Design and Evaluation of Tutorials on Mobile Devices

MASTER THESIS

at

Graz University of Technology

Institute for Software Technology (IST),
Graz University of Technology
A-8010 Graz, Austria

submitted by

Angelika More

Graz, 31 March, 2014

Advisor:
Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany

Design und Evaluierung von Tutorials auf Mobilen Geräten

MASTERARBEIT

an der

Technische Universität Graz

Institut für Softwaretechnologie (IST),
Technische Universität Graz
A-8010 Graz, Österreich

vorgelegt von

Angelika More

Graz, 31. März 2014

Diese Arbeit ist in englischer Sprache verfasst.

Begutachter:
Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany

Abstract

The importance of mobile devices (e.g. smart phones and tablets) is increasing. Therefore, the possibility for the usage of the technology of teaching and learning has emerged. The *Android* application *Pocket Code* developed at the *Technical University of Graz* takes this possibilities to arrange a platform for acquiring programming skills in a playful way. Underage persons are the target group for this application. During the work for this thesis, tutorial prototypes for this application were developed. After an introduction to *Android*, the *Catrobat Project*, *Development Methodologies*, and *Usability*, different theories for the creation of tutorials were examined. With the findings of the theoretical work, an approach for a tutorial was developed. Also different applications with tutorials are analyzed and are used as inspirational resources. Furthermore, evaluation methodologies were elaborated. In this work the evaluation methods of *Heuristic Evaluation*, *Cognitive Walkthrough*, *System Usability Scale*, *Emocards*, *A/B Tests*, and the *Thinking Aloud Method* are presented in detail. In addition, an evaluation concept for a future usability evaluation is arranged in this work.

Kurzfassung

Mobile Geräte (z.B. Smartphones und Tablets) werden immer wichtiger. Dadurch entsteht die Möglichkeit diese Technologie für die Lehre und das Lernen zu verwenden. Die *Android* Applikation *Pocket Code* wird an der *Technischen Universität Graz* entwickelt und bedient sich dieser Möglichkeiten um eine Plattform für die Aneignung von Programmierkenntnissen zu bieten. Minderjährige Personen sind die Zielgruppe dieser Applikation. Im Laufe dieser Arbeit wurden Prototypen eines Tutorials für diese Applikation entwickelt. Nach einer Einführung über *Android*, das *Catrobat Projekt*, *Entwicklungsmethoden*, und *Usability*, wurden verschiedene Theorien für die Erstellung von Tutorials ausgearbeitet. Mithilfe dieser Theorien wurde ein Ansatz für ein Tutorial entwickelt. Es sind auch andere Programme mit Tutorials analysiert worden, die als Inspiration für das erstellte Tutorial dienen. Außerdem wurden im Zuge dieser Arbeit Evaluierungsmethoden ausgearbeitet. Hier werden die Methoden von der *Heuristischen Evaluierung*, dem *Kognitiven Durchlauf*, der *System Usability Skala*, der *Emocards*, der *A/B Tests* und der *Thinking Aloud Methode* im Detail besprochen. Zusätzlich wurde ein Konzept für eine zukünftige Usability Evaluierung im Laufe dieser Arbeit erstellt.

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…… ……..
date (signature)

Acknowledgements

Special thanks go to Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany of the *University of Technology Graz* and the *Catrobat Team* for the encouragement and the opportunity to do this thesis.

Big thanks go to Rony Glabonjat, Claudia Kaplaner and Nadja Lauritsch for their reviewing work. I also want to thank Manuela Lobnig for many discussions and the support during some difficult moments.

Last but not least, I want to thank everyone who supported me while writing this thesis.

Angelika More

Contents

Abstract **V**

Acknowledgements **i**

List of Figures **vii**

List of Tables **ix**

1 Introduction **1**
1.1 Introduction to Android . . . 1
1.1.1 Application Structure . . . 2
1.1.2 Activities . . . 4
1.1.3 Services . . . 5
1.1.4 Content Providers . . . 7
1.1.5 Technical Approach for Creating a Tutorial . . . 7
1.2 Introduction to the Catrobat Project . . . 8
1.3 Introduction to Development Methodologies . . . 9
1.3.1 Extreme Programming . . . 9
1.3.2 Test Driven Development . . . 12
1.3.3 Clean Code . . . 12
1.4 Introduction to Usability . . . 14
1.5 Overview . . . 15

2 Theories for a Tutorial **17**
2.1 Challenges . . . 17
2.1.1 Story Development . . . 18
2.1.2 Tutor Development . . . 20
2.1.3 Engagement . . . 21
2.1.4 Guidelines . . . 22
2.2 Educational Purposes . . . 24
2.3 Inspiration for a Tutorial . . . 24
2.3.1 Scratch . . . 25
2.3.2 Wario Ware D.I.Y. . . . 26
2.3.3 Alice . . . 27
2.3.4 Kodu . . . 29
2.4 Gamification . . . 30
2.4.1 Definition . . . 30
2.4.2 Fantasy . . . 31

2.4.3 Goals . . . 31
2.4.4 Feedback and Guidance . . . 31
2.4.5 Progressive Disclosure . . . 32
2.4.6 Time Pressure . . . 32
2.4.7 Rewards and Punishments . . . 32
2.4.8 Stimuli . . . 33
2.5 Mobile Learning . . . 33
2.5.1 Definition . . . 34
2.5.2 Learning Principles . . . 34
2.5.3 Learnability . . . 36
2.5.4 Adaptive Learning . . . 36
2.5.5 Serious Games . . . 38
2.6 Implementation in Pocket Code . . . 38
2.6.1 First Prototype . . . 38
2.6.2 Second Prototype . . . 42
2.6.3 Tooltip Implementation . . . 42
2.7 Chapter Summary . . . 43

3 Types Of Tutorials **45**
3.1 Manuals . . . 46
3.2 Help Buttons/Tooltips . . . 46
3.3 Training Challenges . . . 47
3.4 Chapter Summary . . . 54

4 Evaluation Methodologies **55**
4.1 Heuristic Evaluation . . . 58
4.1.1 Basic Concepts . . . 58
4.1.2 Adaption for Underage Participants . . . 60
4.2 Cognitive Walkthrough . . . 61
4.2.1 Basic Concepts . . . 61
4.2.2 Adaption for Underage Participants . . . 62
4.3 System Usability Scale . . . 62
4.3.1 Basic Concepts . . . 62
4.3.2 Adaption for Underage Participants . . . 63
4.4 Emocards . . . 63
4.4.1 Basic Concepts . . . 63
4.4.2 Adaption for Underage Participants . . . 65
4.5 A/B Test . . . 65
4.5.1 Basic Concepts . . . 65
4.5.2 Adaption for Underage Participants . . . 66
4.6 Thinking Aloud Method . . . 66
4.6.1 Basic Concepts . . . 66
4.6.2 Adaption for Underage Participants . . . 67
4.7 Metrics . . . 67
4.8 Applying for a Tutorial . . . 69
4.8.1 Results from Previous Evaluation . . . 69
4.9 Chapter Summary . . . 70

5 Conclusion and Future Work **71**

A Cognitive Walkthrough Form **73**

B System Usability Scale Template **75**

C Background Questionnaire **77**

Bibliography **79**

List of Figures

1.1 Android Distribution March 2014 [AndroidDevelopersDashboard, 2014]. . . 2
1.2 Android Activity Lifecycle [AndroidActivities, 2014]. 5
1.3 Android Service Lifecycle [AndroidServices, 2014]. 6
1.4 Usability Attributes [Nielsen, 1994]. 14

2.1 Scratch with Step by Step Introduction [Scratch, 2014]. 25
2.2 Wario Ware D.I.Y. Tutorials Screens [Wario Ware D.I.Y, 2014]. 26
2.3 Alice Tutorial Selection [Alice, 2014]. 27
2.4 Alice Tutorial Screen [Alice, 2014]. 28
2.5 Kodu Tutorial Selection [Kodu, 2014]. 29
2.6 Kodu Tutorial Screen [Kodu, 2014]. 30
2.7 Tutors for the First Prototype. 39
2.8 Design of Tutor for the Second Prototype. 42

3.1 Design for Tooltips in Pocket Code in the Main Menu. 47
3.2 Tooltips in the Project Menu. 47

4.1 Emocard [Agarwal and Meyer, 2009]. 64

List of Tables

1.1 Android File Structure [AndroidProjectManagement, 2014]. 3
1.2 *Extreme Programming* Principles [Beck and Andreas, 2004]. 11
1.3 *Extreme Programming* Practices [Beck, 2000]. 12

2.1 Question Categories [Baecker et al., 1991]. 19
2.2 Influence Factors for Engagement in Software [Prensky, 2001]. 22
2.3 Rewards and Punishment in Games [Hallford and Hallford, 2001], [Gazzard, 2011], [Juul, 2009]. 33
2.4 Several Technologies for Adaption [Brusilovsky, 1998]. 37
2.5 *XML*-Elements for the Tutorial. 40

3.1 Examples for Help Systems [Baecker, 2002]. 46

4.1 Evaluation Phases [Balagtas-Fernandez and Hussmann, 2009]. 55
4.2 Aspects for Usability Testing with Underage Participants [Larkin, 2002] [Hanna et al., 1997]. 58
4.3 Basic Heuristics [Molich and Nielsen, 1990] [Nielsen and Molich, 1990] [Korhonen and Koivisto, 2006]. 60
4.4 Basic Heuristics for Underage Participants. [Alsumait and Al-Osaimi, 2009]. 61
4.5 Emotions in an *Emocard* [Agarwal and Meyer, 2009] [Desmet, 2000] [Stickel et al., 2011]. 65
4.6 Usability Metrics [Kelleher and Pausch, 2005] [Grossman et al., 2009] [Harrison, 1995] [Wong et al., 2003]. 68

Chapter 1

Introduction

An aspect for the usability of a mobile application is the availability and the support of tutorials. The more complex an application is, the more is the need of help for the user. In the past, a lot of research happened on usability, creating tutorials, teaching, learning with computer systems, and therefore, methodologies were examined.

Usability for mobile applications is similar to usability for general software on non-mobile devices. There are aspects which distinguish mobile from non-mobile, for example smaller screen sizes, the mobility, and the variety of applications.

Furthermore, we are adapting the information about usability on mobile devices and mobile applications. The purpose is to develop a tutorial according to the concepts found in literature and evaluate it with a usability test. The basic concepts for tutorial design, learnability, and educational factors are discussed. Moreover, these concepts are applied to the *Android* application *Pocket Code* which is developed at the *Technical University of Graz*. Throughout this work a design for a tutorial and an evaluation concept were created. Also the teaching purpose and the work on integrating tutorials are discussed. So the concepts of learning with mobile devices are presented and are set in context with the support of tutorials instead of human help.

After an introduction on *Android*, the *Catrobat* project, the used development methodologies, usability, theories for tutorials, and the usage in learning are discussed. Furthermore, different tutorials, as well as evaluation methods, are presented.

1.1 Introduction to Android

Android is the open source operating system developed by *Google* for mobile devices. The operating system is based on a *Linux* kernel. It is also a platform for creating mobile applications. All over the world, millions of mobile devices are equipped with an *Android* version. The *Android* applications are distributed over the *Google Play Store*. *Android* also offers the possibility to install custom made applications on the device. It is fast growing and the newest release is *Android* 4.4. [AndroidDevelopers, 2014]

Since 2008 several *Android* versions were released. In Figure 1.1, a diagram about the distribution of the used *Android* versions is presented (March 2014). The leading versions are *Jelly Bean* (*Android* 4.1) and *Gingerbread* (*Android* 2.3). [AndroidDevelopersDashboard, 2014]

The application can be optimized depending on the major share of users for the *Android* version. This concludes with the challenge in developing for *Android* devices to provide support for all relevant platforms and devices. [AndroidDevelopers, 2014]

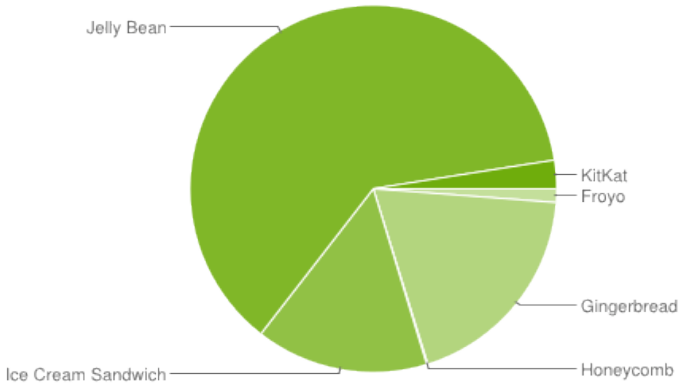


Figure 1.1: Android Distribution March 2014 [AndroidDevelopersDashboard, 2014].

1.1.1 Application Structure

Android applications are programmed in *Java*. The code is compiled with the *Android SDK Tools* and runs on devices with the *Android* operating systems. The applications run on the device in their own security sandbox. In the multi-user *Linux* system the applications are handled as different users. Every application gets a unique *Linux* user ID, and grants access to specific resources used by the application. *Android* takes care of the process management. So it starts a process for an application and stops it if it is not needed anymore. The code of the application runs in different processes. Furthermore, the application gets explicit permissions for the access to resources (e.g. the SD card, the contact list). The permissions and the activities are defined in the *AndroidManifest.xml*. Every application can start a component of another application. This is a unique property of the *Android* system design. An *Android* application consists of four different components, which have different purposes and lifecycles. These components are *Activities* (a single screen with a user interface, see Section 1.1.2), *Services* (run in the background and execute long running operations, see Section 1.1.3), *Content Providers* (manage shared application data, see Section 1.1.4), and *Broadcast Receivers* (respond to system wide announcements, e.g. battery is low). *Activity*, *Services* and *Broadcast Receivers* are started by an asynchronous message called *Intent*. This *Intent* binds the components to each other during runtime. The *Content Provider* is started by request from the *Content Resolver*. So all the transactions are called over this component. The components have to be declared in the *Android Manifest.xml*. Furthermore, permissions and minimum *API Level* are specified in the manifest. [AndroidFundamentals, 2014]

Every *Android* project has the same file structure. The project consists of folders and files described in Table 1.1. [AndroidProjectManagement, 2014]

Folder/Files	Description
src	In this folder, all source files for the application are stored.
bin	The folder contains all the compiled files including the *apk*-file.
gen	Here, the generated files of the *ADT* (*Android Development Tools*) can be found. For example the *R.java* file.
assets	This folder is empty by default. It can be used to store raw asset files. The files included are compiled into the *apk*-file. Further this files can be used with the help of the class *Asset Manager*.
res	In this folder, application resources (e.g. layouts, drawables, string values) are stored.
res/anim	The folder concludes *XML*-files which are compiled into animated objects.
res/color	*XML*-files describing colors are put into this folder.
res/drawable	The folder consists of bitmap files (e.g. png, jpg), 9-patch image files, and *XML*-files describing drawable objects.
res/layout	*XML*-files which are compiled into a screen layout are saved here.
res/menu	In this folder, there are *XML*-files which describe application menus.
res/raw	The objects in this folder are raw assets. The difference to the *assets* folder is the access to the files. The files are called with a resource identifier.
res/values	The *XML*-files included in this folder are compiled to different kinds of resources. The type of the resource is declared by the *XML*-element type and is placed in the *R.java* file.
res/xml	Different *XML*-files which define various application components are put here.
libs	Private libraries are put in this folder.
AndroidManifest.xml	This file defines the application and every component. Furthermore, the permissions, API levels, external libraries, and others are declared in this file.
project.properties	This file contains project specific settings, for example build targets.

Table 1.1: Android File Structure [AndroidProjectManagement, 2014].

These were only the basic components, but there are still further ones for an *Android* application. So in the following sections the components *Activities*, *Services*, and *Content Providers* are described in more detail.

1.1.2 Activities

An *Activity* is a basic part of an *Android* application and provides the user interface. An application consists of many different *Activities*. The *Main Activity* is the starting point of the application. This *Activity* has to be declared in the *AndroidManifest.xml* as the *Main Activity*. Out of the *Main Activity* other *Activities* are started. The *Activities* are managed like a stack. This means when an *Activity* is started, it gets on top of the stack. When the user is finished with the *Activity* it gets pushed, and the previous *Activity* is in the focus again. This procedure is handled over callback methods which can be overwritten in every *Activity*. The *Activity* receives the callback methods depending on the state change of the *Activity*. An *Activity* can have three states, *Resumed*, *Paused*, and *Stopped*. In the *Resumed* state the current *Activity* is in focus. If the *Activity* has the state *Paused*, another *Activity* is in focus of the user, and the paused *Activity* is still visible. The *Activity* is still attached to the *Window Manager*, and is only killed with very low memory. In the *Stopped* state, another *Activity* is in focus of the user and the stopped *Activity* is totally in the background. The *Activity* is not attached to the *Window Manager*, and is killed when there is a need of memory for another action. The state changes are part of the *Activity* lifecycle presented in Figure 1.2. Here, you can find the callback methods and the paths an *Activity* can take between states. The available callback methods are discussed in the following. [AndroidActivities, 2014]

onCreate() This method is called when the *Activity* is launched or the *Activity* is selected after it was detached from the *Window Manager*. [AndroidActivities, 2014]

onStart() This method is called after the *onCreate()* or the *onRestart()* method is finished, and the *Activity* is coming back to focus. So it becomes visible to the user. [AndroidActivities, 2014]

onResume() This method is called after the *onStart()* or the *onPause()* method is finished. After *onPause()* the paused *Activity* is resumed. [AndroidActivities, 2014]

onPause() This method is called when another *Activity* is created. The current *Activity* is paused, and its state is saved. [AndroidActivities, 2014]

onStop() This method is called when the *Activity* is stopped, and gets detached from the *Window Manager*. [AndroidActivities, 2014]

onDestroy() This method is called when the *Activity* gets finished by the system. [AndroidActivities, 2014]

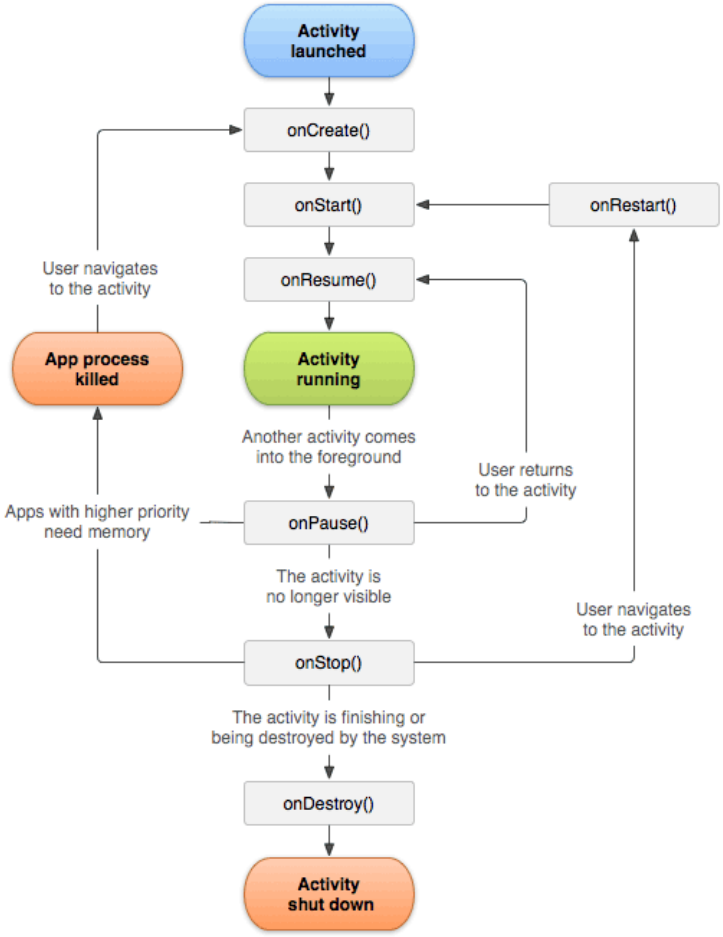


Figure 1.2: Android Activity Lifecycle [AndroidActivities, 2014].

1.1.3 Services

Another component of an *Android* application, introduced here, is *Services*. These components have to be started explicitly by other components. A *Service* in an *Android* application performs long running operations which run in the background, for example network transactions and playing music. This component does not provide a user interface. The *Service* has to be declared in the *Android Manifest*. A *Service* can have two states, *Started* and *Bound*. The *Started* state is reached if another component, for example an *Activity*, starts the *Service*. The *Service* runs even if the *Activity*, which started it, is destroyed. When the operation of the *Service* is done, it stops itself. The *Bound* state is reached if another component, for example an *Activity*, binds the *Service* to it. This allows the *Activity* to interact with the *Service*. The *Service* runs as long as it is connected to the other component. The lifecycle of an unbind and bind *Service* is presented in Figure 1.3. [AndroidServices, 2014]

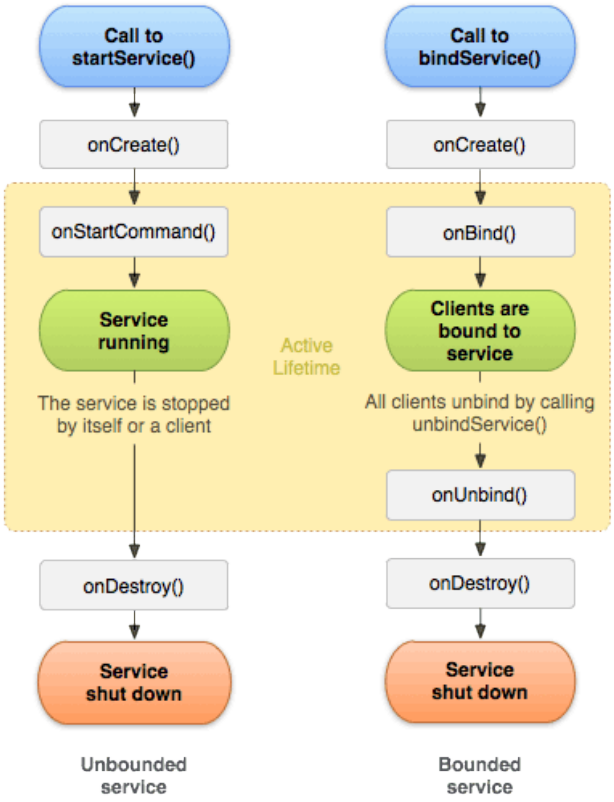


Figure 1.3: Android Service Lifecycle [AndroidServices, 2014].

