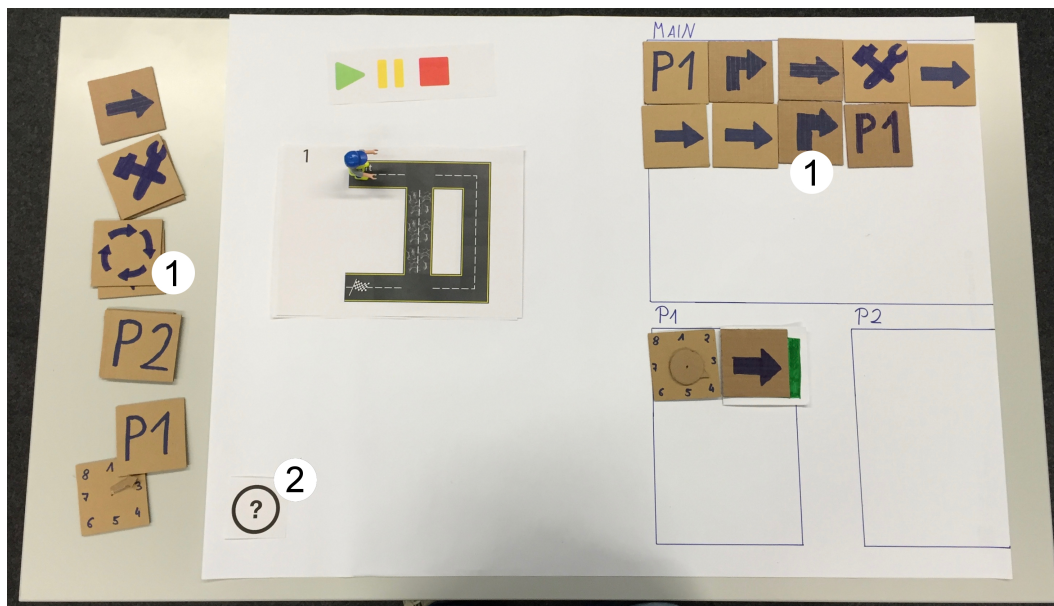


Figure 3.5: Second prototype influenced by the suggestions of the first prototype: (1) both versions of the turn tangible, (2) optional tooltips.

The second improvement involved the loops. We dropped the idea to adjust the number of iterations in a special area and instead created a loop-tangible. With this new tangible, the user has the ability to adjust the number of iterations directly on the tangible. This idea can be realized with real tangibles by mounting an analog rotary controller on top of the tangible. Furthermore, the range of this tangible is indicated by a green area surrounding the tangibles inside the loop. Every time a new tangible is placed inside the loop, its range grows by an additional cell.

New concept for the loop-tangible

Next we changed the symbol of the turn-tangible (1) (figure 3.5) to clarify that the figure only turns on the spot instead of making a further forward movement. For the evaluation, we used the old and new turn-tangible to compare both versions. Finally, we adopted the idea to hide the tooltips and show them temporarily after pressing the button (2).

New symbol for the turn-tangible

Six new users participated in the second evaluation of the paper prototype. Here, again, half of them had experiences in programming and the other half consisted of programming beginners. Just like in the first evaluation, we ex-

Second evaluation of the prototype

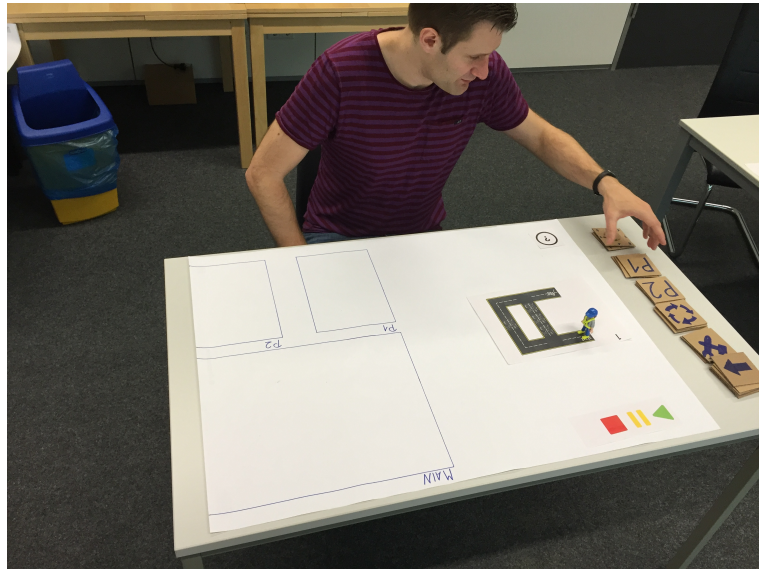


Figure 3.6: A participant is participating in the second evaluation.

emplary created levels for the users (in this case only one level). This was sufficient since this evaluation was primarily concerned with the design of the UI. Furthermore, this level was designed, so that it could be solved in several different ways, such as using only the main function or using function calls together with loops. To increase the difficulty and to encourage the users to use programming concepts, only two repair-tangibles were provided.

Results of the
second evaluation

Although this evaluation was performed to improve the UI, it was still interesting to experience how the users proceeded to solve the given task. To begin with, figure 3.7 shows the level the users had to solve. They could either choose a shorter way with three repair tasks or a longer way with only a single repair task. Out of the six users, four of them chose the shorter way with multiple repair tasks.

Difference between
experienced
programmers and
beginners

Solution (2) in figure 3.7 was provided by an experienced programmer, (1) and (3) by beginners. The experienced participant realized already at the beginning of the level that the number of forward- and repair-tangibles needed was insufficient for either ways and used loops and func-

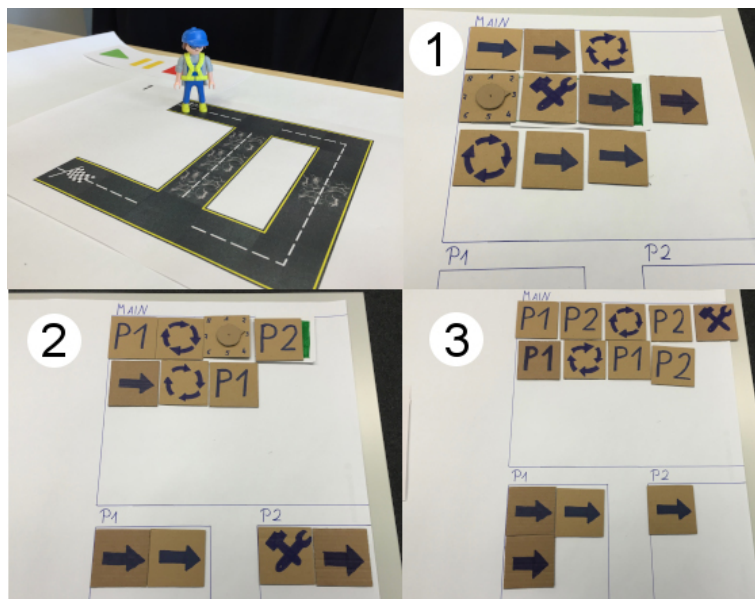


Figure 3.7: Different approaches to solve a single task. (2) was solved by an experienced programmer and (1), (3) by beginners.

tion calls. The other two experienced programmers behaved the same way and were additionally curious about the limitations of this game. They tried movements that led their virtual figure outside of the playing field to see the subsequent process or tried advanced concepts like recursion. All the beginners, however, tried to solve the level by using a single sequence in the main function. After some time, they also realized the problem with the limited number of tangibles and started to make use of programming concepts. User (1) remained in the main function and used a loop-tangible to accomplish the level whereas (3) used function calls. Every participant solved the level with a different set of tangibles. However, it is possible to direct the user to use the programming concepts by limiting the number of usable tangibles. By providing only a single repair-tangible in a level, where at least two broken streets have to be repaired, the user is forced to use at least a loop or a sub function. The difficulty can be further increased by deactivating the number of usable cells in the three functions (main, P1 and P2).

