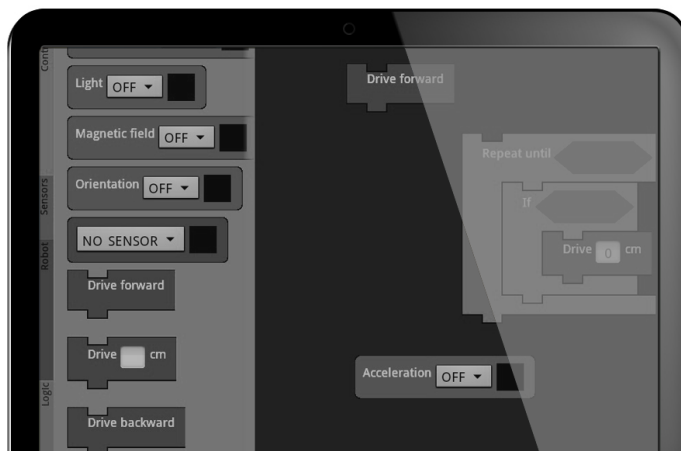


Development and evaluation of a visual development environment for tablets



25%



TABLETS

other

online

books

social

news

email

games

Advisor
Prof. Ulrik Schroeder

Second Advisor
Prof. Dr. Martina Ziefle

RWTHAACHEN
UNIVERSITY

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Hiermit versichere ich, dass ich diese Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Aachen, February 1, 2012

Alexander Friesen

Abstract

Block programming languages are well qualified to learn the basics of programming, because they are capable to prevent the user from making syntactical errors, if the logical constraints are used properly.

This work examines the question of whether the block programming language metaphor is suitable for touch interfaces, by creating a working prototype of a block programming language for Android OS driven tablets. The accentuation is made on the design and development, however in the last chapters of the current work initial evaluation on whether the users are more motivated by using block programming language on tablets, than on static desktop computers are made too.

The outcome of this project is a prototype, which has passed the first user tests and the evaluation of the prototypes interface and it's effect on the user tests participants. The participants showed a great motivation on the project idea, the prototype was graded as exciting during the final user study.

This work's follow-ups should overcome current prototypes issues, to enable the comparison of the building block programming metaphor with the existing desktop computer solutions.

Zusammenfassung

Programmiersprachen, in denen Befehle in Form von miteinander kombinierbaren Blöcken dargestellt werden, sind sehr gut dazu geeignet die Grundkonzepte der Programmierung zu erlernen. Dies ist der Fall, weil die Darstellung der Befehle als Blöcke – die Nutzung der „logical constraints“ ermöglichen um die syntaktischen Fehler durch den Nutzer zu verhindern.

Diese Arbeit untersucht die Frage, ob die Blockprogrammierung-Metapher für die Touchscreen Geräte geeignet ist. Dazu wird ein funktionsfähiger Prototyp einer Blockprogrammiersprache entwickelt. Als Plattform für die Entwicklung des Prototyps wurde Android OS ausgesucht. Obwohl die Betonung dieser Arbeit auf der Entwicklung des Prototyps liegt, wurde im Rahmen dieser Arbeit auch eine Studie durchgeführt, mit dem Ziel den Eindruck des Prototyps auf die Nutzer zu untersuchen.

Das Ergebnis dieser Arbeit ist ein funktionsfähiger Prototyp und die detaillierte Beschreibung dessen Gestaltungs- und Entwicklungsprozesses. Zusätzlich wurden die ersten Nutzerstudien durchgeführt, um die Antwort auf die Frage zu erlangen, ob die Nutzer mehr durch Blockprogrammiersprachen an mobilen Geräten motiviert werden, als durch Blockprogrammiersprachen an klassischen Desktop PCs. Die Teilnehmer der Studie zeigten sich hoch Motiviert, jedoch konnte vorerst kein Vergleich mit den existierenden Desktop Lösungen erfolgen, weil dazu der Prototyp weiterentwickelt werden sollte.