The *Service* is handled over callback methods which are discussed in the following paragraphs. [AndroidServices, 2014]

onCreate() This method is called when *startService()* or *bindServices()* methods are executed by another component to create the *Service*. [AndroidServices, 2014]

onStartCommand() This method is called after the *onCreate()* method is finished, and the *Service* is started. The *onCreate()* method is triggered by the *startService()* method of the calling component. [AndroidServices, 2014]

onBind() This method is called after the *onCreate()* is finished, and connects the client to the *Service*. The *onCreate()* method is triggered by the *bindService()* method of the calling component. [AndroidServices, 2014]

onUnbind() This method is called when a client gets disconnected from the *Service*. [AndroidServices, 2014]

onDestroy() This method is called when the *Service* is stopped by itself or a client. [AndroidServices, 2014]

1.1.4 Content Providers

Content Providers work as the interface between data and the application. Furthermore, *Content Providers* manage the access of shared data to separate the data from the code. The data in one process is connected to running code in another process. The data is presented as tables in the *Content Provider* similar to a relational database. With the help of a *Content Resolver*, it is possible to query the data in the *Content Provider*. In an application, other components (e.g. *Activities*) always have read and write access to the *Content Provider*. An access from a different application is by default not allowed. Nevertheless, it is possible to define a permission to get access to the *Content Provider*. Furthermore, a *Content Provider* can be created from scratch for every purpose needed. [AndroidContentProviders, 2014]

1.1.5 Technical Approach for Creating a Tutorial

Here, we introduce one approach which is used to create a tutorial. This approach is used for the examples in *Pocket Code* presented later in this thesis. The basic idea to create a tutorial for an *Android* application is to create an overlay on top of the actual application. The overlay is invisible to the user. On this overlay, graphical presentations are drawn, for example, a tutor and text explain the basic concepts. The overlay is created by extending the *Android* class *SurfaceView*, and implements *SurfaceHolder.Callback*. The *SurfaceView* provides a drawing view which is integrated in the view hierarchy.

The benefit in using this class is the control of the format of the view (e.g. changing the size of the view). The purpose of the *SurfaceHolder* interface is to get access to the layer itself. By integrating the interface in a class, there is a need to implement the following methods of the *SurfaceHolder* interface:

- **surfaceChanged()**: This method is called after the surface has changed.
- **surfaceCreated()**: This method is called after the surface is created.
- **surfaceDestroyed()**: This method is called before the surface is destroyed.

By default, the surface is behind the window holding the *SurfaceView*. Here, we want to have the *SurfaceView* on top of the *Activity* which is holding the *SurfaceView*. The Surface is z-ordered, and can be set on top with the method *setZOrderOnTop()*.

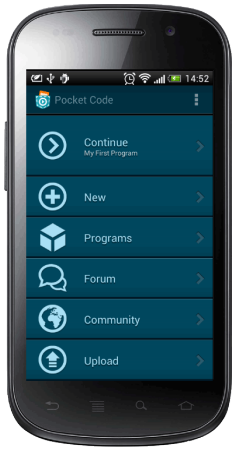
The *SurfaceView* has to be filled with graphics and instructions. So it is necessary to draw on the view. Therefore, the method *onDraw()* from the class *View* has to be overridden. It is also intended to provide the actual functionality of the application. So to provide the clickability, for example a button, the method *dispatchTouchEvent()* from the class *View* has to be overridden.

This is the basic part behind the idea of creating a tutorial for an *Android* application used for this thesis. Yet another challenging part is to automatically get the positions for the objects on the surface and give them their own logic. Further in this thesis, designs of prototypes and approaches for the implementation, which use this concept, are presented.

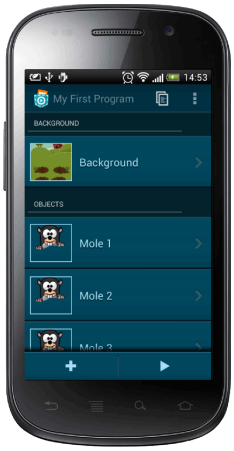
1.2 Introduction to the Catrobat Project

The Catrobat Project [Catrobat, 2014] is a FOSS (free and open source software) project started at the *Technical University of Graz*. It contains a visual programming language called *Catrobat*. The programming language is optimized for mobile devices. *Catrobat* and the software developed are inspired by the *Scratch* [Scratch, 2014] project of the *Lifelong Kindergarten Group* at the *MIT Media Lab*.

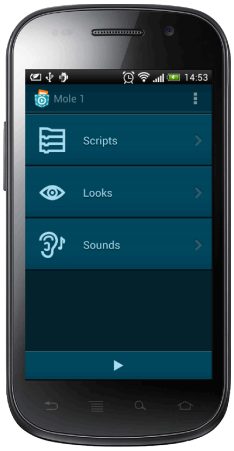
The *Catrobat* team is working on a mobile application for the *Catrobat* language on the *Android* operating system among other platforms (e.g. *iOS*, *WindowsPhone*). The *Android* application is called *Pocket Code*. *Pocket Code* works on mobile phones with *Android 2.3* and higher. Here, we will focus on the *Android* application. The application is still in development and it is released in the *Google Play Store*. The purpose of the project is to give primary kids and teenagers an easy possibility to learn programming without any previous knowledge about programming. The main target group of the released version are male teenagers. Versions for other target groups are planned in the future. The project team is working on designs for other target groups. The approach for teaching the basic programming concepts is simple. The programming statements are represented in block form. The available blocks are designed in a *"Lego"* brick style and the users can put them together to an executable program. The *Catrobat* programs can be created, and are directly executed in *Pocket Code*. The programs can be uploaded, and are published to the community website, where other users are able to download and reuse the programs. The workflow of *Pocket Code* is complex, and new users have to find their way to the first program by exploration. The *Catrobat* team is working on different approaches to make the access to *Pocket Code* easier for new users. In the following the main screens of the application is shown and briefly explained.



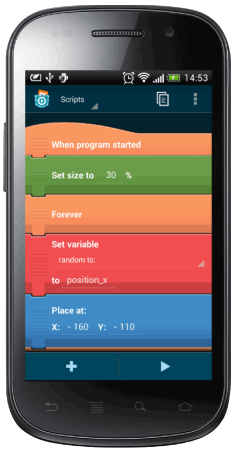
The start screen of the *Android* application *Pocket Code* is shown.



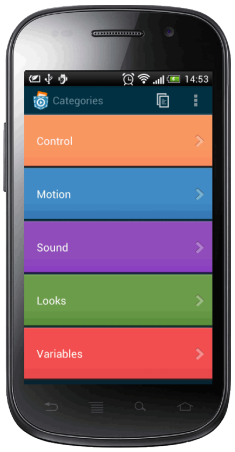
The overview of the project with all objects is here visible.



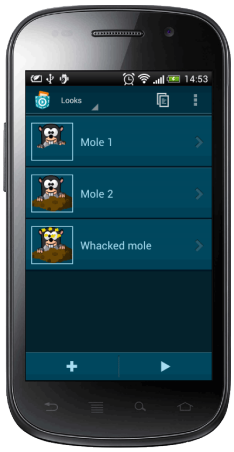
Menu to create scripts, looks, and sounds for an object is presented.



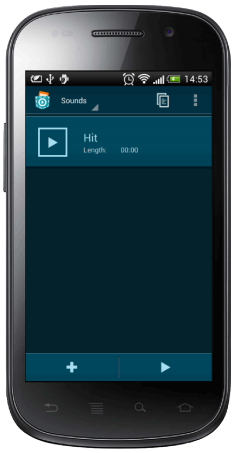
View to combine bricks together to a script.



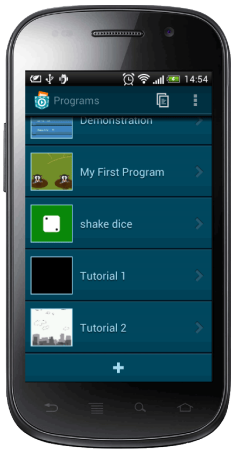
Overview of the brick categories (control, motion, sound, looks, and variables).



Overview of the looks of an object. Also new looks can be created here.



List of the sounds of an object. Furthermore, new sounds can be added.



Overview of all projects created.

1.3 Introduction to Development Methodologies

In the *Catrobat* team, as well as during developing for this thesis, different software development methodologies were used. The methodologies *Extreme Programming* (Section 1.3.1), *Test Driven Development* (Section 1.3.2), and *Clean Code* (Section 1.3.3) are discussed in detail in the following sections.

1.3.1 Extreme Programming

The development methodology *Extreme Programming* is an agile method. It was created to fit the needs of small teams in the process of software development. It works on the assumption, the developers do not know all the requirements. [Beck, 2000]

New functions are developed, integrated, and tested iteratively. Therefore, a first functioning prototype is available very fast. Extreme programming consists of values, principles, and practices. These points are explained in detail in the following. [Beck and Andreas, 2004]

Values There are four basic values in *Extreme Programming*, communication, simplicity, feedback, and courage. The first value is communication. The development team has to communicate with each other constantly. So there is also a constant information exchange between the developers and the customer. The second value is simplicity. In each iteration the team works on fulfilling specific requirements. So this value covers the implementation of the simplest solution for the current problem. The next value is feedback. It covers getting concrete feedback to a task. It also supports realizing the wishes of the customer and furthermore, creating what the customer wants. The last value is courage. The developers should have the courage to communicate if something is not realizable in the current iteration. [Beck, 2000]

Principles *Extreme Programming* contains several principles which are the connection between the values mentioned before and the practices explained further on. These principles are explained in Table 1.2. [Beck and Andreas, 2004]

Principle	Description
Humanity	Software is developed by humans. Therefore, *Extreme Programming* supports a pleasant environment for the developers.
Economics	A software has to be financial efficient.
Mutual Benefit	Also the developed software has to have benefits for developers and customers.
Improvement	During the development process, code is changed and improved several times. The first solution is never optimal.
Diversity	Different skills in the development team offer different approaches. A variety of opinions are welcome for a better product.
Reflection	Constant reflection of the approach leads to a better solution.
Flow	The software should be runnable at any time of the process.
Opportunity	Difficulties in the development process should be taken as an opportunity for further improvements.
Redundancy	Any redundancies should be avoided to ensure a good product.
Failure	No failures should happen after releasing the product.
Quality	Quality has to be provided at any time of the development process.

Principle	Description
Baby Steps	With small and quick steps the team can be flexible. So the developers can adapt to any circumstances very quickly.
Accepted Responsibility	The developers have to take their responsibilities actively. So just allocating randomly responsibilities should be avoided.

Table 1.2: *Extreme Programming* Principles [Beck and Andreas, 2004].

Practices The development process of *Extreme Programming* contains several practices. These practices are explained in detail in Table 1.3. [Beck, 2000]

Practice	Description
Planning Game	It is a process of planning the next release. Therefore, the costs for the implementation are estimated. In the planning process developers and customers are involved.
Small Releases	The releases should be as small as possible. So the time for an iteration is decreased.
Metaphor	When creating user stories, a common vocabulary is needed to avoid misunderstandings between developers and customers. Therefore, a metaphor is used which both groups understand.
Simple Design	The simplest solution is implemented. Preparations for a future functionality are avoided. So everything which is implemented, is wanted of the customer.
Testing	In every iteration, tests are written for the current implementation. An approach to write tests is the *Test Driven Development*. This approach is discussed in detail in Section 1.3.2.
Refactoring	In each iteration of the developing process, the code is refactored. The code does not have to be perfect at the beginning. So the software gets always improved.
Pair Programming	The developers program always in pairs. They share one computer. So one developer is actually coding and the other one is thinking along. The roles are changed regularly. The knowledge exchange is improved with this method.
Collective Ownership	Activities are assigned to the whole team, not to a single person. Furthermore, there is no single person who knows everything. So everyone in the team knows everything.
Continuous Integration	The single components are integrated continuously to a functioning system in small time intervals. So errors are uncovered early in the process.

Practice	Description
40 Hour Week	Overtimes should be avoided. So the developers do not get frustrated with their work.
On Site Customer	The customer has direct impact on the goal of an iteration. The user stories are created with the customer. Therefore, the customer has to be available at all time.
Coding Standards	The development team is following a specified coding standard. So the developers can be used in different areas of the process.

Table 1.3: *Extreme Programming* Practices [Beck, 2000].

1.3.2 Test Driven Development

Test Driven Development is a software developing method which is often combined with agile development. A benefit of writing the tests first is that developers do not write the tests after the programming under time pressure. There are tests for code which the developers have no knowledge about. Also more tests are written with this approach compared to the number of tests written after the development. Furthermore, the tests can help building the design of the software. So the developers have to think about the users when writing tests. The process of writing tests first happens in small iterations. First step is to write a test case for the functionality which is implemented next. The first run has to fail because the tested code is not written yet. In the next step the code for the new functionality is written. In the last step the code is refactored, for example, deleting redundancies and inserting abstractions. This step will be repeated for every new code fragment written. The use of *Test Driven Development* needs tools for continuous integration, build automation, test development, and test automation. [Beck, 2002]

1.3.3 Clean Code

The term *Clean Code* defines a software methodology. The methodology describes a way to write code which is easily understandable for other programmers. So the written software is also easily maintainable. Therefore, adding or changing functionality can be done in a shorter amount of time. An overall rule of this methodology is the so called *Boy Scout Rule*. This rule simply says to check in cleaner code than you checked out. For the implementation of *Clean Code*, several more principles have to be followed. In the following, some of the principles are explained. [Martin, 2009]

Meaningful Names The first principle is the usage of meaningful names. This means the names for variables, methods, files, and so on get a name which describes the intention of the described element. Therefore, the naming process is a challenging part. It needs descriptive skills which is not easy for everyone to do. With practice and keeping this principle in mind, better code can be produced. [Martin, 2009]

Functions This principle handles the usage of functions. A big part is to make the functions small and they should contain only one thing to execute. Furthermore, the usage of functions support the developer in not repeating code. Therefore, the functions are easier to name and to read. The process of creating small functions according to this principle is iterative by refining, renaming, and refactoring code. [Martin, 2009]

Comments The next principle is about the usage of comments. Comments are an easy way to describe the purpose of the code. Unfortunately, this is hard to maintain. Only add comments to the code if really necessary (e.g. copyright information, to do information, and warnings of consequences). With the principles mentioned before, the code should be explaining itself. So comments can be a sign of bad code. So before writing a comment, check if there is a way to rewrite the code. [Martin, 2009]

Formatting The next principle handles the formatting of the code. The purpose of this principle is to support the readability and furthermore, the maintainability of the written code. With the right use of indention and new lines the reading flow for other developers is provided. Therefore, a coding standard for the application is useful for the developers to have a common formatting. [Martin, 2009]

Objects and Data Structures Another principle is about objects and data structures. The purpose is to make aware the importance of hiding data or behavior. The approach is to hide the data and expose the behavior. Therefore, data structures can be extended with new functions easily but additional data structures are hard to realize. In case it is intended to create new data structures objects should be preferred. [Martin, 2009]

Error Handling The next principle contains the error handling. Errors and exceptions can always occur during runtime. Therefore, it is important to include error handling. On the other hand, the code should not be dominated by error handling. If it is hard to see the purpose of a code because of an overload of try-catch sequences, a refactoring of the code is needed. The code should be robust and clean at the same time. The insertion of exceptions in the code can provide it and furthermore, a detailed error message helps finding the source of the error. [Martin, 2009]

Boundaries This principle handles the boundaries to other software. In case third party libraries or subsystems from other developers are integrated into your software, the boundaries between the components have to be kept clean. The boundaries are crucial to the software because every change in the code can take affect on the boundaries. Therefore, there should be just a few positions in the code which have direct access to the libraries. Also the maintenance is easier with this approach. [Martin, 2009]

Unit tests The next principle is about testing. The test should also be kept clean. Therefore, it is supported by the methods of *Test Driven Development* presented before (Section 1.3.2). Tests support the ability of flexibility, maintainability, and reusability of the code. [Martin, 2009]

Classes Another principle for clean code is the creation of classes. Classes should be kept small. This means a class has few responsibilities. So there is only one reason to change. Furthermore, the classes get more cohesive. So with providing a high value of cohesion the classes are getting small. This behavior is supported by the help of instances and abstract classes. [Martin, 2009]

1.4 Introduction to Usability

Usability is an important factor for software and describes the quality of use. Usability engineering is an iterative process to provide usability for software. Therefore, the term usability was defined by the *International Organization for Standardization* (ISO) in 1998. The ISO standard 9241-11 describes the extend to which a product can be used by specified users to achieve specified goals in a specified context using three aspects, effectiveness, efficiency, and satisfaction. The effectiveness is defined as tasks which are completed by the user without any errors. The relation between the resources needed and the completed tasks of the user is called efficiency. The term satisfaction means positive and negative feedback relating to the use of the software. [ISO9241-11, 1998]

This definition is very general and can be implemented on different fields of use, for example software and mobile devices. Over the years different definitions were created. One well known definition was described in [Nielsen, 1994].

This definition uses the system acceptability as a basis for the definition. As shown in Figure 1.4, system acceptability is divided into several attributes. In this context, usability is one attribute which is divided into a total of six further ones, including the attributes of the ISO standard. These attributes are effectiveness, efficiency, and satisfaction as already explained before which are now extended with *learnability*, *memorability*, and *errors*. Learnability describes the ease of use for new users of a system. An easy memorable workflow of tasks is the attribute memorability. The rate of appearing errors is summarized in the term errors as the last attribute. [Andrews, 2014]

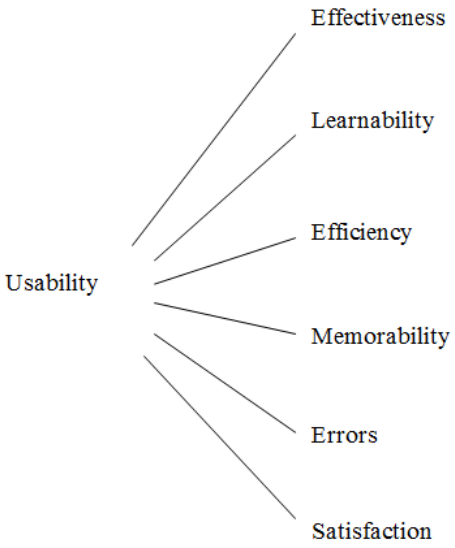


Figure 1.4: Usability Attributes [Nielsen, 1994].

Usability is not only applicable on desktop applications. It is also important on mobile devices. Therefore, the defined usability principles have to be adjusted to the special requirements of a mobile device. The main differences in the mobile context are the connectivity, the screen size, the display resolution, the processing capability, and the data entry methods. First, the mobile context is here defined as the context in which a mobile device is used. This context can vary because of the mobility provided. So the location and the surrounding environment are providing potential distractions for the users. This distractions are hard to include into usability evaluations, but have effect on the usability. The next difference is the connectivity. For mobile applications the availability of an internet connection has to be taken into account. A wireless or a mobile network connection is in most cases available, but the strength of the signal and the speed can be crucial for usability in applications which need an internet connection. Therefore, it is an influence on the usability. Furthermore, there are many mobile devices with different screen sizes on the market. The usability of an application on a tablet with ten inches screen is different to the usability on a smaller screen of a smart phone. The resolution of mobile devices has a variety from low (e.g. 640*480 pixels) to a high pixel count (e.g. 2560*1600). This factor can influence the presentation of the application and so the usability can be different. The processing power of mobile devices is getting close to classical computers, but still needs consideration. A crucial point is the direct impact of the processing on the battery. So for a good usability, this factors have to be taken into account. Mobile devices also provide different input possibilities than desktop computers. Therefore, design decisions, which may reduce input speed and increase errors (e.g. small buttons), are important for applications, which get input data from the user, as well as impact the usability. [Zhang and Adipat, 2005]

1.5 Overview

Previously, introductions were given about the basic concepts included in this thesis. Therefore, an overview of the *Android* developing platform, the *Catrobat Project*, the development methods used, and usability is given.

The basic elements of an *Android* application are introduced. These elements are *Activities*, *Services*, and *Content Providers* and were discussed in detail. Therefore, the sections include the lifecycle and the routine of creating, starting, pausing and stopping these components. Furthermore, the *Catrobat Project* and its visual programming language *Catrobat* was introduced. In this context, the *Android* application developed in the *Catrobat Project* is presented. The application is called *Pocket Code*. It is referenced as a further example in the thesis and is the basis for the created tutorial. The introduction chapter includes summaries to the development methodologies used in the *Catrobat Project* as well as for the implementation for the created prototypes. So the concepts of *Extreme Programming*, *Test Driven Development*, and *Clean Code* are explained. Furthermore, usability was defined. Besides, the origin of the term and the aspects included were discussed. These aspects are effectiveness, learnability, efficiency, memorability, errors, and satisfaction.