Further suggestions for the UI	Additionally, to the observations made in this second evaluations, the participants had several suggestions for the UI and issues understanding the repair-tangible. The users were not sure when exactly to use the repair tangible. The first possibility was to repair the broken street when standing right before it, the second one when standing on top of it. We initially planned to repair the street when standing on top of it, however, most participants commented that it would be more intuitive to repair the broken street before walking on it.
Results for the turn-tangible were still not satisfying	Furthermore, the users struggled with the design of the turn-tangible. For them, both variants of this tangible were still not appropriate. Since we couldn't achieve any successes with the new symbol, we had to think of a new one.
Most used elements were too far away	Five out of six users suggested to rearrange the whole UI. The elements of the UI used most the main function, the controls, and the tangibles. The tangibles are supposed to lie next to the tabletop. The main function and the controls, however, are fixed and cannot be rearranged while playing the game. The suggestion was to switch the position of the main function and the P1 and P2 functions and move the controls down to the bottom of the screen. The consequence would be that the most used elements are within easy reach and the user would be more comfortable.

Chapter 4

Application

This chapter will explain the technical details of the created application. It can be considered as the third iteration of the DIA cycle. The first both iterations were illustrated in chapter 3 and revealed several weaknesses in our initial design. One of them was the design of the turn-tangible. It wasn't clear if it only executes a turn movement or a turn movement with a forward movement afterwards. Additionally, the assembly of the UI wasn't ideal. The primarily used main function and the controls were on the upper side of the screen and therefore not within easy range. Furthermore, we made a slight change to scenario and changed the corresponding tangible as explained in section 4.2.

Third iteration of the DIA cycle

To begin with, in section 4.1 we are going to explain the whole hardware setup, including the used operating system (OS), the used tangibles, and the tabletop. Afterwards, in section 4.2, the latest design and architecture of the application will be presented.

Structure of this chapter

4.1 Hardware

The hardware setup of this application splits into three different components cooperating with each other. At first, the most important part is the integrated development en-

The hardware splits into three connected components

vironment (IDE), where the software is developed. Due to Apples restrictions of the IDE, the application can only be developed on Macintosh personal computers (PCs). Further details will be provided in section 4.1.1. The second component is the tangibles interacting with the tabletop and finally the used tabletop itself.

4.1.1 Operating System

MacOS is currently the only choice for development

Our application *TangibleMike* is based on the MultiTouchKit framework (MTK) created during the Master's thesis of Linden [2014]. The framework was created with Apples IDE XCode and the programming language Objective-C .

An additional OS to capture the touch points

Nevertheless, an additional OS was needed in order to get the touch points from the 55" Microsoft Surface Hub. We used Microsoft Windows 8.1, running inside a virtual machine (VM) on the iMac that is connected to the tabletop. The VM captures the touch points and passes them to the application for further processing.

4.1.2 Tangibles

Requirements for the tangible

In chapter 2, we have already presented different kinds of tangibles. Since we are using a capacitive tabletop, the optical and magnetic variants are unsuitable. The tangibles needed had to be persistently trackable, distinguishable, support analog controller, and posses a communication interface.

Stationary touches are filtered within seconds on capacitive displays

Persistently trackable. It can take several minutes to solve a level when playing *TangibleMike* and, therefore, the tangibles have to be persistently trackable by the application. However, as stated by Voelker et al. [2015], the capacitive displays have filtering mechanisms that remove stationary touches after a short period of time. As a consequence, the user would have to move all the used tangibles every few seconds. This would lower the experience for the user dramatically and the game would be unusable. Therefore, the

tangibles need to be persistently trackable without any user action.

Distinguishable. In the latest version of *TangibleMike* we are using six different tangibles. Each of these six tangibles execute unique commands (forward, turn, loop et cetera). Therefore, each of these tangibles needs to be distinguishable. Furthermore, there are multiple copies of each of them. The number differs in each level to adjust the difficulty. Five of the six tangibles execute simple commands and it is not necessary to distinguish among themselves. But each loop-tangible can have a different number of iteration and it is important to distinguish them. Using a full set of tangibles with a single copy of the loop-tangible requires the application to recognize 6 different tangibles and with each additional loop-tangible the number increases by one. Depending on the level-difficulty, the number can increase further. Additionally, in future work, the application can be extended and new tangibles can be introduced, which results in a higher number of different tangibles.

The full set contains at least six different tangibles

Support analog controllers. The loop-tangible in our second paper prototype possessed an analog rotary controller to adjust the number of iterations. The evaluated users approved this. In order to adopt this idea in our actual application, the real tangibles also have to possess this rotary controller. Furthermore, we already mentioned new tangibles in the last paragraph. This could be, for example, a turn-tangible with a small switch to change the turn-direction. This would also require the tangible to accept analog input and to pass this information to the application.

Loop-tangible with analog controllers

Communication interface. Any kind of input, such as the number of set iterations, has to be transferred to the application. This could be done with a wire or rather wireless. The two well-known wireless technologies are WiFi and Bluetooth. Version 4.0 of the Bluetooth Core Specification introduced Bluetooth Low Energy (BLE) to save energy and, therefore, we choose this technology to guarantee a longer playtime. Using Bluetooth for every tangible, it is possible to distinguish each tangible via its individual Bluetooth identifier (ID).

Use Bluetooth 4.0 to save energy and distinguish the tangibles

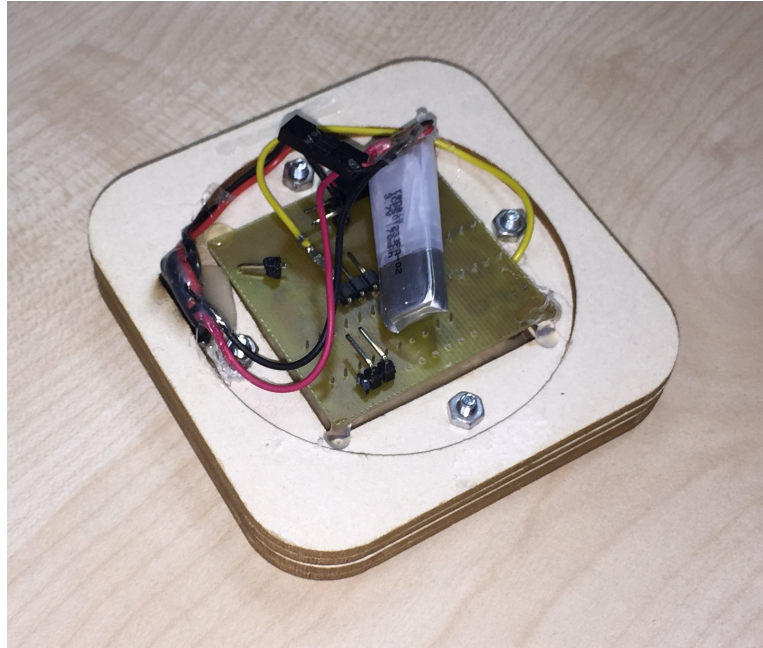


Figure 4.1: Current prototype of a *PERC* tangible. This one was used to test our application.