In der Zukunft sollten die bestehenden Design-Makel mit Hilfe von weiteren Testzyklen behoben werden um den Vergleich der Blockprogrammierung Metapher auf Tablets und Desktop PCs zu ermöglichen.

Contents

1 Introduction 7

1.1 Why mobile touch driven devices? 9

1.2 Block programming language on Tablets 10

1.3 Playful process 12

1.4 Objectives 13

1.5 Summary 14

2 ScratchTab definition 15

2.1 Summary 19

3 Interface Design 20

3.1 Definitions 20

3.2 Environment 22

3.3 Blocks 26

3.4 Gestures 29

3.5 Other design decisions 30

3.6 GUI prototype 31

3.7 Summary 35

4 Software architecture 36

4.1 ScratchTab as a framework 36

4.2 How to use the framework 38

4.3 Source code 40

4.4 Drawing 56

4.5 Lego NXT robot 60

4.6 Summary 63

5 Interaction 64

6 Evaluation 65

6.1 Expert review 65

6.2 User tests 65

6.3 User test results 69

7 Conclusion and Outlook 76

7.1 Conclusion 76

7.2 Outlook 78

8 Related work 81

8.1 Google App Inventor 81

8.2 Open Blocks 82

8.3 TouchDevelop 83

9 Appendices 84

9.1 Attached disc 84

10 Bibliography 85

1 Introduction

Motivation for the current project

This chapter introduces project's visual development environment metaphor - the block programming metaphor. A suitable target group is chosen for the project and the choice of Tablets as the project's platform is explained.

In the current year 2011, some digital skills seem to be required to fully participate in the daily trouble with new virtual media. According to [Monroy-Hernández and Resnick, 2008] there are different levels of participation: content consuming and content generating. Not all levels of participation seem to be accessible for the major part of computer users:

Why
programming?

- Due to the tendency towards usability, content consuming has become quite simple. As a result this level of participation is accessible by many people. Facebook statistics [*Facebook statistics*] illustrate this level well - there are 500 million active Facebook users right now.
- Generating and sharing non interactive content became feasible for the majority too, after WEB 2.0 at the latest. English Wikipedia for example has more than 3.6 millions of articles [*Wikipedia statistics*] and YouTube [*Youtube statistics*] quote 35 hours of uploaded video per minute.
- Creating interactive content is still problematic for most users as stated by [Resnick et al., 2009]. According to Resnick, programming is needed for this level of participation. However programming seem still to be an advantage of a highly skilled minority.

Scratch and
interactive
content creation

To improve the existing situation, "Scratch" - a special programming language which is described by [Resnick et al., 2009] was designed to make the first programming steps easier by working against typical novice programmer's problems:

- Scratch eliminates syntactical errors by using the constraints of a puzzle metaphor. Programming in Scratch is done by combining virtual puzzle blocks. It is only possible to combine puzzle blocks, which syntactically make sense.
- Since Scratch was strongly inspired by the behaviour of LEGO playing children, it inherited the support of different design approaches, which children use when they are constructing. In LEGO it is possible to combine some blocks and put them aside, somewhere on the table, until you need them again. Resnick observed this behaviour in his lab [Resnick et al., 2009] and noticed that this metaphor supported bottom-up design very well, because it allows to build many uncomplicated structures first, before combining these structures into one complicated system. Exactly the same approach is supported by Scratch: there is a virtual desktop, where pieces of code can lay around, without affecting any other programs, until the designer decides to integrate these pieces into his program (see Figure 1). This approach is called "bottom-up". Of course it is possible to design according to the "top-down" approach too: just by combining the bricks one after another. In a nutshell, Scratch supports different design approaches, which should help novice programmers to behave naturally.
- Further on, it is possible to do more complicated things (like game programming) in the same language. Scratch has all the features of a typical scripting language. It is only the aspect of object oriented programming, which is not yet supported, since neither the inheritance nor the polymorphism are implemented.