In the next chapter, theories for creating a tutorial are discussed. Furthermore, the basic challenges in the creation of a tutorial are presented. This includes best practices in creating a story and tutors. Also guidelines and ways to engage the users to the application are presented. Besides, the inspirational resources for a created tutorial are explained with their advantages and disadvantages (Scratch, Wario Ware D.I.Y., Alice, and Kodu).

Also the purpose in the field of education is presented and the principles of *Gamification* are defined additionally. The term is defined and later in the section the basic game elements used to implement *Gamification* (fantasy, goals, feedback, guidance, progressive disclosure, time pressure, rewards, punishments, and stimuli) are discussed. Another theory which is presented is mobile learning. The term is defined and further learning principles, adaptive learning, and serious games are set in context with mobile learning.

In the last section details about the prototypes for *Pocket Code* are presented in detail and the concepts of the implementations are shown. Furthermore, this thesis includes the presentation of prototype designs developed for the *Android* application *Pocket Code*. The purpose for the approaches of a tooltip system and a guided tutorial are discussed.

In the last chapter, evaluation methods and metrics are presented and are put into practical context. Therefore, the evaluation methodologies of *Heuristic Evaluation*, *Cognitive Walkthrough*, *System Usability Scale*, *Emocards*, *A/B Tests*, and *Thinking Aloud Method* are discussed in detail and finally different metrics are defined (e.g. error rate, completion time, performance, and success rate). In the last section a proposal for a usability test is put together.

Chapter 2

Theories for a Tutorial

An application should have a simple structure and good usability. However, a simple structure is not always possible because of a complex content. In most cases, a complex context results in a complicated workflow within the application. Therefore, the inclusion of a tutorial is advantageous to resolve the complexity for the user. There exists a lot of software with little or no help. Of course, there are applications which the users can instantly use. Games are examples for this variety of complexity. There are several classic games without any instructions, for example popular games are *Tetris*, *Pacman* and *Super Mario*. [Andersen et al., 2012]

With huge progress in technology, software and games got more complex. The use of software is not that simple anymore. Depending on the content of the application the workflow can get very complicated. So the first contact of the user with the application can get frustrating. A tutorial can support the user in getting to know the structure, and teach the effective use of the application. The outcome is an easier handling of the software than without any further instructions. Besides, the usability is increased. Further in this chapter, the different challenges in creating a tutorial (Section 2.1), the inspiration for a tutorial in *Pocket Code* (Section 2.3), the educational purpose of tutorials (Section 2.2), the term *Gamification* (Section 2.4), mobile learning (Section 2.5), and the approach of the tutorial in *Pocket Code* (Section 2.6) will be pointed out and discussed.

2.1 Challenges

In the field of mobile applications, tutorials are not very common. The usefulness of a tutorial depends on the complexity of the application. The user gets frustrated easily and stops using the application. An application like *Pocket Code* is very complex and without any knowledge hard to use. The variety of alternative mobile applications is a reason for engaging the user to a specific application. Therefore, it is important to have a tutorial to show the user how it can be used, so that the user will not consider alternatives. It is also a tool to motivate the user to learn more about it. The instant feedback and risk-free environment, which can be provided in a tutorial, invite the user to explore and experiment with the application. In addition, it stimulates the curiosity, supports the learning experience, and perseverance. Already in the early stages of software development, the topic of tutorials was explored. In [Vanderlinden et al., 1988], the increase of learnability with the presence of a tutorial was discovered. The ideas were based on classical computer systems. These basic concepts are also adaptable for mobile applications.

The availability of tutorials is a present topic in software development, for example, in games tutorials are a tool to learn the gameplay. The frustration of the user in finding out how the game works is minimized. Also the engagement and retention for the game is established. The concept is adaptive to other software products like mobile applications. The special features and limitations for mobile devices have to be taken into account. [Andersen et al., 2012] [Kirriemuir, 2002]

In the following sections, the basic challenges of creating stories (Section 2.1.1), tutors (Section 2.1.2), the engagement of the users to the software (Section 2.1.3), and guidelines for creating tutorials in applications (Section 2.1.4) are discussed in detail.

2.1.1 Story Development

One challenging part is to create a story and a sequence of steps to motivate the user and to avoid boredom. There are no explicit design rules, so the developers have to base their ideas on intuition, experience, and similar software to extract adaptable design guidelines. The need for help within an application is based on questions of the user which can appear during the interaction with the user interface. Examples for these questions are *"What can I do with this tool?"* and *"What does this mean?"*. With the identification of such questions, a help system can be developed. Further, the answers should be included in the application, either directly in the systems (e.g. as a navigation area to answer questions how to get to a specific action) or within a help system (e.g. a tutorial where the questions are asked and answered by an animated tutor). [Silveira et al., 2001]

For creating documentation, help, and tutorials, the used material can be viewed as material for answering questions of the users which may appear when using the software. The questions can be summarized in categories. The categories found in Table 2.1 are relevant for identifying the content of a tutorial, independent from the type of the tutorial. [Baecker et al., 1991]

Category	**Description**
Identification	This category contains the explanation of the current object (*"What is this?"*). So users know the term for an object within the application. For example the tutorial explains the concept of a brick in *Pocket Code*.
Transition	This category includes an easily memorable path to the current task (*"Where have I just come from?"*). So the users can reproduce the task on their own after finishing the tutorial. Especially on mobile devices, a task includes accessing different screens within the workflow. For example, in *Pocket Code*, it would be adding a brick to the script of a project.
Orientation	This category describes a navigation for the user and where in the application the current task is available (*"Where am I?"*). So the position of a specific action can be retraced by the user.
Choice	This category consists of an overview of the available options (*"What can I do now?"*). The options are clearly represented in the user interface.

Category	Description
Demonstration	This category contains the explanation of the purpose of a specific tool (*"What can I do with this?"*). So the purpose of a tool is explicitly defined. For example the tutors show the usage of a specific tool, like how to add a brick to a *Pocket Code* project.
Explanation	This category includes detailed illustrations of a specific action (*"How do I do this?"*). The purpose of an action is presented in practice to the user. For example a tutor explains the concept of a brick to the user.
Feedback	This category describes a response from the system about the executed actions (*"What is happening?"*). So the user is always informed about progress, errors and tasks. For example the user gets verbal feedback of a tutor.
History	This category consists of a possibility to track the steps to the current position in the system (*"What have I done?"*). For example, in case of an erroneous behavior, the user knows which sequence of actions it caused.
Interpretation	This category explains the reason for the action (*"Why did that happen?"*). So the user can follow the purpose of the current task.
Guidance	This category includes illustrations of the following steps the user should do (*"What should I do now?"*). So the user does not get stuck in performing a task and gets help when it is needed. For example the a tutor guides the user step by step through the application.

Table 2.1: Question Categories [Baecker et al., 1991].

These categories support the process of creating help components in a tutorial and how they are structured. The most important points are covered by giving answers to the previous question categories in Table 2.1. There are also issues which concern the field a help component should be sensitive about.

Four aspects for tutorials are relevant for the design, presence of tutorials, context sensitivity, freedom of the user, and availability of additional help. With these aspects a comparison between applications can be achieved. These aspects are discussed in the following sections. [Andersen et al., 2012]

Presence of Tutorials

One characteristic factor for tutorials is the availability, whether there is a tutorial implemented or not. Of course this factor is obvious, but it has to be considered for testing the user interface of a software. The presence of a tutorial is important because it supports the ability to engage users to the application. The affect on a software can be simply tested by comparing these two versions (with and without tutorial) of the software with each other, for example by executing an *A/B Test*. [Andersen et al., 2012]

Context Sensitivity

The aspect of context sensitivity means, the help of the tutorial is presented to the user when it is needed within the context of the application. So the information is always related to the current visible state of the user interface. Therefore, the instructions are more effective than presented out of the context. Furthermore, the content of the tutorial should be independent from the user. This means that the same help is always displayed for every user. Also the context should be sensitive about what happens in the user interface. The interface reacts on some input from the users, for example, a hint text appears when hovering over a button. In case of creating user profiles within the application, the collected user data should be taken into account. Depending on the profile the help content varies. For example the users preferences and history of interaction, like different designs for boys and girls. Other influences are the tasks executed by the user. The system creates the help content depending on the current task (e.g. with a wizard). The challenge is the extraction of the current task in a complex system. [Andersen et al., 2012] [Silveira et al., 2001]

Freedom of the User

The idea behind the aspect of freedom of the user in the application is based on the concept of letting the users get to know the application mechanics on their own. So the users can also practice their abilities in a save environment. The degree of restricting the user to specific actions is relevant in order to avoid possibility of the user to make major mistakes when following a tutorial. A tutorial guides the user through a set of tasks. In these tasks the restrictions can vary, for example, in a guided tutorial the user can get exact instructions at the beginning and gets more freedom over time. This means, at the beginning, the tutorial blocks other actions than the intended action from the user. So the basic mechanisms are shown and can be practiced without the fear of failing. Moreover, it also supports the engagement of the user to the application. [Andersen et al., 2012]

Availability of Help

This aspect handles the type of access to help. The additional help can be accessed on demand. In this approach a tutorial is not automatically started at the first start of software. Instead it has to be activated explicitly by the user. The appearance of the help can be different. For example the user presses a button and gets information for the specific situation instantly. Another example would be to open a user manual, documentation, or a tutorial sequence to get help to solve a task. This aspect supports the retention of the user with the application. [Andersen et al., 2012]

2.1.2 Tutor Development

In a tutorial, the task of informing and guiding the user can be done by a tutor. The role of the tutor has to be designed as well. In tutoring systems, a tutor can be used to teach and guide users through the application. A tutor in a tutorial can play different roles, like an expert, a motivator or a mentor. The main difference between these roles are found in the presentation in the tutorial. The presentation is split up into image, voice, animation, and affection. [Baylor and Kim, 2005]

Also the learning strategies have to be considered. So by implementing an one on one learning approach, the expert tutor is created. Therefore, the tutor is simulated as an intelligent agent who can provide knowledge. An expert tutor gives the user direct hints and gestures for fulfilling specific tasks. The tutor is an authoritative figure. The voice is monotone, the speech is formal, and is detached from emotion. The tutor is designed similarly to a teacher. The main task for the tutor is to give information to the user. Another learning strategy is the approach of learning with a co-learner. This approach is build on the idea that knowledge results from a building process. Therefore, the task of the tutor is to motivate and support the user in the learning process. A tutor, who is used as a motivator, has attributes which are similarly to the target users. So it is easier for the user to connect with the tutor than with a more distant tutor like the expert version. Therefore, the help of the tutors is expressed enthusiastically and is mixed with colloquial sentences. So a conversation is simulated between the tutor and the user. Furthermore, emotions, like frustration, confusion and enjoyment, are communicated. The main task for the tutor is the motivation and encouragement of the user to further process with the application. A tutor, used as a mentor, guides the user through the application instead of direct instructions. The tutor tries to challenge the user and therefore, encourage the user to find a solution for a task. The tutor is not authoritarian and works collaboratively with the user. The design of the tutor is less formal than the expert tutor but more mature than the motivator. The main task for the tutor is in between of the expert and the motivator tutor, it is the information distribution and encouragement of the user. [Baylor and Kim, 2005] [Aimeur and Frasson, 1996]

2.1.3 Engagement

Another challenge for a tutorial is to engage users to a specific application. Current technology of mobile phones is used to solve this challenge. In this case, a mobile application also brings the advantage to be available anytime and anywhere. It is also a benefit to provide a whole environment for the games. [Mitchell and Smith, 2004]

For example in *Pocket Code* you have the possibility to create and deploy your own program and it is also possible to download and play games from other users. Some influential factors of engagement are summarized in Table 2.2. The implementation of these factors into software can increase the level of engagement. Furthermore, in tutorials this can be realized easily. [Prensky, 2001]

Influence Factor	**Description**
Enjoyment and Pleasure	It is supported with the fun the users have by using the software. Fun is used as a motivator. So with the amusement with the usage of an application the users are ambitious to solve specific tasks. Besides, the user is relaxed and the learning experience is not perceived as such.
Passionate Involvement	This factor covers the characteristic of play in software. The influence on the engagement is based on the concept of learning in a playable way. Furthermore, the involvement of the user is supported and the user wants to go on and learn more things.

Influence Factor	Description
Structure	It is created by giving the user rules within the tutorial. These basic set of rules defines the limits, in which the user acts. So different users have to take the same sequence of actions to reach a specific goal. The rules make the experiences in the application fair and still challenging. Without rules users can do whatever they like and get bored very fast.
Motivation	It is supported by giving users goals to reach. Without a goal users loose their motivation in using the application because there is nothing to achieve.
Feedback	The learning experience is provided with feedback for the user. With goals the motivation is supported and by getting feedback the users learn what they achieved and if they reached the goal. It is also important if some circumstances are changing. Furthermore, the software gives positive and negative feedback about the actions.
Gratification	The users gets something for winning or defeating for example an opponent. The reward for the user is used as a motivator to go further. The rewards can also be applicable to give the users confidence in their actions. So they know the right path is taken to reach the next goal.
Creativity	It is supported by creating their own problem solving tactic. There is not one single strategy to go through the application. Therefore, the user has to find their own way to succeed the challenges.

Table 2.2: Influence Factors for Engagement in Software [Prensky, 2001].

2.1.4 Guidelines

Beyond the aspects to engage the user to an application, also fulfilling guidelines for creating a tutorial can support the developing process. In [Grabler et al., 2009], the following guidelines for creating tutorials are suggested:

- **Step by Step:** When learning a new task, users tend to split up the task by their own judgment into a sequence of simple steps. For example a sequence of instructions in a tutorial should be used to teach a user one specific task.
- **Succinct:** The steps should be as short as possible. Every step, which is not necessary, can be eliminated. The steps should not be repeated to avoid boredom of the user.
- **Annotations:** Graphical tips like arrows and highlighted positions support the understanding of the user of the instruction.
- **Text and images:** The combination of text and images help to increase the effect of the instructions on the users. It is more effective than just text or just images.

- **Grid-based layout:** The layout of the tutorial should clearly define the sequence of steps. Furthermore, images and text should be placed near to the described objects in a grid-based layout.

The design of the tutorial has a big influence on user acceptance of the application. The presentation of the tutorial supports the application in guiding the user to learn more about it. Unfortunately, it is also possible to achieve the contrary effect.

In [Adams, 2011], the following tips for creating an instruction set are presented. Besides, the user gets engaged to the application. The first tip is *"Do not force the player to take the tutorial!"*. The idea behind this approach is not to bore the user with things they already know. In case the user is already familiar with the concept of the application there is no need to bother them with a mandatory tutorial. The most important part is to give the user the opportunity to stop or skip the tutorial. So there are no annoying instructions for users at the beginning of the application. The next tip for creating a good tutorial is *"Do not make the player read a lot!"*. This concept is based on the interaction between the tutorial and the users. It is important that users do not have to read a long text to get the information needed. It is also not helpful to let users navigate through a various number of screens with only text on it. Of course, without any instructions a tutorial does not make any sense. The best way is to keep the instructions short and let the user try out the instructions. The next advice for a good tutorial is *"Provide a good description of buttons and menu items!"*. This advice is related to describing buttons or menu items with terms which are clearly to the user. A negative example would be a reference to a *Send* button but there is no button which is labeled with the term in the application. If using icons for buttons the concept behind the buttons has to be introduced, so the users understand the function of the button. Furthermore, it is essential to always describe menu items with the whole path to it. For example click on menu X and the submenu Y. Therefore, misunderstandings are not possible. The next advice is *"Do not leave steps out!"*. The idea is to avoid skipping instructions where they are needed. Of course, if a sequence of tasks was already mentioned before in the tutorial there is no need to explain the same things again. The important part is to give all necessary information to the users, so they do not get stuck at some point. However, how detailed a tutorial is, depends on the complexity of the application which is described. When leaving some steps out, create an additional possibility to give information to the user. Even for information the users should already know. For example after finishing the tutorial, the program should provide help buttons to the users. So there is a way to give short tips how to proceed in the application. The next tip for creating a good tutorial is *"Do not punish the user for mistakes!"*. This concept is based on the idea to reset the tutorial after users make a mistake. So they have to go through a lot of instructions all over again. In this case, they would be annoyed and the frustration level rises. Therefore, do not make the punishment too hard. In a tutorial, the users should be in a save environment. In the best case, the users are set back not very far in the sequence of the tutorial and they get information about what was wrong. The last tip for a good tutorial is *"Let the users abort the tutorial!"*. The idea behind this tip is to always give the user the opportunity to stop the tutorial at any time. Also provide a possibility to skip some lessons if there are more than one task to fulfill in the tutorial. [Adams, 2011]

The guidelines are very general and depend on the specific application because it is hard to find a clear pattern for every application. Also the length of a tutorial is not clearly defined, but it depends on the purpose of the application as well as on the complexity of the material which is brought to the user.

2.2 Educational Purposes

In the educational area, different approaches to bring learning material to the users, are tried out. The attempt of using game elements in combination with mobile devices and the differences to the common teaching styles are discussed further on.

The traditional teaching style with one person (e.g. teacher) telling everything to a group of other people (e.g. students) is not effective. The reason is that every user has a different speed of learning. So with a traditional learning style just a few people are reached with the learned material. The rest is either challenged or bored. In the best case, the teaching style is personalized for every user depending on their learning speed. The personal needs of every person can be met and the success in learning is maximized. The big problem with personalized learning is the demand of manpower. A teaching style in between these two constraints has to be found. Before designing and creating a tutorial, the behavior of the users have to be analyzed. Therefore, the learning process is divided into three phases. In the first phase, the task has to be explained in an appropriate form. So the user understands it easily, remembers the sequence and rehearses it by performing tasks. The user can learn by trial and error in a secure environment of the tutorial. In the second phase, the initial errors are detected and the users learn the right procedure to perform the tasks. In the third phase, the users get more freedom and become familiar with the task. The time a user needs for each phase is different from person to person. [Harrison, 1995]

Besides, the learning process and the theoretical background in learning, there have been some interesting findings in the use of mobile devices discussed. For example the *One Laptop Per Child Organization* made an interesting experiment with kids in Ethiopia. They gave tablets to children and observed their behavior with this unknown technology. The surprising result was that the children learned by themselves how to use them. [Talbot, 2012]

With the usage of mobile devices and applications, a new approach for teaching can be evolved. In literature, educational games were created and analyzed based on the learning effect on students. The lessons learned out of educational games can be adapted for applications on mobile devices. Even with a design according to all common design rules it is hard to create software which meets the needs of different persons. In school teachers can react on the preferences of their students, but in software this is a challenging part to develop. Therefore, the concepts of *Gamification* and mobile learning are presented later on.

2.3 Inspiration for a Tutorial

There are several programs with the purpose of simplifying the programming process and to teach programming. In this context, we are interested in the approach to explain the workflow and the usage of the programs. These programs are used as an inspirational resource for creating a tutorial for *Pocket Code*.

Furthermore, the implementation, advantages, and disadvantages can be analyzed on already existing examples. So the basic concepts are extracted from the existing tutorials and can be adapted for mobile applications. Here, we are going to discuss the following programming environments:

- *Scratch* [Scratch, 2014], details follow in Section 2.3.1.
- *Wario Ware D.I.Y.* [Wario Ware D.I.Y, 2014], details follow in Section 2.3.2.
- *Alice* [Alice, 2014], details follow in Section 2.3.3.
- *Kodu* [Kodu, 2014], details follow in Section 2.3.4.

2.3.1 Scratch

Scratch is a visual programming language including a development environment developed by the *Lifelong Kindergarten Group* at the *MIT Media Lab*. The user can program stories, interactive games, and animations within the developing environment of *Scratch*. The programs can be shared with other users over the website. It has been developed since 2007 and the current version is 2.0. The release of *Scratch* 2.0 replaced the desktop programming environment with a web-based environment. So the software is platform independent and the major purpose is to teach the concepts of programming. The main target group are kids but it is also suitable for teenagers and adults. In the program, objects are created and the behavior, look and, sound is customized with the code of the users. In *Scratch*, there are blocks which represent code fragments. With these blocks, it is possible for example to change the look of the object. Furthermore, the basic control structures of programming languages are included (e.g. loops, if-statements). Please see Figure 2.1 to take a look on the user interface. Before *Scratch* 2.0, there were only online manuals and so called *Scratch* cards. A *Scratch* card explains a specific task in *Scratch* (e.g. how to change the color). Since *Scratch* 2.0, there is an additional step by step introduction available. In ten steps, a simple program is created to show the user the basic usage. [Scratch, 2014]

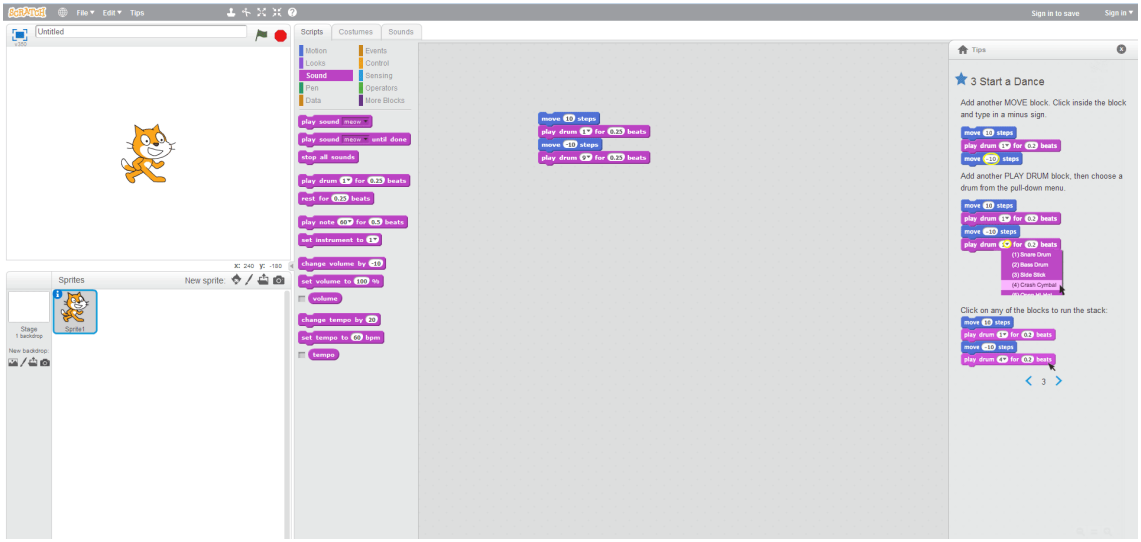


Figure 2.1: Scratch with Step by Step Introduction [Scratch, 2014].