PERCs as the
tangibles of our
choice

Considering the four mentioned requirements, we choose *PERCs* (chapter 2) for our application. First of all, using a field sensor, *PERCs* can detect whenever they are on the tabletop or not. Furthermore, a Bluetooth adapter is implemented and therefore data can be exchanged between the application and the tangible. Additionally, the tangibles can be distinguished via the Bluetooth ID. Last but not least, *PERCs* are using micro controllers and possess input/output ports. Analog components such as rotary controllers can be connected. *PERCs* fulfill all requirements and additionally comes with a framework to support all functions in our application. To distinguish the tangibles visually, the transparent top case can be opened and a piece of paper with the respective symbol can be inserted.

Issues with *PERCs*

Unfortunately, the *PERCs* are still in development and therefore several problems come up. While implementing TM, we had struggles to persistently track the *PERCs*. After several seconds they were not recognized by the tabletop, while an older version of the tangibles worked without any

issues. After many hours of debugging and troubleshooting, we found out that a faulty feature in the MTK framework filtered the recent version of *PERCs*. Usually, a field sensor detects whether the tangible is on the tabletop or not. Additionally, the light sensor was brought in as a backup solution for this purpose. But the latest software running on the micro controller of the tangible was not compatible. By removing this function from the MTK, we were able to solve this problem. Another problem was the transmitted flood of data via Bluetooth. The tangible sent its information many times per seconds. Using only a handful of them was no problem at all, but *TangibleMike* was designed to support up to 20–30 tangibles or even more (if supported by the tabletop). Using this many tangibles could lead to serious performance issues. Since the tangibles are mainly straying at the same place, requesting the needed information by the application would reduce the data flood.

In order to eliminate this and several other small issues, the inner life of the tangibles is currently being renewed. The tangible benefits from this remake since new functionalities will be added. The new design includes a power switch and a universal USB port. Currently, it is only possible to power off or reset the tangible by unplugging a connector. This method was very inconvenient and time-consuming when dealing with a higher number of tangibles. Implementing the switch would simplify this process. The universal USB port could be used both for charging the tangible and also for updating the software of the micro controller. At this point, a special adaptor is needed to charge the battery and an additional micro controller to update the software. However, the renewal of the tangibles is not part of this thesis and won't be further discussed.

Since the tangibles are still in development, it is not possible to use them in combination with *TangibleMike*. A customized prototype of them was working with our application, but it is still uncertain if a greater number of them will work smoothly with our application. We tested four of them simultaneously and could not detect any issues, but we already experienced the mentioned problem in relation to the powering of the tangibles. It was time-consuming to power on and reset them. Up to this point, student as-

Renewal of the
tangibles

PERCs are still not
ready for use in
combination with
TangibleMike

sistants built the tangibles and technical issues while soldering could not be ruled out. But the built process will be outsourced to a manufacture in order to prevent technical issues and reduce the size of the inner life. Once this is done, we can test the compatibility with our application further and perform an extensive study to evaluate the usage of tangibles.

4.1.3 Tabletop

Capacitive
multi-touch tabletop

The MTK framework by Linden [2014] was mainly created and tested on a 55" Microsoft Surface hub and after the small adjustment has proven to be reliable regarding the recognition of the tangibles. Therefore, we used the same model for this thesis. The tabletop has a capacitive multi-touch screen and detects up to 100 touch points at the same time. Furthermore, the tabletop has an integrated and full-fledged computer. However, we only captured the touch points using the screen and passed them to our application that was running on an iMac. The developers designated the tabletop to be used in an upright position. For our purpose, however, a special frame was built and the tabletop was mounted on top of the frame in order to use it in a horizontal position (figure 4.2).

4.2 Software

Third iteration of the
DIA cycle

In this section we will describe the latest working version of our application *TangibleMike* in detail. The application is part of the third iteration of the DIA cycle. This iteration is based on the prior iterations and eliminates the problems and misunderstandings of the paper prototypes. In section 4.2.1, we will detail the changes of the design between the latest paper prototype and the actual application. Section 4.2.2 will describe the underlying architecture of our application and how the individual parts collaborate with each other.



Figure 4.2: 55" Microsoft Surface Hub lying on a specially constructed frame.

4.2.1 Design

The latest version of our application consists of two different game modes. Both modes only differ in the variant form of tangibles, everything else is equal. The first mode uses "virtual" tangibles, just like Lightbot. The user is dragging the tangibles from the pool (1) in figure 4.3 to the programming areas (2) or (3). In the second mode *PERCs* are used as tangibles and therefore the pool of virtual tangibles (1) is deactivated. The user can choose the mode in the options menu. We implemented the virtual mode in order to evaluate the real tangible. Performing a study with real tangibles only would provide results, but without any reference values we wouldn't be able to make a statement whether the results are positive or negative. And since we want to evaluate if tangibles can motivate young students to learn programming, the participants will play both modes one after another and compare them afterwards.

In the evaluation of the second paper prototype we stated that the participants had struggle to find the right time to use the repair-tangible. It was not clear if the tangible had

TangibleMike consists of two different game modes

Modified scenario of the game

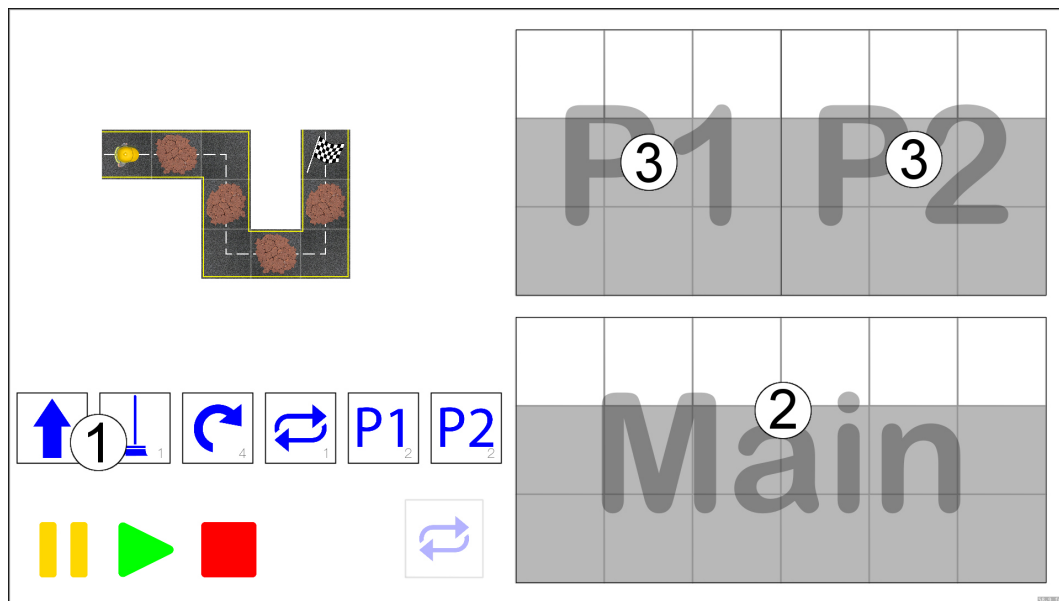


Figure 4.3: This is the UI of the third iteration of the DIA cycle and the latest version of the application.

to be used before stepping on the broken road or afterwards. Therefore, we thought of a slightly different and, hopefully, more intuitive scenario. In our new scenario, the protagonist Mike still has to reach the finish. But now, instead of broken roads, there are dirty roads Mike has to clean in order to accomplish the level. The conceptual difference here is that something broken cannot be used until it is repaired. But it is still possible to use something dirty or step on a dirty road. Therefore, Mike has to step on the dirty road first and clean it afterwards. The repair-tangible was changed to the more suitable clean-tangible with an appropriate symbol.