Measured by the 500.000 Scratch projects [Resnick et al., 2009] in 27 months, created by children and adults, one can say: Scratch is a great success as an easy programming language for learning the basic concepts of programming.

As a further development possibility, Scratch authors mentioned a Scratch version for mobile devices [Resnick et al., 2009]). This work is about the adaptation of Scratch for

tablets (tabs) – a mobile device which is gaining more and more popularity.



Figure 1: A part of the Scratch user interface. On the right side of this picture there is a program, put together by dragging puzzled blocks to the virtual desktop.

1.1 Why mobile touch driven devices?

Why should a mobile touch driven device maybe useful?

An implementation of Scratch on a mobile platform with a touch interface would provide a possibility to work on projects in the field more comfortably, since the mobile platform can be always used locally without additional tools but the own fingers.

The idea behind this project is to replace the mouse, which is the control device of the original Scratch, by a touch interface. On mobile devices, capable to run the original Scratch, there is often only a poor surrogate of a mouse (e.g. touch pads on laptops). A touch interface driven building block programming language would introduce comfortable programming by using the fingers as input devices.

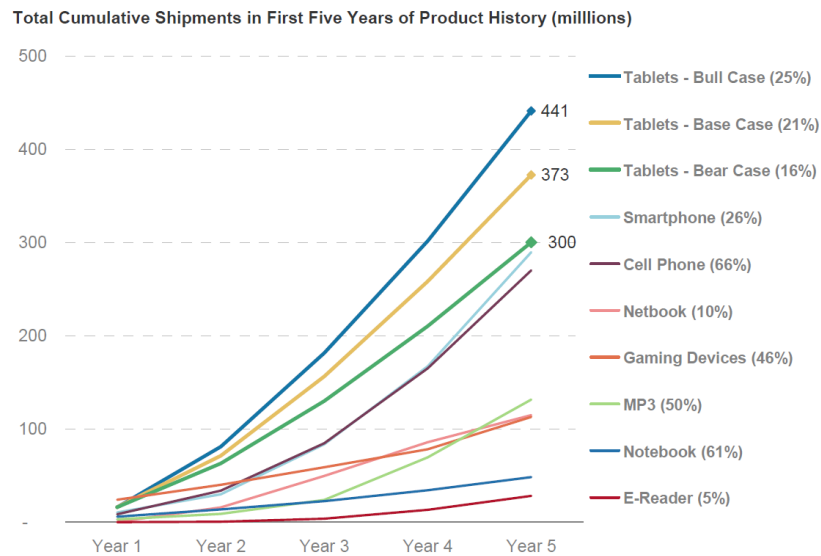


Figure 2: Tablet will be the best selling platform. Percentages represent current penetration rates for each device. Figures for bull, base, and bear case forecasts represent penetration rate in year five. For notebook, cell phones, and gaming devices, shipments are in early years of product history.
Source: Morgan Stanley Research, 2011 - [Stanley, 2011]

This change would support mobile programming projects in the field and allow concentrating on the projects themselves, getting immediate feedback on-site, without interrupting the creative flow by walking to a stationary PC.

A conceivable situation, in which it is advantageous to get immediate feedback during an activity could be the measuring of the own speed by using the mobile device's sensors. The advantage of a mobile approach would be the immediate feedback, so that the user can react on it directly. No try and evaluate cycles would be necessary.

1.2 Block programming language on Tablets

Why and how to implement a block programming language on tablets.

Tablets are a very promising platform. Among the devices listed on figure 2 tablets will probably be the best selling product within the next five years as stated by [Stanley, 2011]. For this reason alone, they should be considered as a platform for Scratch, given that tablets are suitable for a Scratch-like programming language.