The advantages and disadvantages of *Scratch* are:

- The usage of *Scratch* is simple. A program can be put together by dragging and dropping of the programming elements needed.
- A complex project idea is hard to realize. The reason is the more complex a project is the more complex the scripts get. Therefore, the whole project gets hard to read.
- The programming structures (e.g. loops) are visible.
- Suitable for people of all ages.
- *Scratch* can be used for projects in different contexts, not only for computer science topics.

2.3.2 Wario Ware D.I.Y.

Wario Ware D.I.Y. is a game for the handheld game console *Nintendo DS*. The game consists of about 90 mini games. The characteristic of the game is the possibility to create your own mini games. These games can be shared and downloaded on the website. Also the tutorial at the beginning is very detailed. So after starting the game a tutor shows and tells the way through the first steps. Also the user can try out simple predefined mini games to see the possibilities. There is a simple story behind the tutorial. The story is that the user is a new programmer in the company of *Wario*. Therefore, the user has to start creating games or taking some teaching lessons from the tutors. In three lessons, the users is taught the main functionality of how to create a mini game within a guided tutorial. It is designed with tutors, who guide the users step by step through the tutorial. In every lesson, the user gets more freedom and has to do more steps on his own. In Figure 2.2, you can find screenshots of a tutorial lesson in *Wario Ware D.I.Y.* [Wario Ware D.I.Y, 2014]

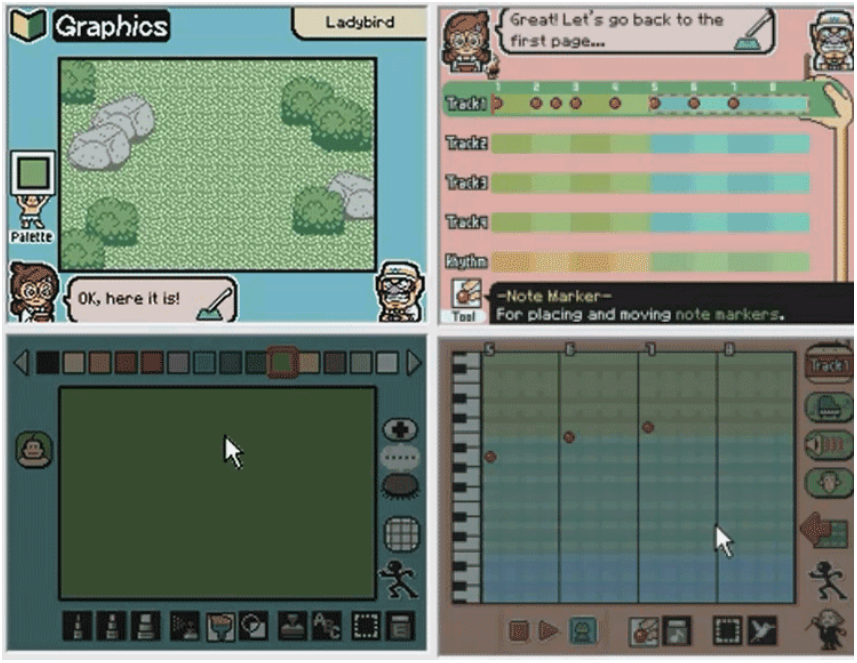


Figure 2.2: Wario Ware D.I.Y. Tutorials Screens [Wario Ware D.I.Y, 2014].

The advantages and disadvantages of *Wario Ware D.I.Y.* are:

- The ease of use.
- Only point and tap are available for user actions. For example button clicks are not programmable.
- A detailed introduction which covers the basic concepts for creating games.
- Different templates are available.

2.3.3 Alice

The *Alice Project* is a multi-university project. Among others, the *Carnegie Mellon University* is a big contributor to the project. *Alice* is a programming environment for creating 3D animations. In the programming environment, the user can create stories, interactive games, and videos for the web. It is a free available teaching tool and shows the users the concepts of object-oriented programming. In *Alice* 3D objects are available to insert into an virtual world and the task is to program the behavior of the objects. The user can drag and drop the components of the program (e.g. if-statement) and put them together to a runnable animation. There is also a tutorial implemented in *Alice* 2.3. The tutorial is divided into four lessons. Please see Figure 2.3 for the dialog to select one of the lessons. The steps in the tutorial are presented on notes which are on top of the *Alice* environment. In Figure 2.4 the tutorial in *Alice* is displayed. [Alice, 2014]

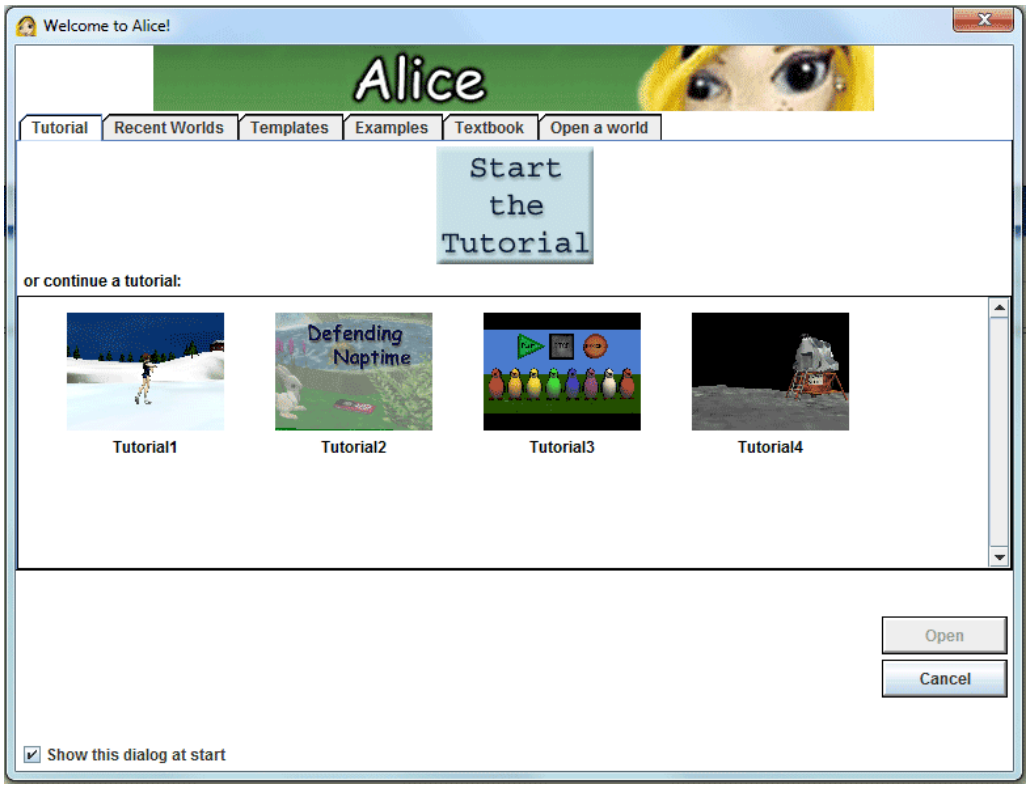


Figure 2.3: Alice Tutorial Selection [Alice, 2014].

In the first lesson, the tutorial is explained and a quick tour through the development environment is given. Also a simple routine is programmed on an ice skater example. In the second lesson, the creation of objects and how to create methods are presented on a bunny example. In the third lesson, the user is taught how to create responses on mouse and keyboard events with a penguin example. In the last lesson, the user learns how to create customized scenes for the programs on top of a spaceship example. On the website, there are also screen capture videos and textual manuals available for further help. By following these tutorials, the basic concepts of programming in *Alice* and the *Alice* programming environment is introduced. On top of this, the users should be able to create their own projects. [Alice, 2014]

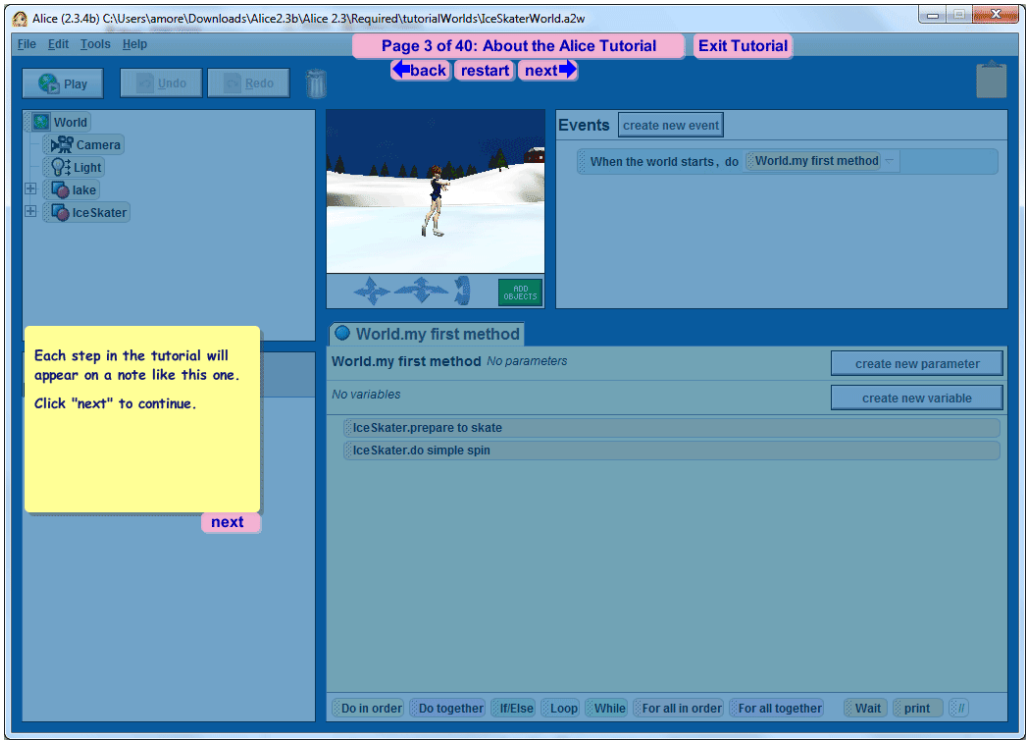


Figure 2.4: Alice Tutorial Screen [Alice, 2014].

The advantages and disadvantages of *Alice* are:

- The programming elements can be dragged and dropped into the script.
- The programmed sequences are visible to the user.
- There is no need to learn a special syntax.
- The main purpose of the software is teaching.

2.3.4 Kodu

The *Kodu Game Lab* is a programming environment developed by *Microsoft Research*. The main purpose is to provide an environment to create games without any programming knowledge for PC and *XBox360*. The target group are primarily kids. A program can be created with simple mouse clicks by creating objects and giving them attributes. There are several tutorials to provide an easy access into programming. In different lessons, the usage is presented to the user. In a step by step procedure, the user creates first games. So the user can learn the functionality by going through the tutorial courses. For example one tutorial covers the object creation (creating an apple and a gaming agent, a so called *Kodu*). Besides, in the tutorial the concepts of programming the *Kodu* agent to eat the apple is described. In another tutorial, the user is shown how to add and change the background, as well as the usage of scores for the created games. In Figure 2.5 and Figure 2.6 screenshots of the tutorial selection and presentation are shown. Furthermore, there are several examples. The created games can also be shared in a community and other games can be downloaded. [Kodu, 2014]

Figure 2.5: Kodu Tutorial Selection [Kodu, 2014].

The advantages and disadvantages of *Kodu* are:

- Programs can be created with a keyboard or a gamepad because *Kodu* is available for PC and *XBox360*.
- Introduction lessons are available.
- The environment is in 3D.
- The programming fragments (e.g. loops) are hidden.

Figure 2.6: Kodu Tutorial Screen [Kodu, 2014].

2.4 Gamification

Gamification is a procedure to make a special context more appealing to users. Therefore, typical gaming elements are used in a non-gaming context. This is done for a better engagement of the user to the application. Furthermore, complex tasks which otherwise can be stressful or boring to the user get solved in a fun way. In the following sections, this term and the basic elements are discussed in more detail.

2.4.1 Definition

The term *Gamification* first appeared in digital media and is often referred to other terms like *"funware"* or *"productivity games"*. Meanwhile, the term *Gamification* is a well known term in the field of games. *Gamification* is the term for using design elements in non-gaming contexts, which are typically used for games. *Gamification* correlates to games and is distinguished into playing and playfulness. The definition is build on gameful design and therefore, the use of game design elements. A gamified application consists of gaming elements. [Deterding et al., 2011]

The main reason for using *Gamification* is to increase the engagement of the user to the application. There are different further reasons why games are beneficial for learning. Games afford an abstract way to gain experience which cannot be achieved by just reading a book. The users interact with the software and make decisions to achieve progress. [Grappiolo et al., 2011]

Games offer the opportunity to play different roles and make experiences from different points of view without any serious consequences (e.g. learning different conflict solutions). For tutorials, there are several gaming elements which can be used. Further, the gaming elements fantasy, goals, feedback, disclosure, time pressure, rewards, and stimuli are discussed in more detail in the next sections. [Li et al., 2012]

2.4.2 Fantasy

Fantasy is a feature which can be easily transferred to different user interfaces. In an application, fantasy is the ability to create a mental image of physical objects in the mind of the users. Fantasy happens only in the mind of the users. So objects and situations, which are not real, are created. Therefore, the user can connect emotionally with the game. Fantasy splits up into emotions and metaphors. Besides, an emotional factor is created and a metaphor can help the users by learning to use the application. This means a similar concept can be implemented on different applications. So the users already know the handling. For example a floppy disc icon symbolizes a save function. Fantasy supports the process of satisfying the emotional need of the users. This process is difficult to implement because different users have different needs. So a good middle way has to be found. One way is to realize the fantasy of the target group and another way is to provide several presentations for different user groups. [Li et al., 2012] [Malone, 1982]

2.4.3 Goals

A challenging activity is the need of having a goal with an unknown outcome. This means the users should be uncertain about reaching the goal. On the other hand, the users should also be uncertain about not reaching the goal. Therefore, it is more enjoyable and the users do not get bored. However, the users should get feedback about their progress in achieving a goal. One approach for making a goal uncertain to reach is to implement multiple level goals. Furthermore, common approaches are implemented in keeping score and creating time pressure. Clear goals are important to give the users a definition of the tasks. Goals also help the users to understand the assignment. The goals should be presented early but at an appropriate time and they should also be clearly defined. The feature of incentives gives the user an additional motivating factor in the learning environment. So the performance is measurable and can be compared to others. [McNamara et al., 2010] [Sweetser and Wyeth, 2005] [Li et al., 2012] [Malone, 1982]

2.4.4 Feedback and Guidance

Typically in games the feedback comes instantly to the user. So the user is informed at any time about the progress. Furthermore, the system should help the user in case of erroneous behavior. The feature of giving the user feedback is an important construct for learning. On the one hand giving the user reassurance, while on the other hand providing critical comments and help messages. Influences on feedback in the learning process are timing (immediate or delayed), content (error feedback or explanatory feedback), and delivery-method (visual or auditory). The user should always have the feeling to have control over the character, the actions, and the environment in the game. The immediate feedback to the user is important for the progress in the game. Therefore, the players are aware of their current status. [Sweetser and Wyeth, 2005] [McNamara et al., 2010] [Li et al., 2012]

2.4.5 Progressive Disclosure

The game should provide an increase of the skills. So it is ensured the task matches the skills of the users. The application should challenge the user in a way the users do not get the feeling it is impossible to accomplish. For example a tutorial can help novice users at the beginning and gives just hints to expert users. The user should concentrate on the game, so the game should provide incentives to keep the players busy. The users should have the possibility to start the game without any previous knowledge. Every skill needed can be learned within the game. Furthermore, the learning phase should be also interesting. The feature of varying the task difficulty can be critical. If the difficulty is too high the user gets frustrated and quits using the application. If the difficulty is too low the user gets bored and has the same effect of exiting the application. So an optimal level of challenge for the user has to be provided. [McNamara et al., 2010] [Li et al., 2012] [Sweetser and Wyeth, 2005]

2.4.6 Time Pressure

Time makes tasks more difficult and more challenging for the users. The users get the task to complete a set of tasks within a specific time range. As a reward, it is possible to gain extra points for completing in time. The ratio between number of tasks and the time range has to be set at a point where it is challenging for the user but not impossible to accomplish. [von Ahn and Dabbish, 2008] [Li et al., 2012]

2.4.7 Rewards and Punishments

Rewards act as a motivation for the user, e.g. to gain points or unlock levels. In contrast to rewards, there are also punishments for the user in the game. The punishments have to be used carefully. It can have the effect of frustrating the user. So it is possible to destroy the engagement to the application by an excessive use of punishments. It can be a tool for learning with a small degree of punishment to show the user erroneous actions. There are different types of rewards and punishments in games identified. These rewards and punishments are found in Table 2.3. [Hallford and Hallford, 2001] [Gazzard, 2011] [Juul, 2009]

Reward/Punishment	**Description**
Rewards of Glory	This type of reward does not have an impact on the gameplay. These rewards give the user a positive experience. An example for such a reward are coins which can be collected. There is no forwarding of the level progression but the users get the possibility to compare their score with others.
Rewards of Sustenance	This type of reward is an extension of the rewards of glory. For example when collecting a defined number of coins the user can gain extra lives. So these rewards extend the possible time for playing the game without restarting.

Reward/Punishment	Description
Rewards of Access	This type of reward allows the users to access new locations or resources which were locked before in the game. The rewards have no further use in the game. An example for such a reward is a key to unlock a door within the game.
Rewards of Facility	This type of reward enables the user to do tasks in the game which were not able before.
Energy Punishment	The user has an energy bar which is decreased when performing a false move.
Life Punishment	For making a mistake in the game one life is taken from the user.
Game Termination Punishment	Some actions in the game are punished with terminating the game.
Setback Punishment	This punishment brings the user back to a specific point. For example the user has to play one level from the beginning.

Table 2.3: Rewards and Punishment in Games [Hallford and Hallford, 2001], [Gazzard, 2011], [Juul, 2009].

2.4.8 Stimuli

Stimuli provide a higher engagement of the user to the game. This is done by appealing the user with high quality graphic and sounds. So it is a pleasure for the user to play the game. The game experience should be deep. Furthermore, the users should feel involved in the game. The application should provide platforms for the communication between users. The feature of control provides influence of the user on the environment. So the environment can be personalized by the user and is dependent on the choices of the user. For example, the goals vary depending on the success of the user in the game. The feature of the environment defines the design of the application. [McNamara et al., 2010] [Sweetser and Wyeth, 2005] [Li et al., 2012]

2.5 Mobile Learning

Over the past years, mobile platforms have found their way into the everyday life. The main purposes of mobile phones are telephoning, texting, and playing games. So there is a new interest in the learning behavior with mobile devices. In this thesis, primarily mobile phones and tablets are referred to mobile learning devices. Other mobile devices, like the *Nintendo DS*, are not considered in the following.

2.5.1 Definition

On top of the basic concepts of e-learning, the term mobile learning (*m-learning*) was defined. The one definition of *m-learning* which is used in this thesis is: *"The use of mobile devices to provide access to learning content and information resources is called m-learning."* [DeGani et al., 2010] [DefinitionforMobileLearning, 2014]

Of course there are several advantages and disadvantages for learning with mobile devices and learning games. In [Mitchell and Smith, 2004] the following are identified:

- The learning objects may not be equivalent to the gaming objectives.
- The focus on winning and completing can distract from the actual learning goal.
- The learning material may not be adaptable.
- The game cannot reach every target group (e.g. male and female have different preferences).
- The learning games are too easy or difficult and therefore, it decreases motivation.
- *M-learning* eliminates the lack of interest and confidence.
- It reduces training time because of an easy access for the users.
- The learning material is processed visually.
- It provides a risk-free environment for learning.
- Can be adapted to different learning speeds and styles.
- Supports cognitive learning and decision making.