New symbol for the
turn-tangible

Participants initially complained that the angular design of the arrow would indicate a turn movement with a closing forward movement, therefore we again changed the turn-tangible's symbol. The second symbol with the four circular arrows was associated as a loop-tangible. After looking up several symbols for a turn movement in different games, we adopted finally the symbol from Lightbot. Lightbot was released in 2008 and, since then, has proved to be successful. The new symbol combines the single arrow of our first

attempt with the round shape of our second one.

The participants proposed to rearrange the whole UI. In every single level the main function has to be used to create a program, even if it is merely used to call the P1 or P2 sub functions, containing the remaining commands. Furthermore, the control buttons (4) in figure 4.3 are also used in every level to execute the built program. Since we are using a 55" tabletop, it may be hard to reach the constantly used elements, especially for smaller people. Hence, we moved the control buttons to the lower part of the UI and switched the main function with the sub functions. The playing field does not require any actions from the user (except for the built program) and therefore remained at the same place.

Newly arranged UI
for *TangibleMike*

4.2.2 Architecture

The architecture of our applications is composed of several independent parts. In this section, each of these parts will be explained and pointed out how they work collaboratively with each other. The parts are as follows:

The architecture
consists of six parts

- storyboard
- playing field
- tangibles
 - virtual
 - real
- programming grids
- program execution
- data logging

Storyboard. The storyboard is the base frame of the application and controls the flow of the play. To do so, we are using the State Machines from Apple's GameplayKit [2016]. Each scene in figure 4.4 stands for a different state. Each

Storyboard as the
base frame of the
application

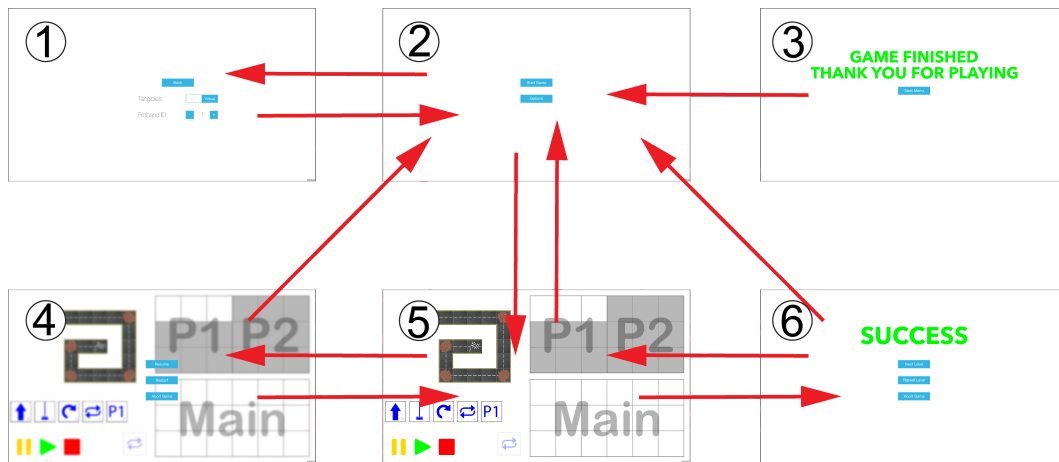


Figure 4.4: All scenes in *TangibleMike*: (1) options menu, (2) main menu, (3) game finished, (4) pause menu, (5) game scene, (6) level solved.

time the state machine changes to another state, the according scene is shown to the user. Scene (2) is the main menu from which the user can start the game or head to the options menu (1). In the options menu, the user can choose between real or virtual tangibles. Furthermore, the user ID can be set for the purpose of logging user data. Scene (5) is the actual game and from this state it is only possible to enter the pause menu (4) by touching the according button or the game over scene (5) by finishing the level. Inside the paused state, the level can be resumed, restarted, or aborted by moving to the main menu. The game over scene shows two different messages depending on whether the final level was solved (3) or a previous one (6). Solving a level prior to the final one gives the option to abort the game, repeat the last solved level or move to the next one. Finishing the last level of the game gives only the option to move to the main menu.

Multidimensional
array as the playing
field

Playing field. The playing field is a multidimensional array with eight rows and nine columns. The dimension can be easily changed by modifying the appropriate variable in the code. For the existing levels, the current size was sufficient. Each cell of the resulting grid is an object and stores several important information. These are: own position and size, whether it is active or not, possible directions if active, pointers to all direct and active neighbor-cells, and

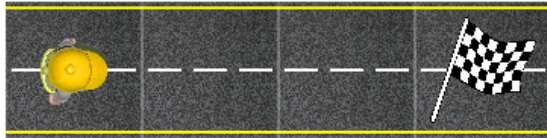


Figure 4.5: This is the resulting level from the code in Listing 4.1

flag if it is the last cell on the road (finish).

A cell can be activated by assigning a sprite to its object. There are eleven different street segments and each of them comes with a second variant. The distinction of each segment is made between a clean and a dirty street. The dirty variant has an additional sprite representing dirt. In order to simplify the creation of the level and prevent logic errors by creating wrong links between the cells, most of the procedure is atomized. In this version, the creating of a new level still has to be done in the code. An example for a level can be seen in the code below.

Activate a cell by
assigning a sprite

Listing 4.1: Code for the first level of TM

```

1  if (number == 1)
2  {
3      [[self returnCellAtRow:3 andColumn:2] addTextureWithNameToCell:@"01"];
4      {...}
5
6      .difficultySettingsForLevel = [NSDictionary dictionaryWithObjectsAndKeys:
7                                     [NSNumber numberWithInt:3], @"Forward",
8                                     [NSNumber numberWithInt:0], @"Turn_Right",
9                                     [NSNumber numberWithInt:0], @"Turn_Left",
10                                    [NSNumber numberWithInt:0], @"Loop",
11                                    [NSNumber numberWithInt:0], @"Clean",
12                                    [NSNumber numberWithInt:0], @"P1",
13                                    [NSNumber numberWithInt:0], @"P2",
14                                    [NSNumber numberWithInt:3], @"MainGrid",
15                                    [NSNumber numberWithInt:0], @"P1Grid",
16                                    [NSNumber numberWithInt:0], @"P2Grid", nil];
17
18      .firstCell = [self returnCellAtRow:3 andColumn:2];
19      .lastCell = [self returnCellAtRow:3 andColumn:5];
20 }

```

Line 3 shows exemplarily how sprites are assign to cells and thus activate them. The following lines set the difficulty of the first level by limiting the amount of usable tangibles and the number of active cells in the respective program-

Purposes of the
program instructions

ming grids. Lines 18–19 set the first and last cell. This has to be declared since the builder Mike starts at the first cell and the level is finished, when all cells are cleaned and the last cell is reached. Applying this configuration, the level in figure 4.5 is created. As already said, the connections between the cells are set automatically and therefore the creator does not have to take any more actions.

Virtual tangibles

Tangibles. We moved the most used elements to the lower area of the screen, therefore, we also positioned the virtual tangibles in this area. In figure 4.3 (1) they can be seen on the lower left side, right above the control buttons. The number of usable tangibles is predefined in the level editor and the corresponding tangibles are drawn on the scene. Each tangible has a counter, displaying the number of remaining tangibles of each kind. After the last one of a kind is used, the starting position remains empty, until one of used tangibles is removed from the programming grid. The user can move the tangibles by dragging and releasing the touch. After releasing the touch, there are three options. If the tangible is inside one of the programming grids, it is assigned to the cell next of its center position. If the cell is occupied, the tangible is reset to its starting position on the lower left side of the scene. Finally, if the tangible is released outside one of the programming grids, it is automatically reset to its starting position. Additionally, to remove a tangible from the programming grid, the user just has to drag it outside the grid. The only exception is the loop-tangible. As already explained, it possesses an own area, where the number of iterations is set. This area is behaving like programming grid and only accepts loop-tangibles.