Tablets will
become the best
selling device

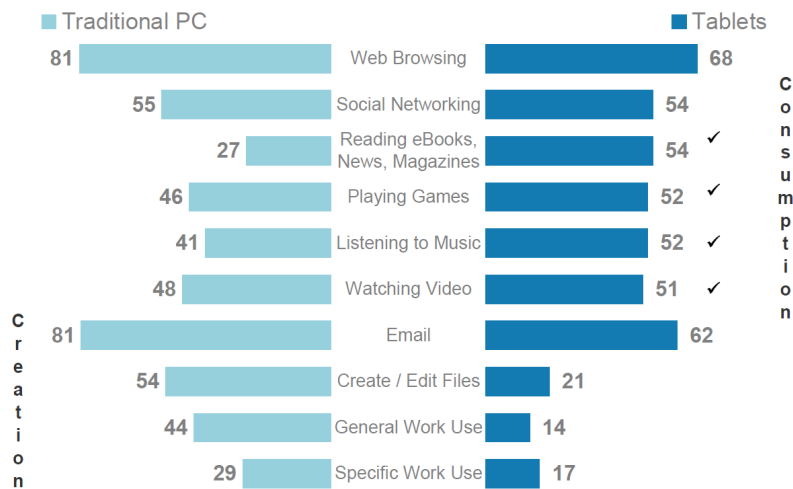


Figure 3: Tablets geared towards content consumption. Traditional PC is average of desktop and notebook. Only people, who stated to use their devices regularly for a concrete activity are considered here.

Source: Morgan Stanley Research, 2011 - [Stanley, 2011]

For the purpose of continuing Resnick's undertakings in programming popularization it would be helpful if programming could become an attractive activity on such a new and promising platform as tablet. For programming to make the leap from PCs to tabs the characteristics of activities should be considered, which have already made their way from the domain of PCs to the tablet-like devices. A statistical report by Morgan Stanley Research Global [Stanley, 2011] testifies: tablets score in content consumption and desktop PCs - in content creation, see figure 3.

What makes a tablet application attractive?

Apparently tablets' popular activities are browsing, listening music, watching video and playing games. Since the intention of the current project is the implementation of a programming language, which is a very interactive process, this project should mimic popular interactive tablet activities, but the most popular tablet activities are passive since tablets score in content consumption. The only long termed and interactive activity, which is popular on tablets according to [Stanley, 2011] on figure 3 or [AdMob, 2011] on figure 4 is gaming, so maybe a playful experience can help making a block programming language popular on tablets.

In particular, when comparing the properties of popular tablet activities with unpopular tablet activities in [Stanley, 2011] and

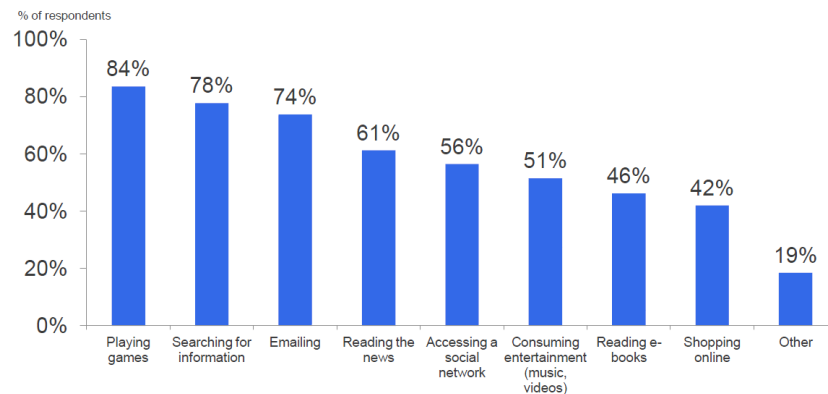


Figure 4: Select all the ways in which you use your tablet.
Source: Google AdMob Survey, 2011 - [AdMob, 2011]

[AdMob, 2011], the following should be kept in mind while designing a tab-programming language:

- Tablet programming language should reduce the active interactions to the minimum to resemble the content consuming applications.
- Also it should avoid the usage of actions like typing to avoid becoming unusable on tablets, which was the case with the editing-files-activity and the writing-emails-activity.

1.3 Playful process

How can Tab Scratch programming become a playful process?