The possibilities for mobile learning expand with the available features of mobile devices, like GPS, Bluetooth, and motion sensors. For example with the GPS feature, the position of the user can be used within the application. So the application gets more interactive with this feature. Furthermore, the users feel as a part of the learning experience and it also influences the learning effect by making it more sustainable. [Lavin-Mera et al., 2009]

In the following sections, detailed information about learning principles (Section 2.5.2), adaptive learning (Section 2.5.4), learnability (Section 2.5.3), and serious games (Section 2.5.5) are presented.

2.5.2 Learning Principles

Including principles of learning into applications, as well as in games, is a challenging part. For creating educational software, four learning principles are identified, these are the active and critical learning principle, the design principle, the semiotic principle, and the semiotic domain principle. These principles are discussed in the following. [Gee, 2003]

Active Learning Principle

The learning environment encourages an active and critical learning experience of the student. The users should feel like an active agent in a positive learning experience. The actions and decisions of the users take effect in the user interface. So the users have influence on the design of the application. For a deep learning, the user has to be engaged to the application. Therefore, the users can create a new identity in which they are committed to. The users get challenged in the application and so the users do not get bored. The problems should be ordered in a way which brings the user from easy solvable problems to complex tasks. The learning experience rises if the difficulty level changes. The challenges encourage users to go further and have the feeling that the task is hard but doable. So there is a progress visible for the user. They also get the chance to practice new skills and develop the ability to automate the newly learned skill. With this approach, the application can control the learning pace for the user. [Gee, 2004] [Gee, 2003] [Mitchell and Savill-Smith, 2004]

Design Principle

The design of the application supports the learning experience on a visual level. The users should be able to customize the style of learning and it should also be possible to change the style. So the best way for the user to learn content is achieved. The users should get the feeling to manipulate objects in the application, for example the user is able to control a robot. With this, the effectiveness of the actions is increased. The manipulation helps the users to reach the goals. Sandboxes support the learning process by putting users into situations which imitate the real application. So the users can learn, but cannot do anything wrong quickly. [Gee, 2004] [Gee, 2003] [Mitchell and Savill-Smith, 2004]

Semiotic Principle

The use of a multiple sign system (e.g. images, words, symbols) is important to represent the learning material in an appealing representation. The capability of processing textual information out of context is poorly integrated in the human mind. In an application, the information, which is needed, should be accessible when the user can apply it ("Just in Time") or when the user has the feeling he/she needs help ("On Demand"). [Gee, 2004] [Gee, 2003] [Mitchell and Savill-Smith, 2004]

Semiotic Domain Principle

At some point the user has to apply the learned material on the specific domain and further, the users have to adapt it to other domains. The best way of learning is to give users the overall picture because then the users will understand how it fits into the actual application. When the user learns a new skill, it should not be out of context. Otherwise, the user gets bored quickly and the learning process is not effective. So when introducing a new skill to the users, there should be a relation to the application. A set of related skills can be summarized within a strategy. [Gee, 2004] [Gee, 2003] [Mitchell and Savill-Smith, 2004]

2.5.3 Learnability

In the process of creating an application, usability is one factor to evaluate the user interface. One aspect of usability is the ability to learn how to handle the application. This aspect is called learnability and is also correlated to the use of tutorials in software. A tutorial supports the learnability factor. The term learnability is defined in different contexts (e.g. linguistics, mathematics). In software development, learnability means a system is easy to learn by the user and it is also part of the initial learning curve. [Nielsen, 1994]

In this thesis, this definition is used and is related to information design. So the learnable information has to match five factors. It has to be memorable, logical, reconstructable, consistent, and visual. For memorable information, the human memory has big influence on the learning experience. The memory is connected to associativity. Many checklists and procedural lists are hard to remember and decrease the learnability. The user has to go back to the starting point to remember the executed task. Learning material, which is perceived as logical, should be presented step by step to the user. There is just the information which is needed for the task. The learned material is reconstructable if the concept can be performed without referring to a specific sequence of actions (e.g. the folder structure in *MS Windows* and in a command line interface). The information has to be consistent in the use of terms, phrases, and style. The visual presentation of information is also a tool to increase learnability. [Haramundanis, 2001]

Nevertheless, learnability strongly depends on the experience of the user. The level of experience with computers and mobile devices, quality of domain knowledge, and the experience with similar applications are relevant criteria. The learning process is distinguished into three phases, initial learning, extending learning, and learning as a function of experience. The initial learning process means that users have no knowledge about the context of the application and they have to learn the usage of it. The extending learning process covers that users have knowledge about the context and only need to get familiar with the instructions in the specific applications. In this case, a step by step tutorial can be boring for an experienced user. So it is important to give the user the possibility of stopping or forwarding the tutorial. The learning as a function of experience process is the middle way of the initial learning and the extended learning. The user has no domain knowledge but has knowledge about a similar system. So the user has a general understanding of which tools and functions are available. For instance, a user of *Scratch* will find their way through *Pocket Code* more easily than a user without previous experiences. [Grossman et al., 2009]

2.5.4 Adaptive Learning

The approach of adaptive learning is introduced in this section. Here, we define adaptive learning as the ability to adapt the software to the learning requirements of the user. For example *Pocket Code* is an application with the basic idea to show kids and teenagers the concepts of computational programming (e.g. conditions, iterative, loops). In order to offer the users an attractive environment, they should be motivated to create and share their own projects. Furthermore, the users are taught programming skills in a game-based way. The complexity of learning to write programs in *Pocket Code* is very high. Also understanding the workflow and the relationship between the programming fragments are challenging for the user without help. So learning to program from scratch through exploration is frustrating and not effective. For example a tutorial is useful to teach the basic steps in a playful way. So the users can adapt the material learned with the application for further use. [Torrente et al., 2009]

The interactivity is important to get a personalized experience. This is an approach to let users learn on their own but also give them the possibility to get answers on ambiguities without the need of a human teacher. The main aspect for this approach is the capability of the user to adapt the concepts. The adaptation in learning depends on different aspects, for instance the level of prior knowledge and learning styles, as well as the combination of several aspects. [Torrente et al., 2009]

Adaptive user interfaces are not necessarily a benefit. User interfaces do not appear to the users in the same way as designers intend to. There is always the possibility, the interface is not comprehensible and does not give the user the feeling of control. The users cannot build adequate mental models of the system. In some domains, the understanding of the system design model does not need to be known by the user for sufficient ease of use. [Paymans et al., 2004]

In literature, there are several technologies how to implement the adaption in a system. According to [Brusilovsky, 1998], these approaches can be found in Table 2.4.

Name	**Description**
Curriculum Sequencing	The sequence of tasks is according to the preferences of the user. So it helps the user to find an individual path of tasks through the information. Furthermore, it is distinguished between knowledge and task sequencing. The difference is the sequence based on the topic or the task to be learned.
Intelligent Analysis of User Solutions	This approach deals with actions and solutions of problems provided by the user. So the user gets feedback about the solutions (e.g. what was wrong, what was incomplete).
Interactive Problem Solving Support	The goal is to provide interactive help for the user. Depending on the user, the help can be a hint for the next step or even execute the next step for the user. So the actions of the users have to be watched and it needs to be saved at which level the user needs help.
Example Based Problem Solving	This approach saves examples from previous experiences of the user and this examples are used by students to solve a specific problem. The system suggests which previous example is relevant.
Adaptive Presentation	The visual presentation is based on the preferences of the users (e.g. expert users get more detail information than novice users).
Adaptive Collaboration Support	Here, the preferences of different users are collected to be combined into a collaborating group.

Table 2.4: Several Technologies for Adaption [Brusilovsky, 1998].

2.5.5 Serious Games

In first instance, the purpose of games is to entertain. Typical elements of games can be used in education, like presented in the previous sections. Similar to *Gamification*, where gaming elements are used to make a non-gaming content more appealing to a user, serious games combine entertainment with non-entertainment factors like education. The principles are adaptable for the combination of gaming elements with other contents (e.g. advertising, simulation, and politics). For education, the aspects are for example teaching, training, and informing. The idea behind is to give a motivating platform for learning a specific topic to the user. Furthermore, games give users the possibility to interactively take actions and make decisions that impact the content. Users also have the opportunity to practice their knowledge on a secure virtual environment. [Hakulinen, 2011] [Grappiolo et al., 2011] [Barbosa and Silva, 2011]

The challenging part of designing a serious game is finding the balance between serious objectives with fun and interactive elements. The design process can be split up into the following steps: [Chaffin and Barnes, 2010]

- Identify the target purpose (e.g. education).
- Identify measurable objectives which can be achieved (e.g. solve a puzzle).
- Create a metaphor to connect the purpose to the objectives (e.g. a brick is a programming block).
- During creating the metaphor define the instructions to meet the purpose (e.g. putting several bricks into a script).
- Prepare support for the users to achieve the objectives (e.g. tutorials).
- Create a prototype.
- Test the effectiveness of the game (e.g. with a usability test).

2.6 Implementation in Pocket Code

During the work for this thesis, prototypes for a guided tutorial and a tooltip system were developed. Here, the basic structure of all prototypes is presented. Further, a simpler version of a tutorial in the form of a tooltip system was created. The first tutorial prototype was developed on the basis of a younger target group than on the second tutorial prototype. Furthermore, the lessons learned from creating the first prototype were helpful to create the tutorial for a target group of teenagers. The design for user interface of the second prototype is later presented in Section 3.3. Also the visual presentation of the tooltip system is shown in Section 3.2. In the following sections details about the implementation of the prototypes are presented.

2.6.1 First Prototype

The first prototype of a guided tutorial for *Pocket Code* was intended for a target group of children at an age between 8 and 14 years. Therefore, the tutors were created as cartoon animals (cat and mouse). In Figure 2.7 the tutors are shown.

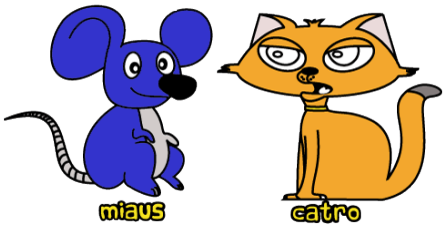


Figure 2.7: Tutors for the First Prototype.

The tutorial has to be started explicitly by the user. In the first dialog the users can choose which lesson of the tutorial they want to take. The lessons, the behavior, and the words for the tutors are saved in an *XML*-file. The file consists of the elements presented in Table 2.5.

***XML*-Tag**	**Description**
`<LessonCollection>`	This element is the root element of the *XML*-file. It consists of one `<LessonArray>` and the `<CurrentLesson>` of the tutorial.
`<LessonArray>`	This element contains several `<Lesson>` elements which can be chosen by the users after activation. A `<Lesson>` is activated after finishing the previous one.
`<Lesson>`	This element includes `<LessonName>`, `<LessonContent>`, `<LessonID>`, and `<CurrentStep>` which are described in the following in detail.
`<LessonName>`	This element contains the name of the lesson which is further presented in the user interface.
`<LessonContent>`	This element contains the tasks for the tutors. The different tasks are described in the following.
`<LessonID>`	This element is a unique id for the lesson.
`<CurrentStep>`	This element describes the current position of the user in the lesson. So the user can go forward and backward within a lesson.
`<CurrentLesson>`	The value describes how far the user is with taking the tutorial lessons. So the lessons are activated one after another.
`<TaskAppear>`	This task makes the tutor appear at a specific position. Therefore this element consists of a `<TutorType>`, the horizontal, and vertical coordinates.
`<TaskDisappear>`	This task makes the given tutor disappear from the user interface. Furthermore, it consists of the `<TutorType>`.
`<TaskSay>`	This task has the function to make the tutor speak. It consists of the `<TutorType>` and a `<message>`.

XML-Tag	Description
`<TaskFlip>`	This task lets the tutor flip in the opposite direction. Therefore, it consists of the `<TutorType>`.
`<TaskWalk>`	This task makes the tutor walk from one position to another. Besides, it consists of the `<TutorType>` and the coordinates of the position to walk to.
`<TaskJump>`	This task handles the ability of the tutor to jump to another position. Therefore, it consists of the `<TutorType>` and the coordinates for the new position.
`<TaskSleep>`	This task defines the time the tutor does nothing. It contains the *TutorType* and the time to wait in milliseconds.
`<TaskNotification>`	This task is a notification for which the execution of the tutorial is waiting. So it consists of a `<NotificationType>` and `<NotificationString>`. For example the user has to tap on a button before the tutorial moves further to the next step.
`<TutorType>`	This element defines which tutor executes the corresponding task. *CATRO* or *MIAUS* are a valid value.
`<message>`	The text entered in this element is shown to the user. It is shown to the user in a bubble and it appears letter per letter.
`<NotificationType>`	This element defines which notification is active. Therefore, the tutorial is waiting for an input from the user.
`<NotificationString>`	This element provides additional information for the `<TaskNotification>`.

Table 2.5: *XML*-Elements for the Tutorial.

In Listing 2.1 there are concrete examples for the use of the *XML*-file for the tutorial. The listing is a minimal example. So every task is only used once.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<org.catrobat.catroid.tutorial.LessonCollection>
  <LessonArray>
    <org.catrobat.catroid.tutorial.Lesson>
      <LessonName>Erste Schritte</LessonName>
      <LessonContent>

        <org.catrobat.catroid.tutorial.tasks.TaskAppear>
          <TutorType>CATRO</TutorType>
          <x>60</x>
          <y>55</y>
        </org.catrobat.catroid.tutorial.tasks.TaskAppear>

        <org.catrobat.catroid.tutorial.tasks.TaskSay>
          <TutorType>CATRO</TutorType>
          <message> Here is the text for the tutor. </message>
        </org.catrobat.catroid.tutorial.tasks.TaskSay>

        <org.catrobat.catroid.tutorial.tasks.TaskFlip>
          <TutorType>CATRO</TutorType>
        </org.catrobat.catroid.tutorial.tasks.TaskFlip>

        <org.catrobat.catroid.tutorial.tasks.TaskWalk>
          <TutorType>CATRO</TutorType>
          <walkToX>50</walkToX>
          <walkToY>70</walkToY>
        </org.catrobat.catroid.tutorial.tasks.TaskWalk>

        <org.catrobat.catroid.tutorial.tasks.TaskJump>
          <TutorType>CATRO</TutorType>
          <newX>50</newX>
          <newY>70</newY>
        </org.catrobat.catroid.tutorial.tasks.TaskJump>

        <org.catrobat.catroid.tutorial.tasks.TaskSleep>
          <TutorType>CATRO</TutorType>
          <SleepTime> 100 </SleepTime>
        </org.catrobat.catroid.tutorial.tasks.TaskSleep>

        <org.catrobat.catroid.tutorial.tasks.TaskDisappear>
          <TutorType>CATRO</TutorType>
        </org.catrobat.catroid.tutorial.tasks.TaskDisappear>

        <org.catrobat.catroid.tutorial.tasks.TaskNotification>
          <NotificationType>BRICK_CATEGORY_DIALOG</NotificationType>
          <NotificationString>1</NotificationString>
        </org.catrobat.catroid.tutorial.tasks.TaskNotification>