Real tangibles

The real tangibles, however, reside next to the tabletop, in a small basket, for example. Every time a real tangible is placed on the screen, the number of remaining tangibles of this kind is decremented. At this point, there is no visible counter of remaining tangibles. Actually this is done by providing the maximum number of real tangibles individually for each level. As already mentioned, the loop-tangible has its own area to set the number of iterations. Currently, the number of iterations is reset, when the loop-tangible is lifted from the screen. This can be changed in future work, so that it is stored in the application for each loop-tangible

until the application is exited.

Programming grids. Each programming grid consists of a single array and each position in the array contains a cell. Just as the cells from the playing field, these cells also stores a number of information needed. First of all, each cell can be active or inactive. Inactive cells are used to set a higher difficulty by reducing the overall number of available cells in the grid. Tangibles cannot be placed inside inactive cells and are reset to its starting position if tried. When deactivating a cell, every following cell in the respective grid is also inactive. When successfully placing a tangible inside a cell, the border-color is set. An empty cell has no glowing border. A cell with a tangible has a light blue border and the loop-tangible is highlighted with a green border-color as well as all tangibles inside a loop. This is done to differentiate the tangibles that are inside a loop (and inside the loop-block) and the ones that are not.

Each grid is an array of cells

Initially, we hadn't specified how to assign tangibles to a loop-block. A possibility was to extend the loop-block by manually marking the cells, but we decided to do this automatically to lighten the user's workload. When a new loop-tangible is assigned to a cell, the next cell is automatically added to the loop-block. However, when the next cell is already occupied, all following cells are added to the loop-block until a free cell or the last active cell is reached. By skipping the last (free) cell of the loop-block, the loop is closed in a transferred sense. When removing the loop-tangible, all green highlighted cells are turned back to normal (no color or blue).

Latest attempt for the loop-tangible

From the beginning, we deactivated the possibility to use recursion. The recursion is an advanced programming concept and we are concentrating on the basics. But still it would be possible to implement it. At the moment, we disabled the possibility to place a P1 or P2 tangible in any of the sub functions. By enabling it, recursion would be possible. Since the execution stops when the finish is reached or the builder Mike steps outside the street, no break condition would be needed. But the drawback would be that once the recursion has started no further tangible could be executed after the recursion.

Recursion was deactivated

Procedure of the
program execution

Program execution. The execution of the program is started by touching the play button on the lower left side of the scene. Once this is done, all elements except the pause and stop button are deactivated. During the execution, the user can still enter the pause menu. By doing this, the execution is aborted. By touching the stop button, the execution is also aborted and Mike is reset to his starting position. In a real IDE, changes to the code won't affect the actual execution and therefore we deactivated all other elements. Since it is not possible to prevent the user from moving the real tangibles, a snapshot of the current state of all programming grids is made. All used tangibles are written to a new array that is used by an execution algorithm to execute each action. The outcome of the created program is defined during the execution. In other words, at the beginning of the execution it is not certain if the program will fail or succeed. Before each forward movement, the execution algorithm checks whether the next movement is possible or not. If not, the program fails and Mike is reset to his starting position and an animated red X informs the user. If the next movement is possible, the algorithm executes it and checks afterwards if the finish is reached. That being the case, it is checked if all dirty streets are cleaned. If so, the level is finished and the application moves to the next scene. Otherwise, the animated red X is shown and Mike is reset to his starting position. Using the clean-tangible multiple times on the same spot does not affect the outcome of the level. This means that the user can't fail by cleaning the same spot over and over again. On a clean street, this action is ignored, since people can sweep a clean spot repeatedly in real life, even if it doesn't serve a purpose.

Data logging is
already implemented
in the latest version

Data logging. Due to the lack of enough real tangibles, we still cannot perform an extended study. But as a matter of prudence, we have already implemented a mechanism to log important data for the upcoming study. Since it is possible to change the identifier of the test person, we can differentiate the results of each test person without having to close the application and save the results manually. For each individual identifier (starting by 1), two comma-separated value (CSV) files are created.

The first file stores the results for each level (one dataset per level). Each dataset contains the following items: timestamp, user ID, level number, start and ending time, number of failures, number of each tangible and the snapshots of each programming grid. The second CSV file contains information about each pass of each level. Every time the play button is touched, a new dataset is created and appended to the file. The second file contains the following items: timestamp, user ID, level number, iteration, the number of used tangibles and also a snapshot of each programming grid. By using the CSV file format, the results of the study can be directly imported into a software package for statistical analysis.

Two separate files for logging

Chapter 5

Evaluation

5.1 Study Design

At the end of the third iteration of the DIA cycle, we once more performed a study to evaluate our application. Initially, we planned to involve real tangibles in our study, but as already mentioned in section 4.1.2, the tangibles are still being improved and we hadn't a sufficient number of working ones. Therefore, we altered our initial plan and reduced the extent of this study. Instead of evaluating the benefits of using tangibles instead of graphical objects, we continued to concentrate on improving the UI of *TangibleMike*.

Evaluating the third iteration of the DIA cycle

Since the application's general functions are working properly (including the tangible support), we concentrated on smaller bugs, the gameplay, and the appearance of the UI. The best method to find those bugs and to gather opinions is to let the participants play the actual game. This is a common method and used by game designers and software engineers, respectively. Therefore, we created ten different levels.

Study setup

We created the levels, just as for the extended study, with an increasing difficulty. The created levels can be seen in the Appendix A. The first five levels intend to introduce all the different kinds of tangibles and are kept short and

Level design

simple. Although level five introduced the sub functions, level six showed an example of how they can be used in a program. From level seven onwards, the user had to use a combination of different programming concepts to solve the level.

Procedure of the study

After signing the consent form, the study continued with the explanation of the subject, the goal, and the procedure of the study. After answering potential questions, the user started playing TangibleMike. In the first level, we gave a detailed explanation of the UI and explained how to create and execute a program and how to solve a level. Furthermore, in the first five levels, we explained the functions of each new tangible. This was necessary, since we haven't implemented tooltips and an introduction yet. The participant always had the possibility to ask us questions or abort the study. During the game, we notated potential comments of the user. After completing the game, the user had to fill out a short questionnaire (Appendix A).

5.2 Results

Eight subjects with programming expertise participated in the study

A total of eight participants, aged between 21–31 ($M = 24.5$, one female), took part in this study. All of them were university students and, except for one, studying computer science. The remaining participant studied technical communication with computer science as the second subject. Furthermore, all participants had between 3–15 years ($M = 7.4$) of experience in computer programming and rated their own experience on a scale of 1 to 7, where 1 is very unexperienced and 7 very experienced, with an average of 5.25. After the personal questions, the questionnaire split into two segments. The first segment covered questions about the application in general whilst in the second segment the participant had to rate the different tangibles.