Fortunately, Scratch's programming process, which this project will build on, already has a strong support for programming as a playful activity:

Playful process

"We have always been intrigued and inspired by the way children play and build with LEGO bricks. Given a box full of them, they immediately start tinkering, snapping together a few bricks, and the emerging structure then gives them new ideas. As they play and build, plans and goals evolve organically, along with the structures and stories. We wanted the process of programming in Scratch to have a similar feel. The Scratch grammar is based on a collection of graphical 'programming blocks' children snap together to create programs. As with LEGO bricks, connectors on the blocks suggest how they should be put together. Children

can start by simply tinkering with the bricks, snapping them together in different sequences and combinations to see what happens. There is none of the obscure syntax or punctuation of traditional programming languages. The floor is low and the experience playful."

[Resnick et al., 2009]

Therefore, to keep Scratches playful experience, we will use Scratches block programming metaphor on tablets.

1.4 Objectives

What is the goal of this project?

To sum up the upper elaborations, the main task of this work will be in designing a block programming language for tablets, implementing a prototype and to do the first evaluations of the block programming on a touch driven device.

Since, due to many reasons, a block programming language is not suitable for advanced users (e.g. because advanced users are arguably more effective, when using a keyboard interface, see [Raskin, 2000] or because in a block programming language every command takes some space, so that above a certain number of commands the needed block is too hard to find) the target group of this project will be the novice programmers. This fits in with Scratches intention of making the creation of interactive content more accessible, as described in the first paragraph of the current chapter.

Target group

The research question will be: is Scratches building block metaphor transferable to tablets and are possible user more motivated using a mobile device instead of a static desktop PC?

Research question

As described above an "in-the-field" activity is required to unfold the potential of the on-going prototype. For this reason this project's prototype will involve the possibility to control a LEGO Mindstorms¹ robot. This activity fits the requirements of being interactive and playful. Additionally pairing the robot with a mobile device enables "in-the-field" actions, as described above.

"ScratchTab" was chosen as the name for the current project.

¹<http://mindstorms.lego.com/>

1.5 Summary

A short overview of the chapter.

The current chapter focused on the choice of a suitable visual metaphor and a suitable, touch driven mobile device for the project. As a visual programming metaphor the block programming, as implemented by Scratch was chosen. Tablets were chosen as a platform.

The next chapter will define interface requirements for ScratchTab, as a preliminary stage to the system design.

2 ScratchTab definition

Definition of the system, definition of the objectives to be implemented by the system.

The last chapter the motivation for the current project, the choice of the platform and the metaphor were explained. Together with the metaphor, block programming language "Scratch" was introduced.

In this chapter the requirements of this project's prototype-interface will be defined by doing an analysis of Scratch. Scratch's interface sets the minimum requirement for this project's prototype, because it already has a wide audience (see [Resnick et al., 2009]) whose existing knowledge can be reused, if this project will pass the prototyping stage. At the same time the first steps towards adopting the interface for the tablet platform are made here.

No modes

Scratch is designed as a one window application. It avoids any kind of modes, which should be avoided according to [Raskin, 2000]. That's why ScratchTab will behave the same.

Tinkerability

The authors of Scratch mention tinkerability, see [Maloney et al., 2010]. By calling the Interface tinkerabile they mean the following features, which make Scratch learnable by the try and error strategy:

- Immediate feedback
- the possibility to disassemble the blocks during run-time, without stopping the script explicitly
- the possibility to run any piece of the the constructed script separately, up to the possibility to execute every single block.

ScratchTab should try to keep this tinkerability feeling.

Feedback

Scratch always provides feedback. Every operation like drag (figure 5a), errors (figure 5b), execution (figure 5c) provide visual feedback. Providing immediate information to the user, about the processes which currently happen within the program, is one of the most important rules for Scratch too.

On a touch interface the feedback should be given in an other way, than on desktop interfaces. Because the hand, which is always between the screen and the user when some action like dragging is done, blocks the view - the feedback should be shifted from the place of action. However the relation to the place of action should not be lost.