      </LessonContent>
      <LessonID>0</LessonID>
      <CurrentStep>0</CurrentStep>
    </org.catrobat.catroid.tutorial.Lesson>
    <CurrentLesson>0</CurrentLesson>
  </LessonArray>
</org.catrobat.catroid.tutorial.LessonCollection>
```

Listing 2.1: Concrete *XML*-Example.

2.6.2 Second Prototype

This prototype is based on the experiences made with the first prototype. In cooperation with designers from the *Catrobat* team, a new design was created. Furthermore, the target group was changed to young teenagers aged between 12 and 16 years.

At a first execution of the application, the guided tutorial starts automatically. So the first lesson is mandatory to show the users the basic concepts. In case the users are already familiar with *Pocket Code*, a possibility to skip the beginning lesson is available.

On the contrary to the first prototype, only one tutor is used for guiding the user through the tutorial. The character of the tutor is developed as a teaching agent. In Figure 2.8 the design of the created tutor is presented. The second prototype for the tutorial is still in development when finishing this thesis.

Figure 2.8: Design of Tutor for the Second Prototype.

The creation of the story and the sequence of actions of the tutor are saved in a *XML*-file like in the first prototype presented before. The tutorial is planned to be split up into two phases. The first phase is a mandatory lesson every user has to take at the first start up of *Pocket Code*. The second phase is to work with the application. The usage can be done with or without a tutorial depending on the personal decision. The design for the mandatory lesson is presented in Section 3.3 in detail.

2.6.3 Tooltip Implementation

The idea for a simple tooltip system came up because of the need of a tutorial. Therefore, short textual messages appear on the user interface when activated. So the user gets an idea what a specific element (e.g. a script item) does. The user can activate the tooltips by clicking on a button in the title bar. When activated, every important element of the user interface gets an additional question mark button. By clicking on the question mark, a tooltip for the corresponding element shows up. It disappears after a second click on the button. A detailed presentation of the implemented tooltip system is shown in Section 3.2. The tooltips are drawn on top of the application on a *SurfaceView* like the tutorials presented before. The position of the tooltips are computed out of the position of the corresponding element. So there is no need to save the positions. The messages are stored in the resources of the *Android* application. Thus the texts are easy accessible for the application.

2.7 Chapter Summary

In this chapter, different theories about the creation and the usage of tutorials are presented. The challenges of creating stories and tutors are explained in detail. So a sequence of steps is created and the users do not get bored when going through the tutorial. Therefore, different aspects are important, for example, the demonstration of tool handling, navigation within the application, and guidance of the tutor. A tutor can be created with different characteristics, for example, like a teacher. The benefit of the effort of creating a story and tutors is to engage the users to the application. Besides, factors like enjoyment, creativity, and gratification have to be satisfied to get an optimal engagement level for the users. Also some guidelines and best practices, for creating a tutorial, are introduced. Moreover, some inspirational resources were presented in this chapter (Wario Ware D.I.Y., Scratch, Alice, Kodu). Here, the approaches of the presented applications for a tutorial were analyzed and also the advantages and disadvantages were identified. Furthermore, the idea of using tutorials as a teaching tool was explained.

Therefore, teaching styles and learning processes were analyzed. Also the term of *Gamification* was defined. In this context, gaming elements, like fantasy, goals, feedback, progressive disclosure, time pressure, rewards, and punishments, were identified. Also the term mobile learning was defined which includes learning principles, a definition of adaptive learning, and the concept of learnability. At the end of the chapter, implementation details about the prototypes, which were developed for *Pocket Code*, are presented. The considerations of the different theories had influence on the prototype development. The most important part is the structure of the tutorial in the background. Therefore, a *XML*-structure was used to arrange the lessons. The content for the tutorial lessons and the tooltips are presented later in this thesis.

Chapter 3

Types Of Tutorials

For software products, an important part of the development is to give support to the end users. This support can be presented in different forms. Here, we focus on help and tutorial systems. The goal of these systems is to give the user a possibility to learn the necessary tasks for an effective use of the application. Furthermore, the content of the application can be communicated. Here, we start with a simple documentation, going over tooltips, and come to more complex training challenges. The design of the prototypes for the *Android* application *Pocket Code*, which have already been introduced in the previous chapters, will be presented.

First, an overview on help and tutorial systems in general will be given. In literature, different presentations of help systems were analyzed. In [Baecker, 2002] five examples for creating documentation and helping systems are explained briefly. These examples are summarized in the following (Table 3.1).

Name	**Description**
Screen Linking	In this method, the computer system of the user is connected to another computer system with a human supporter. So the supporter watches over the user and follows the steps visually. This method is primarily used in support services. It can also be used for live demonstrations and teaching tasks. Questions and problems are discussed on demand via chat or telephone. This approach is an abstract version of the face-to-face teaching style.
Visual Streaming	In this method, information is transferred as audio, video, slides and live demonstrations in a web interface. So the information can be presented to an unlimited number of people around the world. In a split screen, the data is divided, e.g. in a slide presentation with keywords, a demonstration screen and a chat system which are included in one browser window.
Animated Icons	In this method, animated icons are the key objects. Icons are small and easily recognizable for the user. The meanings of the icons have to be clear and universal. By adding animation to the icons a dynamic visual representation is added (e.g. adding a textual hint about the use of the icon).

Name	Description
Screen Capture	This method uses a tool to record the whole screen. The purpose is to document the steps for a specific task in a visual way. The capture is taken from the interaction of an expert user with the software. Text, sounds, and animations can be added to the capture for a better understanding. The range of use is demonstration, explanation, and guidance for the user.
Structured Web Video Systems	This approach is used for demonstration within a documentation system. A specific task is presented in a video which is embedded in a website. The videos are in a hierarchical structure. The user can watch the videos in the hierarchical order consecutively or jump just to single videos (e.g. a *"how insert a picture"* video).

Table 3.1: Examples for Help Systems [Baecker, 2002].

In the next sections, we want to go in detail with tutorial systems for mobile devices. Methods and guidelines for creating tutorials were developed in first instance for classical computer software systems. They are also relevant but have to be adapted to the special needs of mobile devices (e.g. smaller screens). Furthermore, the relevance of an appropriate design has to be considered.

There are different presentations of tutorials. Here, three approaches are discussed in detail with examples implemented in *Pocket Code*.

3.1 Manuals

This tutorial method is mentioned for completeness. It is the classical way to distribute information about software. Manuals are usually just text (and screenshots) and describe the main functions of software. This method is easy to realize and is used in practice very often in different forms. A manual is reactive, this means users tend to only look into a textual tutorial when they fail to perform a task correctly. In case of a simple written document, the user has to read a lot and switch between the software and the manual which leads to an interruption in the working flow of the users. For mobile applications, this approach is not suitable and disadvantageous because it is annoying for the user to change between screens on a mobile device. [Ames, 2001]

3.2 Help Buttons/Tooltips

A simple and easy way to give users information are help buttons and tooltips. On demand, the user can show and hide the additional information. So the information is always available on every screen, if needed. The information has to be short and descriptive for the specific element or screen.

A prototype of this design was developed for *Pocket Code*. In the design for the *Pocket Code* tooltip system, a question mark button in the action bar is added. With the button in the action bar, the tooltips are activated and deactivated. If the tooltip system is active in the current activity screen, the available tooltip buttons will appear next to the described element. With clicking on one of the buttons, a bubble with a short description of the function is shown. In Figure 3.1 and Figure 3.2 are screenshots of this design. So the user is able to turn information on and off as needed.

This approach just gives little information to the user. They have to walk through the application on their own and learn the workflow by exploration. The tooltips should be a support if the user is stuck at some point. One further approach for the use of such a system could be a combination with a guided tutorial.

Figure 3.1: Design for Tooltips in Pocket Code in the Main Menu.

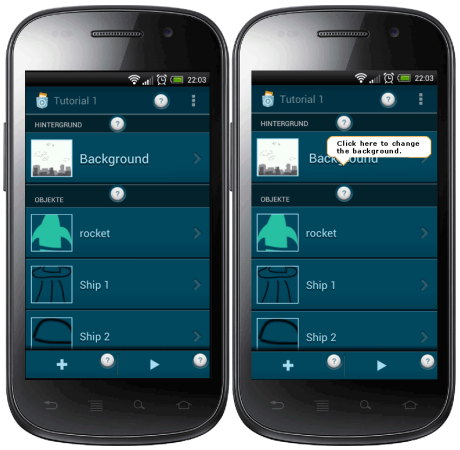


Figure 3.2: Tooltips in the Project Menu.

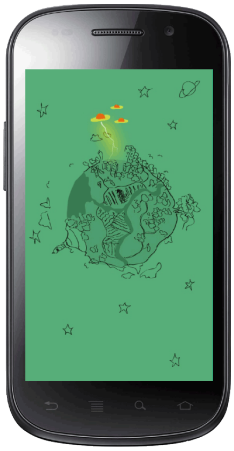
3.3 Training Challenges

Another approach to create a tutorial is the concept of training challenges. Therefore, users get the chance to learn the game play within a save environment. This training ground can be expanded by mentors or tutors which guide the user through the training.

During the work for this thesis, a prototype of such a tutorial system for *Pocket Code* was created. In *Pocket Code*, it is planned to create a mandatory beginner lesson to present the basic concepts to the users. The beginner lesson should also motivate the users to further use the application and become eager to know more about the functions and features of it. Further, there will be lessons to deepen the knowledge, if needed, integrated in the *Pocket Code* interface. It should include several lessons and in each lesson the level of difficulty is raised. This means that in the beginning lesson, every step is explained in detail and the user is told what to do next. In the further lessons of the tutorial, the users have to make more steps on their own. In case to avoid boredom, such a tutorial should have a story which addresses the target group.

Such a tutorial is in development for *Pocket Code*. In the following, the design for a starting lesson is presented which was created in cooperation with the *Catrobat* team.

Start screen of the tutorial. The user starts the lesson by clicking on the start button.



A starting animation is processed.

The tutor introduces himself and gives instructions to the user for the next steps.

The predefined program is played. The user has to touch the rocket to go further.

As a result of the touch event the UFO explodes.

The tutor gives instructions to the user about what happens next.

The user gets information about the composition of a *Pocket Code* project.

The user has to draw a new object.

The user should press the *Next* button to finish the drawing.

The next instructions for the user are presented.

The user has to look at the coordinate axes and confirm with a button click.

The user needs to enter the position of the new object.

A new rocket is inserted at the given coordinates.

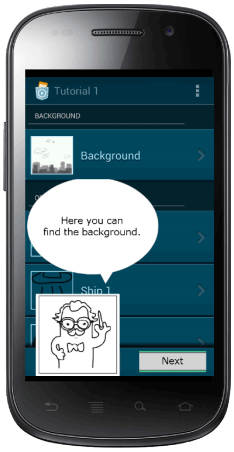
After clicking on the rocket, the tutor appears and explains the next step.

The user has to enter another set of coordinates to make the object move.

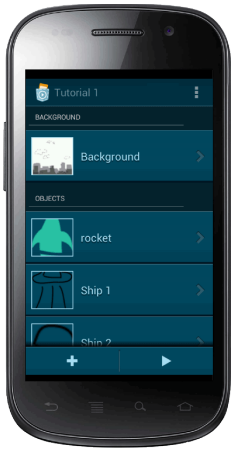
The object is now able to fly after a tap of the user.

The next UFO explodes.

The next step is to change the background.



The tutor explains where to find the available backgrounds.



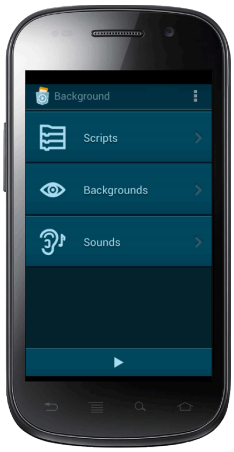
The user has to select the background button to continue.

The tutor explains the composition of the background.

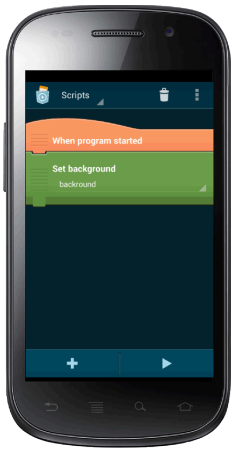
Detailed description of sounds and backgrounds are presented.

Further explanation about scripts are given to the user.

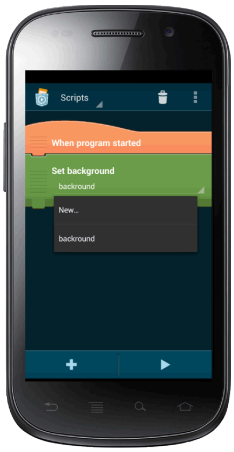
Further instructions for the user are presented.



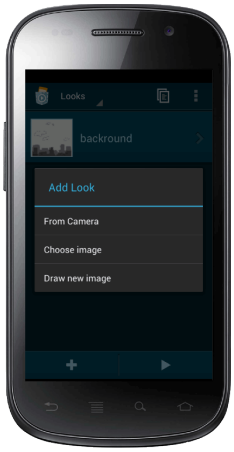
The user has to click on scripts to go further.



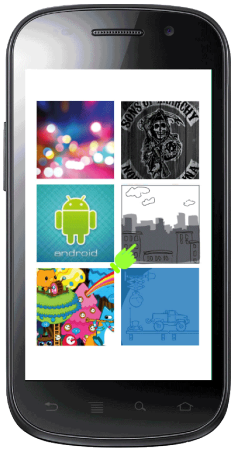
The script view is presented.



By clicking on background the user can change it.



The user has to select the source of the background.

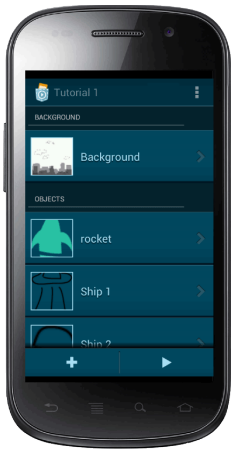


A background image has to be selected.

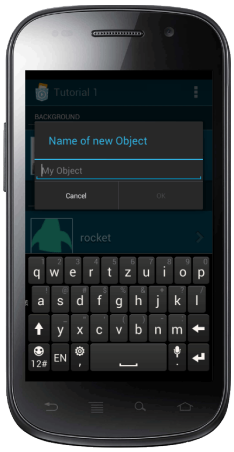
The new background is set and the user gets feedback of the tutor.

Further instructions are presented to the user.

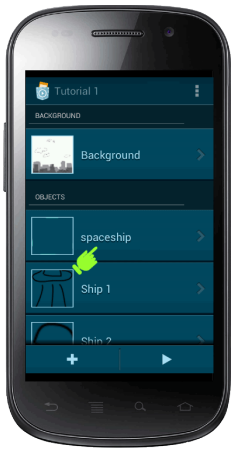
The tutor explains how to add a new object to the project.



The user has to click on the button for adding an object.

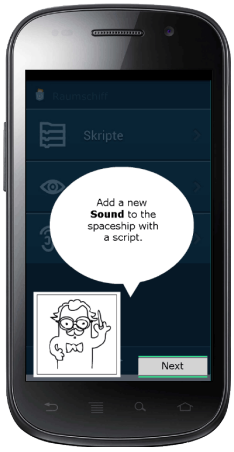


The user has to enter a name for the new object.

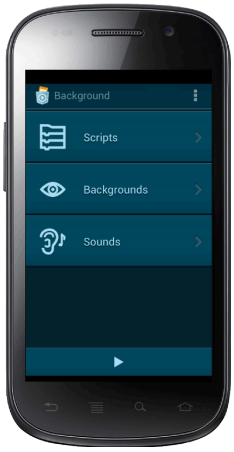


The created object has to be selected for the next step.

Further instructions are expressed by the tutor.



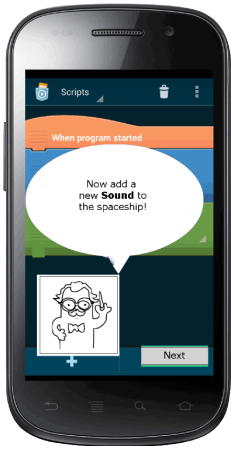
The tutor gives the instruction to create a new sound.



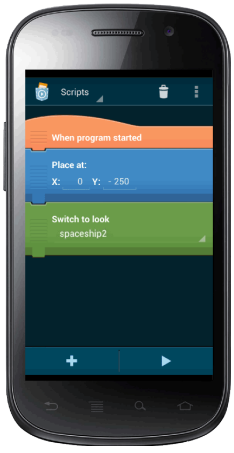
The user has to click on scripts to go further.

Explanation of the predefined bricks are presented.

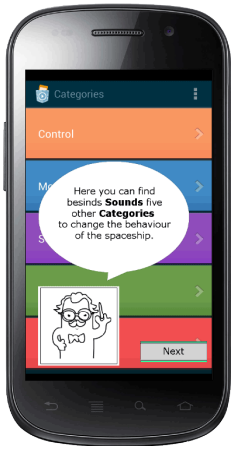
Information about every brick is explained.



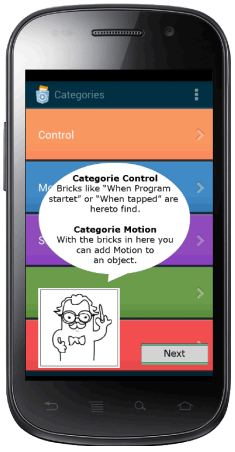
Further instructions are presented.



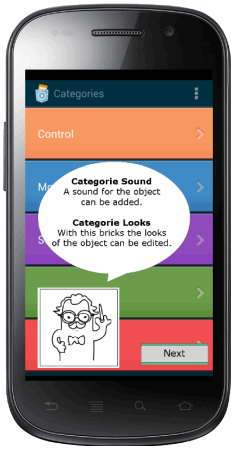
The user has to add a sound to the script.



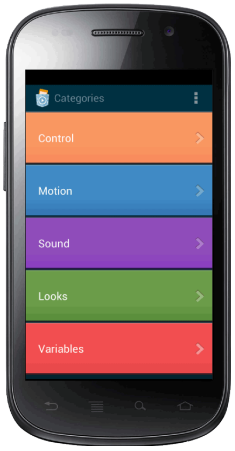
Detailed description to the category screen is shown.



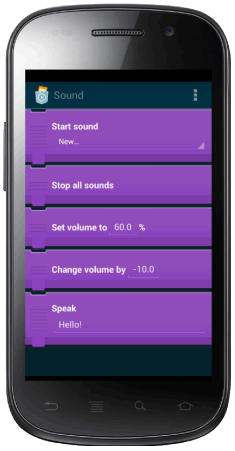
Detailed information about the categories *Control* and *Motion* are brought to the user.



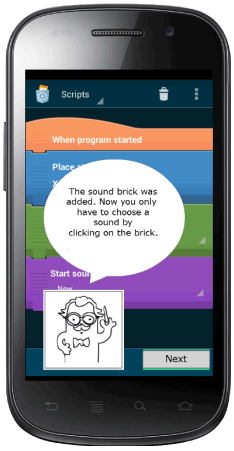
Detailed information about the categories *Sound* and *Looks* are brought to the user.



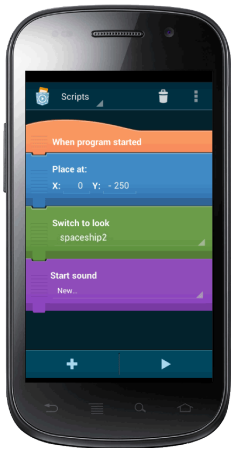
The user has to select the category *Sound* to go further.



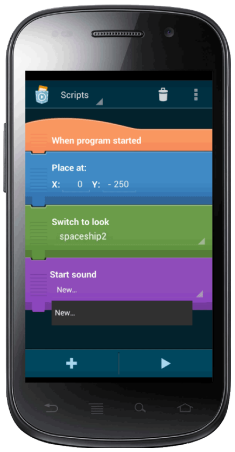
The *Start Sound* brick has to be selected.



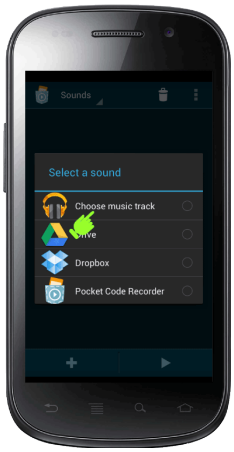
Instructions and explanations by the tutor are shown.



The new inserted brick has to be selected.



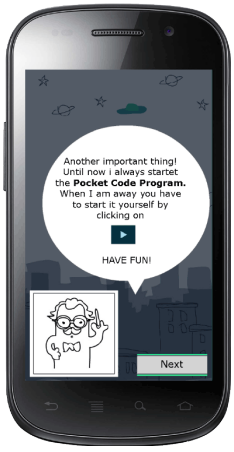
A new sound has to be inserted.



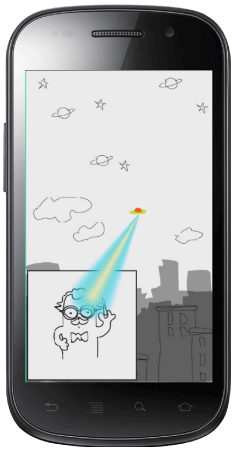
The source for the new sound has to be clicked.

The sound is selected by the user.

The sound is inserted into the project and the tutor gives last comments.



Final instructions in the lesson are shown.



The lesson is finished with this screen.

3.4 Chapter Summary

In this chapter, types of tutorials are discussed. Therefore, some examples were presented (e.g. screen linking, visual streaming, and screen capture). Furthermore, concrete examples, which were implemented on the base of the *Android* application *Pocket Code*, were introduced. So the design of a tooltip system and a guided tutorial are shown which were created in cooperation with the *Catrobat* team. Besides, guidelines and influential factors helped when creating the prototypes. Furthermore, the differences in the approaches were clarified. A guided tutorial with training lessons is applicable because of the complexity of the workflow of the application. Therefore, the implementation takes a lot of effort. So a faster solution for providing help to the users is the tooltip system. A combination out of both types are possible. Here, these prototypes show the possibilities and further work is done in the *Catrobat* project.

Chapter 4

Evaluation Methodologies

Usability evaluation covers the discovery of possible usability problems in a user interface. Furthermore, evaluation helps finding ways to solve these problems. The process of an evaluation can be split up into four general phases, preparation, collection, extraction, and analysis phase. These phases and a short description can be found in Table 4.1. [Balagtas-Fernandez and Hussmann, 2009]

Phase	**Description**
Preparation Phase	In this phase, the preparation of the application for logging is done. Therefore, the information, which is needed for an appropriate evaluation, is recorded. The main task for the evaluator is to decide which information is relevant for the usability evaluation, for example, metrics about efficiency.
Collection Phase	In this phase, data of the users is collected. This data includes usability problems, for example, implementing a logging functionality within the application to monitor the users actions.
Extraction Phase	This phase handles the preparation of the collected data for further analysis, for example, converting the collected material into a *XML*-file.
Analysis Phase	The data is analyzed referring to the problems of the users with the interface in the current evaluation phase, for example, processing the information in a visual presentation like a graph.

Table 4.1: Evaluation Phases [Balagtas-Fernandez and Hussmann, 2009].

The evaluation process includes the use of methodologies to gather information about the application. Here, we focus on the evaluation of usability. In the field of usability, there are different methodologies to evaluate software. Depending on the purpose of the evaluation, four categories can be identified: [Ellis and Dix, 2006]

- **Explorative:** The purpose of this category is to evaluate the usage of software and the state of the art of similar software. The evaluation is done before the interface is developed, for example, software logging and observational studies.

- **Predictive:** The purpose of this category is to predict the success of the software in practice. The evaluation estimates the quality of an interface in advance. This is done after a design is developed and before the implementation is done, for example action analysis.
- **Formative:** The purpose of this category is to get information to improve the interface design. The evaluation is done during the developing process. So problems in the interface are identified and can be eliminated. For example *Heuristic Evaluation* (Section 4.1), *Thinking Aloud Method* (Section 4.6) and *Cognitive Walkthrough* (Section 4.2).
- **Summative:** The purpose of this category is to evaluate the overall quality of the interface. The evaluation is done after the developing process. The performance of the users with the interface is evaluated, for example, questionnaires, *System Usability Scale* (Section 4.3), and *A/B Tests* (Section 4.5).

Here, we focus on formative and summative evaluation methods. For evaluating the effectiveness of a tutorial design, there are different methods possible. In the case of usability, there are the following evaluation methods which are discussed in detail in this thesis:

- **Heuristic Evaluation:** This informal method consists of usability experts which review the elements of an interface under usability principles. Details are explained in Section 4.1. [Nielsen and Molich, 1990]
- **Cognitive Walkthrough:** This method is used to analyze the problem solving behavior of the users by simulating the process. In this process, the goals and memory content of the users should lead to the predicted next step in the process. Details are explained in Section 4.2. [Lewis et al., 1990]
- **System Usability Scale**: This technique uses a questionnaire to gather information about the application under test. This information is assessed with a numerical value. Details are explained in Section 4.3. [Brooke, 1996]
- **Emocards**: This method is used to measure information about the emotional side of the users. With the usage of a so called *Emocard*, a value on a scale of pleasure and arousal is determined. Details are explained in Section 4.4. [Desmet, 2000]
- **A/B Test**: This testing method is used to compare two or more interfaces with each other. The users are split up into groups. Furthermore, each user group gets to evaluate one interface. Details are explained in Section 4.5. [Crook et al., 2009]
- **Thinking Aloud Method**: For this method the users have to speak out their thoughts loudly during the test. Details are explained in Section 4.6. [van Someren et al., 1994]

These evaluation methods can also be used for the evaluation with a younger target group. For the execution of usability test with children and teenagers the methodologies have to be adjusted. Therefore, the aspects in Table 4.2 have to be considered. [Larkin, 2002] [Hanna et al., 1997]

Aspect	Description
Detailed Planning	The test room should be decorated in a way the test users feel comfortable. Also make sure the users are familiar with the input methods. For example let the user get to know the device with some simple tasks. Furthermore, change the sequence of the tasks. So it is provided that the last tasks vary and the results are not compromised through the lack of concentration of the users at the end of the testing period.
Carefulness About Legal Issues	While the developing process of the evaluation materials for young users make sure to check legal issues. So the evaluation is assured against child labor laws. Furthermore, the parents should sign a legal agreement.
Age Appropriate Test Design	The vocabulary in the testing material should meet the users knowledge. If the users have problems reading or understanding words the facilitator should help the users.
Possibility for Feedback	Prepare the possibility for giving feedback to the tested application. For younger participants it is easier to give verbal feedback than writing it down. The facilitator can motivate the users to explain their problems with the software. So not only positive feedback is collected.
Age Appropriate Language	Depending on the users age, the attention span differs. So make sure the tasks are adjusted accordingly. Also include breaks for a test period over 45 minutes. Otherwise the concentration of the users is lost.
Knowledge of Participants	The topics in the application under test should meet the expert knowledge of the young users.
Explanation of Purpose	The participant has to feel comfortable in the testing environment. So it is necessary to explain the reason of the test in detail and also use an easily understandable language, appropriate for the test users. Therefore, prepare an introduction script. Also some small talk at the beginning supports to break the ice between the facilitator and the test persons. Furthermore, try to build appropriate expectations of the assignment. For example the test users do not expect a finished software under test.

Aspect	Description
Usage of the Results	The results of a usability evaluation with children or teenagers might be hard to implement. Besides, check the methodology against what you expect to get from the evaluation.

Table 4.2: Aspects for Usability Testing with Underage Participants [Larkin, 2002] [Hanna et al., 1997].

In the following sections the evaluation methodologies are discussed in detail. Furthermore, a proposal for adapting these methods for a younger target group is presented.

4.1 Heuristic Evaluation

The analytical evaluation method compares user interfaces against predefined guidelines and heuristics. This method is called *Heuristic Evaluation*. It is useful for a summative evaluation with a complete application and is also applicable for a formative evaluation with a prototype. This evaluation method and the adaption for underage participants are presented in the following sections. [MacFarlane and Pasiali, 2005]

4.1.1 Basic Concepts