Application in general

In the first segment the participants had to rate the implementation of the scenario, indicate whether they understood the task and could identify the learning objective, indicate if the UI was generally clear, and rate the overall impression of the application. The results are shown in figure

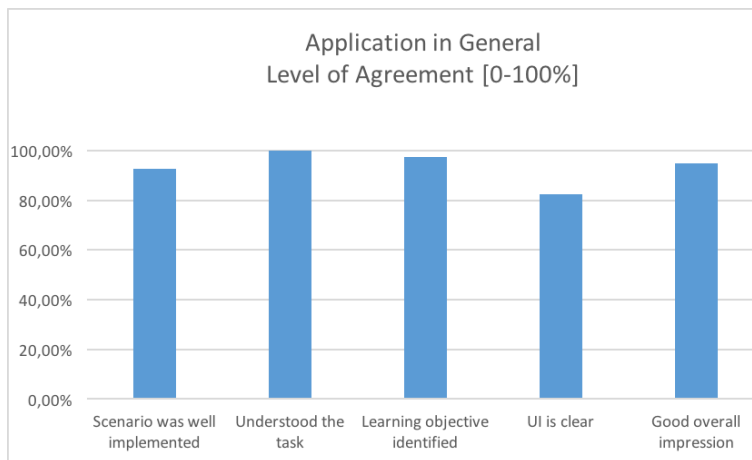


Figure 5.1: Results of the questionnaire's first segment.

5.1. In general, the application made a good impression on the participants. All participants understood the assigned task and could identify the learning objective of each level. The UI, however, will need further refinements in the following iterations of the DIA cycle. The border between P1's and P2's programming grids, for example, was ambiguous and hard to see. Furthermore, the UI needs a remake to improve the visual appeal.

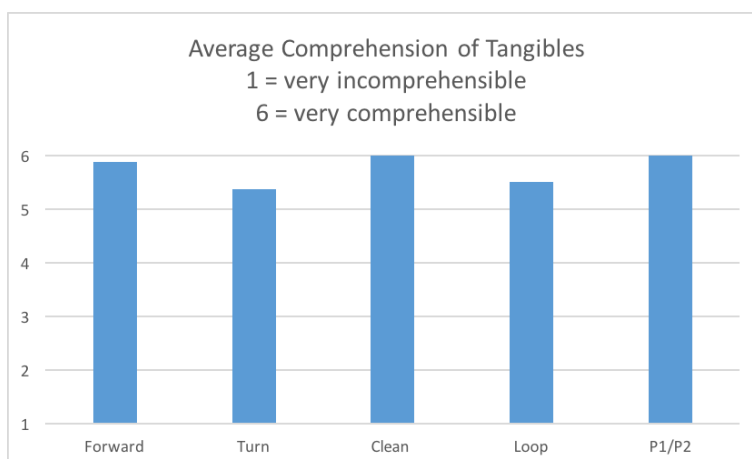


Figure 5.2: Results of the questionnaire's second segment.

Comprehension of the tangibles

In the second segments we asked to participants to rate the comprehension of each tangible (figure 5.2). As expected, the forward-tangible and the P1-/P2-tangibles were highly rated. Participants of both previous studies already had no problems understanding these tangibles. After the recent change of the scenario the clean-tangible was also maxed out. However, the turn-tangible and the loop-tangible still need further refinements. Despite the explanation of the loop-tangible, it still wasn't clear how to set the number of iterations. Additionally, this process is un-intuitive. Finally, some participants stated that the turn-tangible is missing an indicator that shows how far Mike is rotating.

5.3 Discussion

The three performed studies revealed several weaknesses of *TangibleMike*. We could overcome most of them by implementing the participant's suggestions. However, some of them like the loop-tangible and turn-tangible are still present. At the moment it is not possible to tell if tangibles can benefit the learning process of programming beginners. To perform the initially planned study we have to overcome the remaining weaknesses and wait for the completion of the PERCs.

However, we could still collect a lot of feedback in these three studies. We could confirm that all parts of our applications are working. The support for PERCs was also implemented and successfully tested with some prototypes. In order to test the limitation of the Bluetooth connection, a greater number of tangibles is needed.

The participants stated after the study that they had a nice time playing *TangibleMike* and offered to participate in the upcoming study. It will be interesting to see if we can motivate High School students the same way.

Chapter 6

Summary and Future Work

To conclude this thesis, we will summarize it and name ideas, how future work could improve our application and improve the learning process of programming beginners.

6.1 Summary

In order to address the shortage of IT specialists by motivating High School students to consider a career in the IT sector, we created an application to teach programming beginners the fundamental concepts of programming. In our application *TangibleMike* we are using *PERC* tangibles on capacitive multi-touch screens. *TangibleMike* is a game, where the user navigates a figure called Mike and solves programming tasks. Instead of using programming code, the user aligns blocks that represent different commands. These blocks are either graphical objects or tangibles. We created two versions in order to evaluate the tangible's benefits over graphical objects.

However, we couldn't perform the initially planned study due to technical issues of the tangibles. Instead, we further concentrated on improving the application and prepare it

for the final study. We asked several people with programming expertise to play *TangibleMike* and afterwards rate the game in general and each tangible in particular.

The study showed that all parts of our application are working. Additionally, we could successfully test our application with several prototypes of the *PERC* tangible. *TangibleMike* made a good overall impression and all participants understood the given tasks and could identify the learning objective of each level. Furthermore, the participants comprehended most of the tangibles. Only the loop-tangible and turn-tangible caused difficulties to some participants.

6.2 Future work

We already mentioned that the *PERCs* are being renewed and with the new tangibles we will be able to perform the comparison between the tangible and the non-tangible version of *TangibleMike*.

However, our application still needs refinements before the study can be performed. The symbol of the turn-tangible has to be clarified and the process of setting the loop iterations changed. We could add an additional indicator to the turn-tangible that shows the number of degrees it is turning.

In this thesis we presented the idea to attach an analog rotary controller to the tangible to set the number of iterations. This can be realized in further iterations. In order to have a consistent loop-tangible in both versions of our application, the virtual variant of the loop-tangible also has to be adjusted. In the virtual version the number could be set by performing a multi-touch rotary gesture directly on the loop-tangible.

In the latest version of *TangibleMike* the loop-block is automatically enlarged. Participants of the study stated that it would be better to set the size manually. We also noticed that participants intended to place a tangible outside

the loop-block but accidentally placed them inside it. The manual enlargement could be achieved by dragging the area with two fingers from the first cell of the loop-block to the desired last cell.

New tasks and other programming concepts can be added to our application. At this point we deactivated the possibility to use loops inside other loops and recursion. Activating them would provide a new level of complexity.

Finally, our application needs a visual refinement. The latest version is still a prototype to test the implemented elements of the game. In order to distribute *TangibleMike* to High School students, it is necessary to improve the appearance of the UI.

Appendix A

Appendix for the Application Study

Appendix A contains the levels of TangibleMike and questionnaire used for the study of the third iteration of the DIA cycle.