System has no explicit error state

In Scratch there is a high degree of freedom given to the user. The block-form constraint prevent the users from making syntactical errors by putting wrong blocks together, see "block type" on page 20 for details. Other errors can still occur during the block execution, caused by the semantic of the script (e.g. dividing by zero).

These errors are not corrected, but presented to the user, because they are caused by wrong thinking and not by obscure programming language definition. The user is expected to understand these errors and correct the wrong thinking.

Even occurred errors do not stop the whole system (which would be the case if it e.g. would produce an error popup). Instead the error only affects the stack, where it occurred. Occurred errors are presented by highlighting the affected blocks with a red background and highlighting the affected stack with a red frame (figure 5b).

Every stack represents a thread, so that one stack is independent of the other stacks. It can only affect other stacks

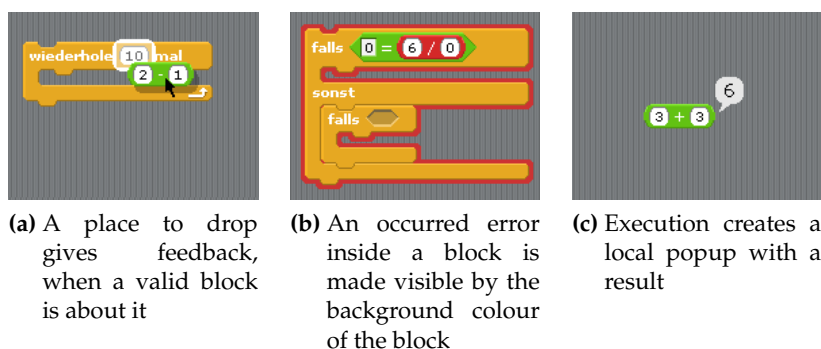


Figure 5: Scratch provides feedback during the program execution.

in a way, defined by the system e.g. by sending messages or by influencing paired devices like the NXT robot. Stack A should not stop another stack B, if an error occurs within stack A.

Block interaction

The blocks in Scratch are instantiated by dragging them from the pane on the left screen side to the workspace. The blocks can be deleted by dragging the block back to the block pane. The blocks are always gripped by clicking the left mouse button once and holding the mouse during the drag process.

On a touch interface such an interaction could cause problems, since the blocks have to be much bigger and since the finger is a much bigger pointing tool, than a cursor peak. Zoom can be a solution for this problem, see page 18

One tap interaction may cause conflicts because touch driven interfaces use less control widgets and many actions, which are done by widgets on a desktop interface implicate a tap too. (e.g. scrolling) If such a conflict occurs, a more unambiguous gesture may be required for the conflicted interactions. (for details see section 3.4)

Block groups

The blocks are separated into groups in Scratch. Each group is accessible through a special button as seen on figure 6.

This approach increases the consumed space with every new group added. On a small device like a tablet there is too little space to permanently give up so much of it for a widget to switch between block groups, see page 22 for details.



Figure 6: Block groups

Execution

In Scratch the blocks are executed by double clicking a block. Taking over the double click to the touch interface as the gesture to trigger the block execution can be a problem, because double tapping is much more difficult than double clicking with the mouse, the double clicking is done when the hand is laying down on the table and double tapping on a mobile device will have to be performed from different angles, with the hand being in the air, perhaps while the user is in an unstable position. Additionally, if a double tap goes wrong, the gesture will end up in a single tap which will unintentionally disassemble the block stack, making the execution unnecessary unsafe. More details about the execution gesture are presented on page 30.

Constraints

The input fields in different blocks allow different input in Scratch. E.g. mathematical blocks do not allow textual input. However these constraints are not visible for the user, they reveal themselves when the user tries to enter some data of a wrong type. The error reveals itself, when the data do not appear in the input field.

Soft keyboard allows preventive warning about constraints by manipulating the keyboard, so that only buttons with the allowed characters appear.

To sum up ScratchTab will implement the functionality, described above. It will provide executable, draggable blocks, sorted by their function.