The *Heuristic Evaluation* method analyzes user interfaces under the perspective of usability experts. So the evaluators assess the interface with positive and negative feedback. The so called heuristics are general usability principles and design guidelines. The experts take these heuristics for evaluation. In this context an expert is a person who is trained with the *Heuristic Evaluation* method. It is important that the expert has expertise knowledge about usability and the scope of the application. The experts do not have to be domain experts. [MacFarlane and Pasiali, 2005]

A basic set of a usability heuristic consist of the points presented in Table 4.3. [Molich and Nielsen, 1990] [Nielsen and Molich, 1990]

Heuristic Name	Description
Information Appearance	The interface should only contain relevant information. Irrelevant information should be hidden as long as it is not needed. Besides, the order of the information should be logical and intuitive for the user. For example a design which uses only upper-case letters does not fit this principle.
Clearness	Text in the user interface should be unambiguous for the user. System specific terms should be avoided or explained before they are used. For example initial phrases for a search of a telephone number should be expressed in the point of view of the user. So a phrase could be *"Enter the name for which you want the telephone number"* instead of just *"Enter name"*.

Heuristic Name	**Description**
Minimal Memory Load	The short time memory of a human user is limited. So instructions given to the user should be presented shortly before they are needed. A set of complex commands should be split up into several simple commands.
Consistency	Phrases, actions, and situations should be accessible in a consistent way over the whole application. For example a telephone number should always refer to the same phrase or abbreviation. Furthermore, the user understands the terminology. For example the navigation is consistent, logical, and minimalistic. Also the control keys are consistent and follow standard conventions.
Feedback	The application should keep the user informed about the current status. Also the user should get feedback within a short time after an action.
Exits	In case the user gets to an unwanted screen accidentally, the user should always have a possibility to exit a current state. This exit points should be clearly marked.
Shortcuts	For experienced users there should be shortcuts, which are not visible to novice users. So the user can skip time consuming initiation steps.
Error Messages	The error messages should be defensive, precise and constructive. So the messages do not criticize the user for the mistake and give the blame to the system. Therefore, an error message should describe the cause of the problem precisely and give the user a constructive proposal for a solution of the problem. For example the phrase *"illegal"* should not be in an error message.
Fast Start Up	The sessions can be started quickly. There is no long waiting time for the user after starting the application, like a long loading time. Furthermore, within the workflow, waiting times can be annoying for users.
Interruptions	Interruptions in the gameplay do not reset the whole game. The user can continue the process. Especially applications on mobile devices have to be available anytime and anywhere. Therefore, it is very frustrating for the user to start all over after a session.
Multimedia Presentation	Audio-visual representation supports the game. It is more appealing to users when utilizing the available possibilities to present information.

Heuristic Name	Description
Screen Layout	The screen layout is efficient and visually pleasing. Also the functions provided by the devices are included in the application (e.g. GPS connection, internet connection).
Clear Goals	The software provides clear goals and supports the goals, created by the users.
Progress	The user sees the progress in the game. They can compare the results to previous sessions and results of other users.
Rewards	The users are rewarded for achieving goals. Rewards support the motivation of the users and create challenges.

Table 4.3: Basic Heuristics [Molich and Nielsen, 1990] [Nielsen and Molich, 1990] [Korhonen and Koivisto, 2006].

For the *Heuristic Evaluation*, the design of the user interface can be available in form of verbal description, paper-mockup, working prototype or a running system. The evaluators have to work independently from each other. At the end, the problems found by the evaluators are collected. The problems get weighted with the average severity rate over all rates given by the evaluators. [Nielsen and Molich, 1990]

The advantages and disadvantages of the *Heuristic Evaluation* are:

- The evaluation is easy to proceed with low costs.
- The technique is intuitive.
- It can be applied during the development process. So problems are identified early in the process and can be eliminated.
- The method identifies major and minor problems.
- The *Heuristic Evaluation* can ignore domain specific problems.

4.1.2 Adaption for Underage Participants

The *Heuristic Evaluation* method uses experts to assess problems with user interfaces. The findings of this evaluation are problems in the applications. For example, application designs for children should be evaluated also by children. The problems found by adult evaluators are different to underage evaluators because there is a discrepancy in the focus on what is good or bad in an interface for different age groups. [MacFarlane and Pasiali, 2005]

The concept of *Heuristic Evaluation* has to be extended to fit the needs of underage users. Also the results of the evaluation depend on the knowledge, expertise, and skills of the evaluators. The heuristic introduced by [Nielsen and Molich, 1990] can be adapted easily. The heuristics have to be defined in detail to meet the requirements of *Heuristic Evaluation* with underage participants. Additionally to the heuristics mentioned before, the heuristics in Table 4.4 are applicable. [Alsumait and Al-Osaimi, 2009]

Heuristic Name	Description
Attractive Screen Layout	The screen layout is efficient, readable, memorable and attractive to children.
Appropriate Hardware Devices	The devices used for the evaluation are suitable for the age group of the participants. So the participants do not need to learn special skills to use the device (for example the users know how to handle a smart phone).
Challenges	The underage participants should be able to solve the tasks and reach the goals. The users do not get frustrated when performing actions in the software.
Fantasy	The software uses the imagination of the users. So there is an individual interpretation of the context. It also supports the engaging of the participants to the application.
Curiosity	The software includes surprises, humor, and topics interesting for the users. So the underage participants are intended to be curious what happens next.

Table 4.4: Basic Heuristics for Underage Participants. [Alsumait and Al-Osaimi, 2009].

4.2 Cognitive Walkthrough

The evaluation method presented in this section focuses on providing exploratory learning within the user interface, and it is called *Cognitive Walkthrough*. This method assesses the ability of the system to guide an untrained user. Therefore, a detailed description of the users and the application is needed. So an expert walks through the application from the perspective of the described user. This evaluation method and the adaption for children are presented in the next sections. [Mano and Campos, 2006]

4.2.1 Basic Concepts

The evaluation method is a usability inspection method. The *Cognitive Walkthrough* method analyzes the user interface under the aspect of exploratory learning. There is no previous training for the use of the application. The evaluation method is based on the theory of exploratory learning from [Polson and Lewis, 1990]. Before starting with this technique some factors have to be defined. First, a general description of the users and their knowledge has to be set. The next factor to be defined is a description of one or more representative tasks. The last preparation is a list of the correct actions to complete the tasks, which were defined before. [Polson and Lewis, 1990] [Lewis et al., 1990] [Mano and Campos, 2006]

For the execution of a *Cognitive Walkthrough* the system description or prototype has to include a complete navigation. The tasks defined are executed by usability experts during the evaluation under the point of view of the described typical user. For example the procedure simulates the steps executed by the user with no previous experience about the interface. During the evaluation process an evaluator tries to capture some aspects about the behavior of the users, like ease of access, system response, and goals. The developers ask the experts several questions on how easy the actions could be found and executed in the interface for the evaluation of the ease of success. Furthermore, questions about the system response and its adequacy are brought to the developers. So with the *Cognitive Walkthrough* the discrepancies between the developers view and the users view on a user interface are exposed. An example form for the evaluation with *Cognitive Walkthrough* is described in Appendix A for one task. The questions in the form have to be ranked according to the percentage of users who are expected to have problems with the interface. [Lewis et al., 1990] [Mano and Campos, 2006] [Rieman et al., 1995]

The advantages and disadvantages of the *Cognitive Walkthrough* are:

- The technique identifies task-oriented problems.
- The method supports the finding of user goals and assumptions.
- The *Cognitive Walkthrough* can be applied during the development process. So problems are identified early in the process and can be eliminated.
- The application is not tested by actual users.
- The technique may ignore general and recurring problems.

4.2.2 Adaption for Underage Participants

The most important factor for the execution of a *Cognitive Walkthrough* for a young target group like children and teenagers is the users description. The challenging part for the evaluators is to put themselves into the mind of an underage test user. So the evaluators identify problems with the interface which are relevant for the target group.

4.3 System Usability Scale

This method is a simple approach to assess the usability of an application and is called *System Usability Scale*. In the following the basic concepts and the adaption for underage participants are discussed in detail.

4.3.1 Basic Concepts

The *System Usability Scale* (*SUS*) is a simple approach to assess the usability of a system. The usability is dependent on its definition. As suggested in the ISO standard *ISO 9241-11*, a usability measurement should cover the effectiveness (the quality of completing tasks in the system), the efficiency (the amount of resources consumed by performing tasks) and the satisfaction (subjective perception of the user). The measurement for the *SUS* is processed with a ten item scale. With these ten items, an overview of subjective measurements is created. Please see Appendix B for a template of the scale. [Brooke, 1996]

The *SUS* is used after the user had the chance to test the system, which is evaluated. So the immediate response of the user can be captured with the scale. The *SUS* result is a comparable score between 0 and 100. Therefore, each item in the scale gets a score between 0 and 4. The item score with odd numbers (1,3,5,7,9) starts with 0 on the left side and ends with 4 on the right sight. The item score with even numbers (2, 4, 6, 8, 10) starts with 4 on the left side and ends with 0 on the right sight. The sum of all item scores is multiplied with the factor 2.5 to get an overall value. [Brooke, 1996]

The advantages and disadvantages of the *System Usability Scale* are:

- The approach is simple and easy doable.
- The results give a clear statement about usable or not usable.
- The scores should not be interpreted as a percentage value. It is necessary to normalize the score for a good interpretation.
- The *SUS* only concentrates on one scale, the ease of use of an application.

4.3.2 Adaption for Underage Participants

The use of the *SUS* with a young target group, like children and teenagers, needs a modification of the questions in the questionnaire. So the target users can understand and evaluate the application but the content of the questions stays the same. Therefore, additional explanations have to be provided. This can be done by a facilitator during the evaluation or in a written form directly on the questionnaire.

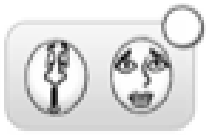
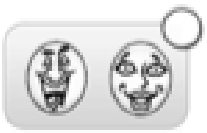
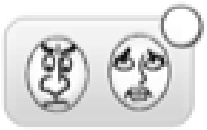
4.4 Emocards

The decision of a user about a good or bad software is not only a rational one. Human beings are driven by emotion and these emotions have effects on their behavior. Therefore, this aspect is also important regarding the usage of user interfaces. The challenge is to collect information about the emotions users have. *Emocards* are a possibility to measure emotions. It is a non verbal approach. It can be executed with low costs and can be easily combined with usability testing. This evaluation method and the adaption for underage participants are discussed in the following sections. [Agarwal and Meyer, 2009]

4.4.1 Basic Concepts

Measuring emotions in a usability test is a difficult problem which can be assessed with the help of *Emocards*. *Emocards* are a tool to measure emotions without any verbal actions by the user. The assessment can be done quickly with low costs and the faces do not need explanation. An *Emocard* consists of sixteen cartoon-like faces. On one card, there is a male as well as a female face and they represent distinct emotions. The faces are a combination out of the dimensions of pleasure and arousal, as presented in Figure 4.1 and Table 4.5. A desirable result for a user interface would be a high value of pleasure and a high value in arousal. The test users get to fill out an *Emocard* right after finishing a task. Furthermore, there is no written explanation on the *Emocard*. So the users do not know the exact meaning of the faces. Therefore, the users have to choose one of the faces which fits their emotional state after executing each task best. [Desmet, 2000] [Agarwal and Meyer, 2009] [Stickel et al., 2011]

Figure 4.1: Emocard [Agarwal and Meyer, 2009].

Cartoon Faces	Emotion
	The faces illustrate the excited emotional state. It is the highest value in arousal with a neutral value of pleasure after the use of the application.
	The faces show the excited pleasant emotional state. So the user is happy and calm after using the application.
	The faces symbolize the average pleasant state. This is the highest value in pleasure with a neutral arousal.
	The faces illustrate the calm pleasant emotional state. So the user is relaxed after the use of the application.
	The faces show the calm emotional state. It illustrates the lowest value in arousal with a neutral value of pleasure.
	The faces illustrate the calm unpleasant emotional state. This means the user is bored and frustrated after the use of the application.
	The faces show the average unpleasant emotional state. This is the lowest value of pleasure with a neutral value of arousal. So the experience of using the application is stressful and upsetting.

Cartoon Faces	Emotion
	The faces symbolize the excited unpleasant emotional state. This means the user is stressed and tense after using the application.

Table 4.5: Emotions in an *Emocard* [Agarwal and Meyer, 2009] [Desmet, 2000] [Stickel et al., 2011].

The advantages and disadvantages of the *Emocards* are:

- The approach is easy and can be executed with low costs.
- Only emotions are measured. So there is just an indirect evaluation of the interface.
- It is a non verbal approach, so the test users do not have to talk.
- The results are a value on a two dimensional scale regarding pleasure and arousal.
- The cartoon faces can be misinterpreted by the users.

4.4.2 Adaption for Underage Participants

The adaption for a young target group like children and teenagers is easily done. The cartoon faces are easy to understand for every age group. There is always the possibility to create faces which are more appealing to the target group. So there is less misunderstanding in the evaluation and the results are more significant.

4.5 A/B Test

The testing method called *A/B Test* is used in different fields besides computer science (e.g. medicine, agriculture, and advertising). The amount of data, which is gained during testing, has to be analyzed. The concept of the testing method and the analysis is presented in the following chapters. [Crook et al., 2009]

4.5.1 Basic Concepts

A simple form of controlled experiments are *A/B Tests*. In the test, two variants are presented to a set of users. A randomly generated subset of users gets variant A and the other subset gets variant B. In relation to tutorials, an example for an *A/B Test* would be to give one group of users the application with a tutorial and the other one without a tutorial. A high amount of data is generated with this testing method to compare the results. Therefore, an evaluation criterion (e.g. error rate, efficiency) has to be defined. This criterion is the base for the comparison. An important factor for the evaluation is the number of test users. In most cases, an *A/B Test* needs about 16-20 test users to get significant results. The results get more significant with even more test users. [Crook et al., 2009]

An *A/B Test* can be designed in two different ways. The first approach uses independent measures. This approach is also known as *Between Groups Experiments*. Two groups of test users get the same tasks but uses different interfaces to solve the tasks. The second approach uses repeated measures. This approach is also known as *Within Groups Experiments*. Therefore, only one group of test users is needed. The test users perform the same task on both variants of the user interface. For example some users perform the tasks first on variant A and then on variant B. The other users first perform the tasks on variant B and then on variant A. [Andrews, 2014]

The advantages and disadvantages of an *A/B Test* are:

- A lot of objective and quantitative data is collected.
- Different designs are compared to each other.
- The number of test users has to be significant for the statistically analysis.
- The results do not include information about the reasons why things are not working.
- The facilitators have to be experts.

4.5.2 Adaption for Underage Participants

There is no need for special modification of this method. The evaluation method of *A/B Tests* can be applied to different target groups.

4.6 Thinking Aloud Method

The *Thinking Aloud Method* gives a group of test users typical tasks to proceed with the application under test. During the test, the users have to bring their thoughts into speech. So direct feedback on the interface is produced. [van Someren et al., 1994]

The basic concepts and an adaption for underage participants are described in the following sections.

4.6.1 Basic Concepts

The *Thinking Aloud Method* is founded in the field of psychological research. The method was developed on top of the older introspection method. The idea behind the method is to observe the actions the observant is aware of. So for the *Thinking Aloud Method* the users under test have to verbalize their actions, feelings, and experience with the software. The challenge of the facilitators is to observe and promote the user with speaking out their thoughts. So the users talk about the current actions, readings, confusion, decisions, and questions which may appear. The main purpose is to collect data about the user interface. [van Someren et al., 1994]

The advantages and disadvantages of the *Thinking Aloud Method* are:

- The method detects many usability problems.
- The method uncovers the reason for the problems.
- A small number of test users are applicable (3 to 5 users).
- The method can be used early in the development process.
- The speed of the users is decreased by the method.
- The behavior of the users can be changed by the process.
- No data about performance is collected.
- It is time consuming.

4.6.2 Adaption for Underage Participants

For the use of the *Thinking Aloud Method* with children and teenagers, the basic concepts can be used. The challenge with young users is to motivate them to speak about their actions. Also during the testing, the facilitators have to assure constant speaking of the users. So for example with quiet children the facilitators have to ask questions to get the necessary information to the interface directly, instead of only asking to speak out their thoughts. Usually, underage participants do not have any experience with usability testing or the *Thinking Aloud Method*. So prior to the test, the test participants should get an introduction and an example test to get comfortable with the method. [van Kesteren et al., 2003]

4.7 Metrics

There are several metrics which can be implemented into a usability evaluation. A selection of metrics can be found in Table 4.6. Depending on the purpose of the evaluation, the metrics are implemented.

Metric	**Description**
Error Rate	Three types of errors are distinguished (skipped steps, incorrect selections, incorrect actions). For every type, a rate is calculated.
Elapsed Time	The measurement starts when the user opens a part of the application (for example the tutorial) and ends with finishing it.
Number of Requests for Help	Every help request to the facilitator is counted.
Task Completion Rate	Percentage of optimal task completion with help, e.g. a manual or a tutorial.
Task Completion Rate without Help	Percentage of task completion without help.

Metric	Description
Task Completion	Ability of task completion within a specific time range.
Error Decrease	Decrease within a time interval of the errors made by the user.
Overall Completion Time	Time needed for successful task completion.
Completion Time within a Time Interval	Task completion for specific tasks within a time interval.
Quality of Work	Quality of work performed during a task, as scored by judges.
Success rate	The success rate of commands after being trained.
Commands Increase	The increase in commands used over certain time interval.
Complexity Increase	The increase in complexity of commands over time interval.
Known Commands	The percentage of commands known by the user.
Used Commands	The percentage of commands used by the user.
Think Time Decrease	The decrease in average think times over certain time interval.
User Comments	The number of learnability related user comments.
Help Commands	The decrease in help commands used over certain time interval.
System Feedback	The metric covers the availability and quality of the system feedback.
Consistency	The metric covers the degree of consistency within the user interface.
Performance	Efficiency of completing the tasks.
User Like	This metric is a subjective value for each user and describes the general appeal of the application.
Internationalization	The ability to provide the user interface in different languages. So the applications have a bigger potential user group.

Table 4.6: Usability Metrics [Kelleher and Pausch, 2005] [Grossman et al., 2009] [Harrison, 1995] [Wong et al., 2003].

4.8 Applying for a Tutorial

In the future a usability test for the implemented tutorial is planned. Here, an approach is presented which considers the findings of this thesis. At the beginning of the usability test a simple questionnaire is filled out by the test persons. In this questionnaire, person specific data is collected. Therefore, information about age, sex, education, experience with mobile devices, and experiences with usability tests are gathered. In Appendix C, a questionnaire for the background information is presented.