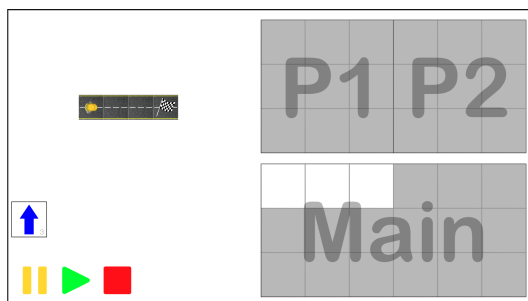


Figure A.1: Level 1 – objective: forward

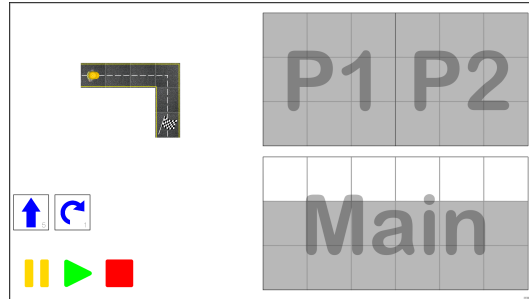


Figure A.2: Level 2 – objective: turn

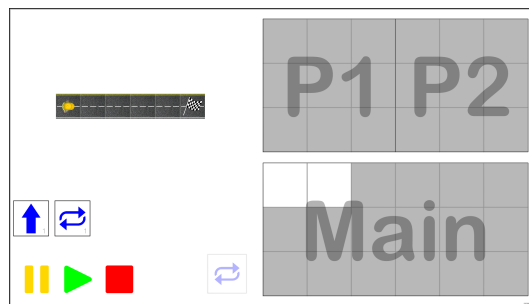


Figure A.3: Level 3 – objective: loops

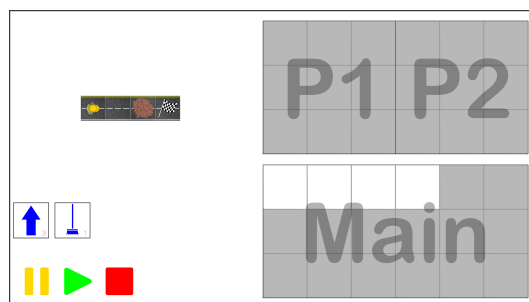


Figure A.4: Level 4 – objective: clean

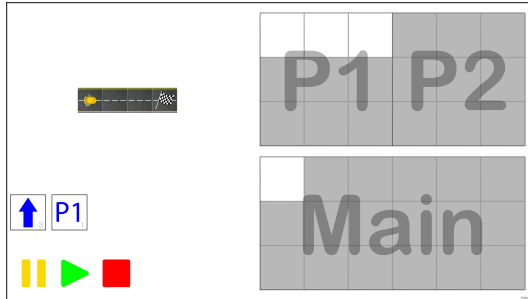


Figure A.5: Level 5 – objective: sub functions

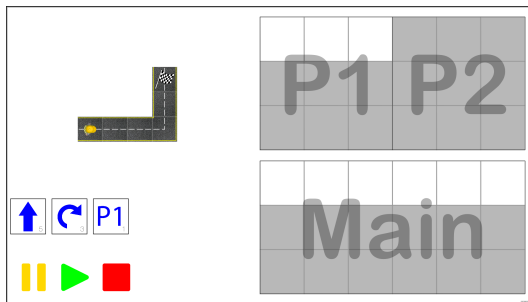


Figure A.6: Level 6 – objective: combination

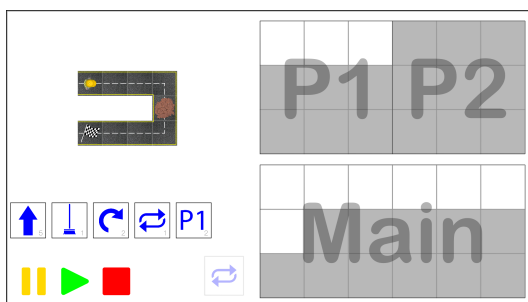


Figure A.7: Level 7 – objective: combination

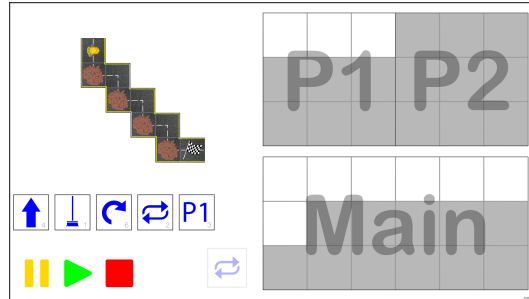


Figure A.8: Level 8 – objective: combination

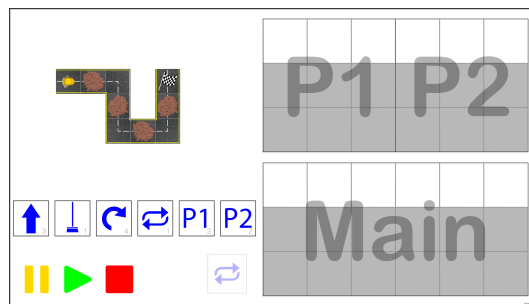


Figure A.9: Level 9 – objective: combination

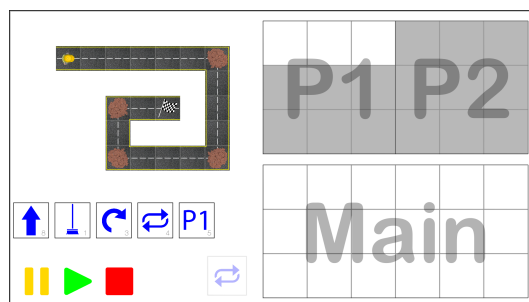


Figure A.10: Level 10 – objective: combination

FRAGEBOGEN

Liebe Teilnehmerin, lieber Teilnehmer,

Mit diesem Fragebogen bitten wir Sie, uns bei einem Forschungsprojekt am Human-Computer-Interaction Center (HCIC) der RWTH Aachen zu unterstützen. Das Projekt beschäftigt sich mit neuen Ansätzen um Computerprogrammierung interessant und motivierend zu gestalten. Wir bitten Sie um Ihre ehrliche Meinung, damit diese in die Entwicklung einfließen kann.

Bitte beantworten Sie diesen Fragebogen ehrlich und vollständig. Dabei zählt nur Ihre persönliche Meinung. Richtige oder falsche Antworten gibt es nicht. Alle Ihre Angaben werden anonymisiert ausgewertet und unterliegen dem Datenschutz. Rückschlüsse auf Ihre Person sind nicht möglich. Wenn Sie möchten, können Sie die Teilnahme jederzeit ohne Angabe von Gründen beenden. Vielen Dank für Ihre Teilnahme und dafür, dass Sie sich Zeit nehmen!

Damian Schablowsky
Dipl.-Inform. Philipp Brauner
Christian Chereck, M.Sc.
Prof. Dr. Jan Borchers
Prof. Dr. Martina Ziefle
VP-Nr.: _____
Datum und Uhrzeit: _____

Figure A.11: First page of the questionnaire and information for the test user.

II. Anwendung im Allgemeinen

Das Szenario wurde gut umgesetzt.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	stimmt gar nicht	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	stimmt absolut
Ich habe die gestellte Aufgabe verstanden.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	stimmt gar nicht	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	stimmt absolut
Ich habe das Lernziel der Anwendung verstanden.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	stimmt gar nicht	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	stimmt absolut
Das User Interface ist klar und verständlich.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	stimmt gar nicht	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	stimmt absolut
Die Anwendung macht im Gesamtbild einen guten Eindruck.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	stimmt gar nicht	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	stimmt absolut

Weitere Anmerkungen:

.....


.....

.....

.....

.....

Figure A.13: Third page of the questionnaire. General questions about the application.



III. Tangibles

Verständnis der Tangibles und deren Funktion in der Anwendung.

Forward	unverständlich	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	verständlich
Turn	unverständlich	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	verständlich
Clean	unverständlich	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	verständlich
Loop	unverständlich	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	verständlich
P1/P2	unverständlich	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	verständlich

Weitere Anmerkungen:

.....

.....

.....

.....

„Begreifbare Programmierumgebung“ – Damian Schablowsky

Seite 4 von 4

Figure A.14: Fourth page of the questionnaire. Questions about each tangible in particular.

Bibliography

Gary Barrett and Ryomei Omote. Projected-capacitive touch technology. *Information Display*, 26(3):16–21, 2010. ISSN 03620972.

David Beede, Tiffany Julian, and David Langdon. Women in STEM : A Gender Gap to Innovation. *U.S. Department of Commerce, Economics and Statistics Administration*, pages 1–11, 2011. ISSN 1556-5068. doi: 10.2139/ssrn.1964782.

Philipp Brauner, Thiemo Leonhardt, Martina Ziefle, and Ulrik Schroeder. The effect of tangible artifacts, gender and subjective technical competence on teaching programming to seventh graders. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5941 LNCS, pages 61–71, 2010. ISBN 3642113753. doi: 10.1007/978-3-642-11376-5_7.

Bundesagentur für Arbeit. Der Arbeitsmarkt in Deutschland - Fachkräfteengpassanalyse, 2016. URL <https://statistik.arbeitsagentur.de/Statischer-Content/Arbeitsmarktberichte/Fachkraeftebedarf-Stellen/Fachkraefte/BA-FK-Engpassanalyse-2016-06.pdf>.

Tzuwen Chang, Neng-Hao Yu, Sung-Sheng Tsai, Mike Y Chen, and Yi-Ping Hung. Clip-on gadgets: Expandable Tactile Controls For Multi-touch Devices. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services companion - MobileHCI '12*, page 163, 2012. ISBN 9781450314435. doi: 10.1145/2371664.2371699. URL <http://doi.acm.org/10.1145/2371664>.

2371699\$\delimiter"026E30F\$http://dl.
acm.org/citation.cfm?doid=2371664.2371699.

Dieter Westerkamp - VDI Verein Deutscher Ingenieure. IT-Standort Deutschland: Mangel an IT-Fachkräften gefährdet Wettbewerbsfähigkeit, 2012. URL https://www.vdi.de/uploads/media/2012-03-05_{_}CeBIT-Statement_{_}Westerkamp.pdf.

Bitkom e.V. In Deutschland fehlen 41,000 IT-Experten, 2014. URL <https://www.bitkom.org/Presse/Presseinformation/In-Deutschland-fehlen-41000-IT-Experten.html>.

G.W. Fitzmaurice, H. Ishii, and W.a.S. Buxton. Bricks: laying the foundations for graspable user interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 442–449, 1995. ISSN 00220221. doi: 10.1145/223904.223964. URL <http://portal.acm.org/citation.cfm?id=223964>.

John D. Gould and Clayton Lewis. Designing for Usability: Key Principles and What Designers Think. *Communications of the ACM*, 28(3):300–311, 1985. ISSN 00010782. doi: 10.1145/3166.3170.

Apple Inc. GameplayKit - State Machines, 2016. URL https://developer.apple.com/library/ios/documentation/General/Conceptual/GameplayKit_{_}Guide/StateMachine.html.

Cagin Kazimoglu, Mary Kiernan, Liz Bacon, and Lachlan Mackinnon. A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. *Procedia - Social and Behavioral Sciences*, 47: 1991–1999, 2012. ISSN 18770428. doi: 10.1016/j.sbspro.2012.06.938. URL <http://www.sciencedirect.com/science/article/pii/S1877042812026742>.

Caitlin Kelleher and Randy Pausch. Lowering the barriers to programming. *ACM Computing Surveys*, 37(2):83–137, 2005. ISSN 03600300. doi: 10.1145/1089733.1089734.

Michael J. Lee, Faezeh Bahmani, Irwin Kwan, Jilian Laferte, Polina Charters, Amber Horvath, Fanny Luor, Jill Cao, Catherine Law, Michael Beswetherick, Sheridan Long, Margaret Burnett, and Andrew J. Ko. Principles of a debugging-first puzzle game for computing education. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, pages 57–64, 2014. ISBN 9781479940356. doi: 10.1109/VLHCC.2014.6883023.

René Linden. *Multitouchkit: A Software Framework for Touch Input and Tangibles on Tabletops and Mobile Devices*. PhD thesis, 2014. URL <https://hci.rwth-aachen.de/materials/publications/linden2015a.pdf>.

Linda M McIver and Damian Conway. GRAIL : A Zeroth Programming Language. *Design*, 1999.

Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, J a Y Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for All. *Communications of the ACM*, 52: 60–67, 2009. ISSN 00010782. doi: 10.1145/1592761.1592779. URL [http://search.ebscohost.com/login.aspx?direct=true{%&}db=bth{%&}AN=45021156{%&}site=eds-live\\$\\delimiter"026E30F\\$files/130/RESNICKetal.-2009-ScratchProgrammingforAll..pdf](http://search.ebscohost.com/login.aspx?direct=true{%&}db=bth{%&}AN=45021156{%&}site=eds-live$\\delimiter).

Statistisches Bundesamt. Datenreport 2016, 2016. URL https://www.destatis.de/DE/Publikationen/Datenreport/Downloads/Datenreport2016.pdf?{_}{_}blob=publicationFile.

Hideyuki Suzuki and Hiroshi Kato. Interaction-level support for collaborative learning: AlgoBlock—an open programming language. In *The first international conference on computer support for collaborative learning*, pages 349–355, 1995. ISBN 0-8058-2243-7.

Simon Voelker, Kosuke Nakajima, Christian Thoresen, Yuichi Itoh, Kjell Ivar \Overg\gaard, and Jan Borchers. PUCs: Detecting Transparent, Passive Untouched Capacitive Widgets on Unmodified Multi-touch Displays.

Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces, pages 101–104, 2013. doi: 10.1145/2512349.2512791. URL <http://doi.acm.org/10.1145/2512349.2512791>.

Simon Voelker, Christian Cherek, Jan Thar, Thorsten Karer, Christian Thoresen, Kjell Ivar Øvergård, and Jan Borchers. PERCs: Persistently Trackable Tangibles on Capacitive Multi-Touch Displays. *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*, pages 351–356, 2015. doi: 10.1145/2807442.2807466. URL <http://dl.acm.org/citation.cfm?id=2807442.2807466>.

M. Weiss, J. Wagner, Y. Jansen, R. Jennings, R. Khoshabeh, J.D. Hollan, and J. Borchers. SLAP Widgets: Bridging the Gap Between Virtual and Physical Controls on Tabletops. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 481–490, 2009. ISBN 9781605582467. doi: <http://doi.acm.org/10.1145/1518701.1518779>.

Thomas G. Zimmerman, Joshua R. Smith, Joseph a. Paradiso, David Allport, and Neil Gershenfeld. Applying electric field sensing to human-computer interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '95*, number May, pages 280–287, 1995. ISBN 0201847051. doi: 10.1145/223904.223940. URL <http://portal.acm.org/citation.cfm?doid=223904.223940>.

Index

AlgoBlock, 7
analog controller, 26
Apple, 26
architecture, 33

Bluetooth low energy, 27
Bricks, 7

capacitive coupling, 12
Clip-on gadget, 11
compiler, 2
CSV, 38

data logging, 38
DIA cycle, ix, 15, 30
Diffuse Illumination, 10

framework, 26, 28
Frustrated Total Internal Reflection, 11

GaussStones, 11
GRAIL, 6

Hall-sensor, 11
hardware, 25
High School students, 47

IDE, 26
indicator, 46
infrared, 10
iteration, 21
iterative process, 15

JSON, 14

LEGO Mindstorm NXT, 7
Lightbot, 17, 20

Macintosh, 26
micro controller, 28

Microsoft Surface Hub, 26
MTK, 26, 29

Objective-C, 26
operating system, 25
Osmo, 9

PC, 26
PERCs, 28
persistently trackable, 26
playing field, 34
Primo, 9
program execution, 38
programming grid, 37
programming language, 2
Project Bloks, 9
PUC, 12

recursion, 23
rotary controller, 27
rotary gesture, 46

Scratch, 8
second paper prototype, 27
semantics, 2
SLAP, 10
source code editor, 2
stationary touch, 26
storyboard, 33
syntax, 2

tooltips, 21

WiFi, 27
wireless, 27

Xcode, 26