Furthermore, the *Thinking Aloud Method* is combined with an *A/B Test*. The results of the test with and without a tutorial are going to be compared. So we can make a statement about the effects of the tutorial of the application. The experiment contains four aspects which were presented earlier in this work. These aspects are the presence of a tutorial, the context sensitivity, the freedom of use, and the availability of help. The first variable, presence of a tutorial, is easy to realize with an *A/B Test*. One group gets *Pocket Code* with a tutorial and another without it. So the users group without a tutorial are forced to learn the usage via trial and error. The second variable, context sensitivity, evaluates the retention of the users. Context sensitivity means that the tutorial describes for example one feature shortly and afterwards the user has to try it out. The opposite would be to give the users a lot of information until they start working on their own. The third variable, freedom of use, evaluates the impact on the user by restricting the available actions. The last variable, availability of help, is to evaluate if the users would search for help to solve problems. After every task the test persons are confronted with an *Emocard*. So the user defines the emotional state after the specific task. At the end of the test, the test persons have to fill out the *System Usability Scale*. Therefore, one value for the overall usability is identified. This combination of methods can be executed with a future tutorial. In the following section, the results of a previous evaluation is presented. This evaluation is based on the prototype described before.

4.8.1 Results from Previous Evaluation

In the past, the members of the *Catrobat* project already executed a usability test with the *Thinking Aloud Method*. Background information, specific questions about *Pocket Code*, and the opinion of the participants were collected besides the usability issues. The evaluation also included a prototype of a tutorial. The prototype was not implemented in *Pocket Code*. The facilitators used a flash animation for the simulation of a tutorial in *Pocket Code*. The purpose was to eliminate usability problems in the design of the tutorial. A detailed analysis of the results of the evaluation are not finished until the end of this thesis.

The feedback of the participants was positive. Therefore, the questionnaire about the content of *Pocket Code* showed that the participants memorized the information which was presented in the tutorial. Also the participants articulated the usefulness of the tutorial. Furthermore, the major part of the test users felt to be able to create programs on their own in *Pocket Code*. The assumption of too much text of the tutor was confirmed. The most important lessons learned from the evaluation were the preparation of clearly formulated texts for the tutor and the inclusion of a *Back* button in the tutorial. So there is a navigation available for the users in the tutorial. Furthermore, the design has to be more appealing. Critical statements on the design were desired. Therefore, these points can be included in further development of the tutorial.

4.9 Chapter Summary

In this chapter different evaluation methodologies were discussed. In detail the methodologies of *Heuristic Evaluation*, *Cognitive Walkthrough*, *System Usability Scale*, *Emocards*, *A/B Test*, *Thinking Aloud Method*, and measurable metrics were explained. Furthermore, the adaption of these methods for children and teenagers were discussed.

The *Heuristic Evaluation* method evaluates a user interface under predefined usability heuristics. The evaluation is processed by usability experts. The result of the evaluation is positive and negative feedback for the user interface tested. The method is easy and cheap to execute but there is a possibility that the method does not identify domain specific problems.

The *Cognitive Walkthrough* method evaluates the ability to guide the user through the application. Therefore, a detailed description of the tasks and the users have to be defined. During the evaluation, aspects like ease of use and system response are assessed. So tasks oriented problems are identified.

The *System Usability Scale* method evaluates a user interface with a ten item questionnaire. Each item gets a score. The sum of all scores is multiplied by a factor and results in an overall value. So the result is a clear statement about the usability of the user interface.

The *Emocards* method measures the aspect of emotions of the users during handling the user interface under test. Therefore, the users have to identify the emotional state after every task.

The *Thinking Aloud Method* motivates the users to speak out their thoughts during the usage of the user interface under test. So direct feedback is created.

Measurable metrics can be implemented in a usability test additionally. Therefore, problems with the user interface are exposed and documented with concrete numbers. Examples for such metrics are error rate, task completion time, and number of used commands.

At the end of the chapter, a proposal for the implementation of a usability evaluation is presented. Besides, a combination out of *Thinking Aloud* and *A/B Tests* is created. Also after every task the emotions should be measured with *Emocards*. In the evaluation recommendation a *System Usability Scale* after the experiment is included. Furthermore, results of a previous evaluation were presented.

Chapter 5

Conclusion and Future Work

The usability of a software is an important factor to measure regularly during the developing process. Therefore, a higher user engagement can be achieved. Good usability takes a positive effect on the impression, an application makes on the user group. This assumption is also applicable on applications for mobile devices. Complex applications like *Pocket Code* need some time for the users to get familiar with the workflow. So a tutorial can support the starting phase. This thesis identifies different aspects and challenges of the development of a tutorial for a mobile device using the example of the *Android* application *Pocket Code*.

As a result of the research, different approaches for a tutorial were created (guided tutorial and a tooltip system). Also the relation to the term *Gamification* is examined and the benefits have influenced the design decisions. Furthermore, there is also a relation to mobile learning. In considering different learning phases the tutorial can be optimized. This concludes in the results that there are no exact definitions or guidelines on how to create a good tutorial. A decision on which approach is better for an application can only be made with the help of an evaluation. There are several evaluation methodologies, some of them were presented in this thesis. The advantages and disadvantages of every method have to be deliberated whether it can deliver useful results or not. Here, an approach for a usability evaluation was put together. So a combination of methods is chosen for the evaluation.

The approach can be used for the *Catrobat Project* for further evaluations. Furthermore, evaluations have to take place several times during the developing process within appropriate intervals. So changes are implemented early in the development stage. An application for an underage target group implemented by adults can build up a gap between the expectations of the young users and the adult developers. So to close this gap it is very important to question the target group and get valuable feedback to improve the tutorial. In the best case, this involvement supports the success and the acceptance of the application and its tutorial.

Appendix A

Cognitive Walkthrough Form

1. Description of the immediate goal of the user.

2. Description of the next action the user should take.

   (a) Is the availability of the action obvious? Why/why not?

   (b) Is the relation of the goal with the action obvious? Why/why not?

3. How will the user process the description of the action?

   (a) Are there any problems occurring? Why/why not?

4. How will the user associate the description with the action?

   (a) Are there any problems occurring? Why/why not?

5. Are the other actions available less appropriate? For each, why/why not?

6. How will the user perform the action?

   (a) Are there any problems occurring? Why/why not?

7. Are there occurring timeouts? If yes, how much time elapsed until time-out? Why/why not?

8. While executing the action, how was the system response?

   (a) How far got the progress towards the goal? Why/why not?

   (b) Can the user access the needed information in the system response? Why/why not?

9. Is the user able to form a modified goal? If any, describe it.

   (a) Is it obvious that the goal should change? Why/ why not?

   (b) If the task is completed, is the completion obvious? Why/why not?

Appendix B

System Usability Scale Template

	Strongly disagree				Strongly agree
1. I think that I would like to use the system frequently.	1	2	3	4	5
2. I found the system unnecessarily complex.	1	2	3	4	5
3. I thought the system was easy to use.	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system.	1	2	3	4	5
5. I found the various functions in this system were well integrated.	1	2	3	4	5
6. I thought there was too much inconsistency in this system.	1	2	3	4	5

7. I would imagine that most people would learn to use this system very quickly.

1	2	3	4	5

8. I found the system very cumbersome to use.

1	2	3	4	5

9. I felt very confident using the system.

1	2	3	4	5

10. I needed to learn a lot of things before I could get going with this system.

1	2	3	4	5

Appendix C

Background Questionnaire

General Information

Gender:

Age:

Education

School:

Grade:

Use of Mobile Devices

Do you have your own mobile devices (smart phone, tablet)?

How long do you use the mobile devices per day?

Which operating system is on your mobile device (Android, Windows, iOS)?

Experience with Usability Tests

Have you participated in a usability test before?

If yes, what kind of study was that?

Bibliography

[Adams, 2011] Adams, E. (2011). The designers notebook: Eight ways to make a bad tutorial. `http://www.gamasutra.com/view/feature/134774/the_designers_notebook_eight_.php`. Last accessed on 2014-03-03.

[Agarwal and Meyer, 2009] Agarwal, A. and Meyer, A. (2009). Beyound usability: Evaluating emotional response as an integral part of the user experience. *CHI EA '09 CHI '09 Extended Abstracts on Human Factors in Computing Systems, Pages 2919-2930, New York, USA.*

[Aimeur and Frasson, 1996] Aimeur, E. and Frasson, C. (1996). Analyzing a new learning strategy according to different knowledge levels. *Computers and Education, Volume 27, Issue 2, September 1996, Pages 115–127.*

[Alice, 2014] Alice (2014). Alice. `http://www.alice.org`. Last accessed on 2014-03-03.

[Alsumait and Al-Osaimi, 2009] Alsumait, A. and Al-Osaimi, A. (2009). Usability heuristics evaluation for child e-learning applications. *iiWAS 09 Proceedings of the 11th International Conference on Information Integration and Web-based Applications and Services, Pages 425-430.*

[Ames, 2001] Ames, A. L. (2001). Just what they need, just when they need it: An introduction to embedded assistance. *SIGDOC 01, October 21-24, 2001, Santa Fe, New Mexico, USA.*

[Andersen et al., 2012] Andersen, E., O'Rourke, E., Liu, Y.-E., Snider, R., Lowdermilk, J., Truong, D., Cooper, S., and Popovic, Z. (2012). The impact of tutorials on games of varying complexity. *In CHI 12: Proceddings of the SIGCHI conference on Human Factors in computing systems, New York, NY, USA.*

[Andrews, 2014] Andrews, K. (2014). Human-computer interaction - course notes. `http://courses.iicm.tugraz.at/hci/hci.pdf`. Last accessed on 2014-03-03.

[AndroidActivities, 2014] AndroidActivities (2014). Android developers activities. `http://developer.android.com/guide/components/activities.html`. Last accessed on 2014-03-03.

[AndroidContentProviders, 2014] AndroidContentProviders (2014). Android developers content providers. `http://developer.android.com/guide/topics/providers/content-providers.html`. Last accessed on 2014-03-03.

[AndroidDevelopers, 2014] AndroidDevelopers (2014). Android developers. `http://developer.android.com/`. Last accessed on 2014-03-03.

[AndroidDevelopersDashboard, 2014] AndroidDevelopersDashboard (2014). Android developers dashboard. `http://developer.android.com/about/dashboards/index.html`. Last accessed on 2014-03-03.

[AndroidFundamentals, 2014] AndroidFundamentals (2014). Android developers fundamentals. `http://developer.android.com/guide/components/fundamentals.html`. Last accessed on 2014-03-03.

[AndroidProjectManagement, 2014] AndroidProjectManagement (2014). Android project management. `http://developer.android.com/tools/projects/index.html`. Last accessed on 2014-03-03.

[AndroidServices, 2014] AndroidServices (2014). Android developers services. `http://developer.android.com/guide/components/services.html`. Last accessed on 2014-03-03.

[Baecker, 2002] Baecker, R. (2002). Showing instead of telling. *Proceedings of ACM SIGDOC 2002, 10 -16.*

[Baecker et al., 1991] Baecker, R., Small, I., and Mander, R. (1991). Bringing icons to life. *Proceeding CHI '91 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1-6.*

[Balagtas-Fernandez and Hussmann, 2009] Balagtas-Fernandez, F. and Hussmann, H. (2009). A methodology and framework to simplify usability analysis of mobile applications. *IEEE/ACM International Conference on Automated Software Engineering 2009.*

[Barbosa and Silva, 2011] Barbosa, A. and Silva, F. (2011). Serious games - design and development of oxyblood. *ACE '11: Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology.*

[Baylor and Kim, 2005] Baylor, A. L. and Kim, Y. (2005). Simulating instructional roles through pedagogical agents. *International Journal of Artificial Intelligence in Education, Volume 15 Issue 2, April 2005, Pages 95-115.*

[Beck, 2000] Beck, K. (2000). *Extreme Programming Explained: Embrace Change First Edition.* Addison Wesley.

[Beck, 2002] Beck, K. (2002). *Test Driven Development by Example.* Addison Wesley.

[Beck and Andreas, 2004] Beck, K. and Andreas, C. (2004). *Extreme Programming Explained: Embrace Change Second Edition.* Addison Wesley.

[Brooke, 1996] Brooke, J. (1996). Sus - a quick and dirty usability scale. *Jordan and Thomas and Weerdmeester and McClelland. Usability Evaluation in Industry. London: Taylor and Francis.*

[Brusilovsky, 1998] Brusilovsky, P. (1998). Adaptive educational systems on the world-wide-web: A review of available technologies. *WWW-Based Tutoring Workshop at 4th International Conference on Intelligent Tutoring Systems, San Antonio.*

[Catrobat, 2014] Catrobat (2014). Catrobat. `http://developer.catrobat.org`. Last accessed on 2014-03-03.

[Chaffin and Barnes, 2010] Chaffin, A. and Barnes, T. (2010). Lessons from a course on serious games research and prototyping. *FDG 2010, June 19-21, Monterey, CA, USA.*

[Crook et al., 2009] Crook, T., Frasca, B., Kohavi, R., and Longbotham, R. (2009). Seven pitfalls to avoid when running controlled experiments on the web. *KDD 09, June 28-July 1, 2009, Paris, France, ACM.*

[DefinitionforMobileLearning, 2014] DefinitionforMobileLearning (2014). Mobile learning definition. `http://emerginged.com/adlmobile/definitions_9.html`. Last accessed on 2014-03-03.

[DeGani et al., 2010] DeGani, A., martin, G., Stead, G., and Wade, F. (2010). Mobile learning shareable content object reference model (m-scorm) limitations and challenges. *Tribal Education Ltd.*

[Desmet, 2000] Desmet, P. (2000). Emotion through expression: Designing mobile telephones with an emotional fit. *Report of Modeling the Evaluation Structure of KANSEI, 3, 1003-110.*

[Deterding et al., 2011] Deterding, S., Dixon, D., Khaled, R., and Nacke, L. (2011). From game design elements to gamefulness: Defining gamification. *MindTrek 11, September 28-30, 2011, Tampere, Finland.*

[Ellis and Dix, 2006] Ellis, G. and Dix, A. (2006). An explorative analysis of user evaluation studies in information visualisation. *Proceedings AVI 2006 Workshop on Beyond time and errors: novel evaluation methods for Information Visualization (BELIV '06), pages 1–7. ACM Press, Venice, Italy.*

[Gazzard, 2011] Gazzard, A. (2011). Unlocking the gameworld: The rewards of space and time in videogames. `http://gamestudies.org/1101/articles/gazzard_alison`. Last accessed on 2014-03-03.

[Gee, 2003] Gee, J. P. (2003). *What Video Games Have To Teach Us About Teaching and Literacy.* Palgrave Macmillan, New York.

[Gee, 2004] Gee, J. P. (2004). Learning by design: Games as learning machines. *Interactive Educational Multimeda, number 8, April 2004, pp 15-23.*

[Grabler et al., 2009] Grabler, F., Agrawala, M., Li, W., Dontcheva, M., and Igarashi, T. (2009). Generating photo manipulatioin tutorials by demonstration. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2009, Volume 28, Issue 3.*

[Grappiolo et al., 2011] Grappiolo, C., Cheong, Y.-G., Togelius, J., Khaled, R., and Yannakakis, G. N. (2011). Towards player adaptivity in a serious game for conflict resolution. *2011 Third International conference on Games and Virtual Worlds for Serious Applications.*

[Grossman et al., 2009] Grossman, T., Fitzmaurice, G., and Attar, R. (2009). A survey of software learnability: Metrics, methodologies and guidelines. *In CHI 09: Proceedings of the 27th international conference on Human Factors in Computing Systems, New York, NY, USA.*

[Hakulinen, 2011] Hakulinen, L. (2011). Using serious games in computer science education. *Koli Calling '11 Proceedings of the 11th Koli Calling International Conference on Computing Education Research, Pages 83-88.*

[Hallford and Hallford, 2001] Hallford, N. and Hallford, J. (2001). Swords and circuitry: A designers guide to computer role playing games. *Roseville, CA, Prime Publishing.*

[Hanna et al., 1997] Hanna, L., Risden, K., and Alexander, K. (1997). Guidelines for usability testing with children. *Magazine interactions Interactions Homepage archive Volume 4 Issue 5, Sept./Oct. 1997, Pages 9-14 ACM New York, NY, USA.*

[Haramundanis, 2001] Haramundanis, K. (2001). Learnability in information design. *SIGDOC 01, Santa Fe, New Mexico, USA.*

[Harrison, 1995] Harrison, S. M. (1995). A comparison of still, animated, or nonillustrated on-line help with written or spoken instructions in a graphical user interface. *CHI 95 Mosaic of Creativity.*

[ISO9241-11, 1998] ISO9241-11 (1998). Ergonomics requirements for office work with visual display terminals (vdt). *Part 11: Guidance on usability.*

[Juul, 2009] Juul, J. (2009). Fear of failing? the many meanings of difficulty in videogames. *In B. Perron and M.J.P. Wolf, The Video Game Theory Reader 2, New York, Routledge.*

[Kelleher and Pausch, 2005] Kelleher, C. and Pausch, R. (2005). Stencils-based tutorials: Design and evaluation. *CHI 2005, April 2-7, 2005, Portland, Oregon, USA.*

[Kirriemuir, 2002] Kirriemuir, J. (2002). The relevance of video games and gaming consoles to the higher and further education learning experience. *Techwatch Report TSW.*

[Kodu, 2014] Kodu (2014). Kodu. `http://fuse.microsoft.com/projects/kodu`. Last accessed on 2014-03-03.

[Korhonen and Koivisto, 2006] Korhonen, H. and Koivisto, E. M. I. (2006). Playability heuristics for mobile games. *MobileHCI 06, September 12-15, 2006, Helsinki, Finland.*

[Larkin, 2002] Larkin, S. (2002). Usability junior - how to run a successful usability test with children. `http://www.stcsig.org/usability/newsletter/0201_usabilityjr.html`. Last accessed on 2014-03-03.

[Lavin-Mera et al., 2009] Lavin-Mera, P., Torrente, J., Moreno-Ger, P., Vallejo-Pinto, J. A., and Fernandez-Manjon, B. (2009). Mobile game development for multiple devices in education. *iJET - Volume 4, Special Issue 2, IMCL 2009, October 2009.*

[Lewis et al., 1990] Lewis, C., Polson, P., Wharton, C., and Rieman, J. (1990). Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. *Proceeding CHI '90 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Pages 235-242.*

[Li et al., 2012] Li, W., Grossman, T., and Fitzmaurice, G. (2012). Gamicad: A gamified tutorial system for first time autocad users. *UIST 12, Cambridge, Massachusetts, USA.*

[MacFarlane and Pasiali, 2005] MacFarlane, S. and Pasiali, A. (2005). Adapting the heuristic evaluation method for use with children. *Proceedings of Interact Workshop on Child-Computer Interaction: Methodological Research, Rome, 28-31.*

[Malone, 1982] Malone, T. W. (1982). Heuristics for designing enjoyable user interfaces: Lessons from computer games. *ACM CHI 63-68.*

[Mano and Campos, 2006] Mano, A. and Campos, J. C. (2006). Cognitive walkthroughs in the evaluation of user interfaces for children. *In proceeding of: Interacção 2006, Actas da 2a. Conferência Nacional em Interacção Pessoa-Máquina.*

[Martin, 2009] Martin, R. C. (2009). *Clean Code: A Handbook of Agile Software Craftsmanship.* Pearson Education, Inc.

[McNamara et al., 2010] McNamara, D. S., Jackson, G. T., and Graesser, A. (2010). Intelligent tutoring and games (itag). *Gaming for Classroom-Based Learning: Digital Role Playing as a Motivator of Study, Chapter 3.*

[Mitchell and Savill-Smith, 2004] Mitchell, A. and Savill-Smith, C. (2004). The use of computer and video games for learning. *Learning and Skills Development Agency.*

[Mitchell and Smith, 2004] Mitchell, A. and Smith, C. S. (2004). *The use of computer and video games for learning.* The Learning and Skills Development Agency.

[Molich and Nielsen, 1990] Molich, R. and Nielsen, J. (1990). Improving a human-computer dialogue. *Communications of the ACM, Volume 33 Issue 3, March 1990 Pages 338-348.*

[Nielsen, 1994] Nielsen, J. (1994). *Usability Engineering.* Morgan Kaufmann.

[Nielsen and Molich, 1990] Nielsen, J. and Molich, R. (1990). Heuristic evaluation of user interfaces. *CHI '90 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Pages 249-256.*

[Paymans et al., 2004] Paymans, T. F., Lindenberg, J., and Neerinex, M. (2004). Usability trade-offs for adaptive user interfaces: Ease of use and learnability. *IUI 04, January 13-16, 2004, Madeira, Funchal, Portugal.*

[Polson and Lewis, 1990] Polson, P. and Lewis, C. (1990). Theory-based design for easily learned interfaces. *Human Computer Interaction, 5, 191-220.*

[Prensky, 2001] Prensky, M. (2001). *Digital Game-Based Learning.* McGraw-Hill: New York.

[Rieman et al., 1995] Rieman, J., Franzke, M., and Remiles, D. (1995). Usability evaluation with the cognitive walkthrough. *CHI Companion 95, Denver, Colorado, USA.*

[Scratch, 2014] Scratch (2014). Scratch. `http://scratch.mit.edu`. Last accessed on 2014-03-03.

[Silveira et al., 2001] Silveira, M. S., de Souza, C. S., and Barbosa, S. D. (2001). Semiotic engineering contributions for designing online help systems. *SIGDOC 01, October 21-24, 2001, Santa Fe, New Mexico, USA.*

[Stickel et al., 2011] Stickel, C., Holzinger, A., and Felfernig, A. (2011). Measuring emotions: Towards rapid and low cost methodologies. *RecSys'11 Workshop on Human Decision Making in Recommender Systems position statement, Chicago, IL, 2011.*

[Sweetser and Wyeth, 2005] Sweetser, P. and Wyeth, P. (2005). Gameflow: A model for evaluating player enjoyment in games. *ACM Computers in Entertainment, Vol. 3, No. 3, July 2005.*

[Talbot, 2012] Talbot, D. (2012). Given tablets but no teachers ethiopian children teach themselves. `http://www.technologyreview.com/news/506466/given-tablets-but-no-teachers-ethiopian-children-teach-themselves/`. Last accessed on 2014-03-03.

[Torrente et al., 2009] Torrente, J., Moreno-Ger, P., Fernandez-Manjon, B., and del Blanco, A. (2009). Game-like simulations for online adaptive learning: A case study. *In Edutainment 2009: The 4th International Conference on E-Learning and Games, Banff, Canada.*

[van Kesteren et al., 2003] van Kesteren, I., Bekker, M., Vermeeren, A., and Lloyd, P. (2003). Assessing usability evaluation methods on their effectiveness to elicit verbal comments from children subjects. *IDC '03 Proceedings of the 2003 conference on Interaction design and children, Pages 41 - 49, ACM New York, NY, USA.*

[van Someren et al., 1994] van Someren, M. W., Barnard, Y. F., and Sandberg, J. A. C. (1994). *The Thinking Aloud Method: A Practical Guide to Modelling Cognitive Processes.* Academic Press, London.

[Vanderlinden et al., 1988] Vanderlinden, G., Cockling, T., and McKita, M. (1988). Designing tutorials that help users learn through exploration. *Professional Communication Conference. IPCC 88 Conference Record. On the Edge: A Pacific Rim Conference on Professional Technical Communication.*

[von Ahn and Dabbish, 2008] von Ahn, L. and Dabbish, L. (2008). Designing games with a purpose. *Communications Of ACM 51, 8, 58-67.*

[Wario Ware D.I.Y, 2014] Wario Ware D.I.Y (2014). Wario ware d.i.y. `http://www.wariowarediy.com`. Last accessed on 2014-03-03.

[Wong et al., 2003] Wong, S. K., Nguyen, T. T., Chang, E., and Jayaratna, N. (2003). Usability metrics for e-learning. *OTM Workshops 2003, LNCS 2889, pp. 235-252, 2003, Springer Verlag Berlin Heidelberg.*

[Zhang and Adipat, 2005] Zhang, D. and Adipat, B. (2005). Challenges, methodologies, and issues in the usability testing of mobile applications. *International Journal of Human-Computer Interaction Volume 18, Issue 3, 2005.*