2.1 Summary

A short overview of the chapter.

This chapter defined the minimum requirements to the ScratchTab's interface, by analysing of an established block programming language. Some initial adaptations to the touch interface were already done. However the complete user interface design with respect to the touch-driven user-computer interaction will be done in the next chapter.

3 Interface Design

The interface designer's point of view.

Previous chapter defined the requirements to ScratchTab's interface by analysing an established block programming language.

In this chapter the development of the interface is described. This chapter presents the interface designer's point of view to the ScratchTab development.

In our case the interface is limited by the device properties like size, number of available gestures, whereby in the case of gesture deficit there is a way out in combining gestures or introducing new. On the contrary the limited size must be respected and so the limited size should be the property which this ScratchTab's interface should be optimized for, with the highest priority.

If they do not break device limitations, the ScratchTab interface should be consistent with Scratch's behaviour, described in the previous chapter. This allows to use available knowledge of Scratch experienced users, which can be useful if this project will pass the prototyping stage.

3.1 Definitions

Introduction of ScratchTab terms

To define some common language, some basic terms are introduced here.

Def 1: Programming block type

The type of a block can be recognized on its shape. For a full list of possible shapes see page 40, possible types are: Command, Data, Boolean, Head. Block's shape is used as a physical constraint, so that it becomes possible to define which blocks can be combined together and which can not. Owing to this approach it is impossible to make syntactical errors because blocks cannot

Programming
block type

be combined in a erroneous way. The number of block types is limited to keep the combination possibilities manageable.

Def 2: Programming block group

Block's group is recognized on it's colour. Possible groups are: Control, Variables, Numbers, Logic, Sensors, Robot. The block's group announces the appointment of the command, which the block represents. For that the number of groups is not anchored in the language architecture and can easily be extended, e.g. if the ability of controlling a new device X was added to the system the commands related to the device X will probably format a new group.

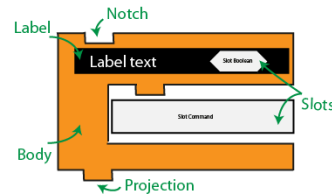


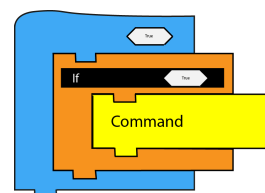
Figure 7: Programming block

Def 3: Programming block

Graphical blocks of different shapes and colours, which can represent commands, control structures, variables, logical expressions. The main part of the block is called body. The text inside of a block is wrapped by a container named label. See figure 7. The blocks are capable to form a stack have a notch at their top side and a projection at the bottom.

Def 4: Script or Command Stack

A sequential construction made of programming-blocks is a script or command stack. Executing a script is resulted in a sequential execution of all nested blocks. The active block is highlighted. See figure 8.



Command Stack

Def 5: Slot

A slot is an area inside of a programming block, where other programming blocks can be inserted. Every slot can contain blocks of a predefined type. Slot should react on users actions to signalize. Slots can have different states, to signalize to the user that a slot is capable to receive the chosen block or that the block is right above the slot and can be dropped into it.

Figure 8: Block stack alias script.

Slot

3.2 Environment

The ScratchTab's graphical user interface except of the blocks

The user interface for maintaining the programming-blocks consists of a workspace, a blockpane and a tool-bar (see figure 9). The workspace is a container, where users put programming blocks together into a program. The blockpane is a list of programming blocks, wrapped by a scrollable block-pane container where the blocks are sorted by their purposes (like Control, Variables, Sensors, Logic etc). The width of the blockpane is 5cm, which is sufficient to wrap the widest block, as determined by building a prototype, see page 31. Since a vertically and horizontally scrollable pane of this size did not cause any problems, the actual solution was accepted and no further alternatives were considered. In particular a resizable panel was not considered. The rest of the horizontal space is assigned to the workspace. The tool-bar is reserved for widgets to enable GUI control by the user (e.g. zoom, program execution speed).

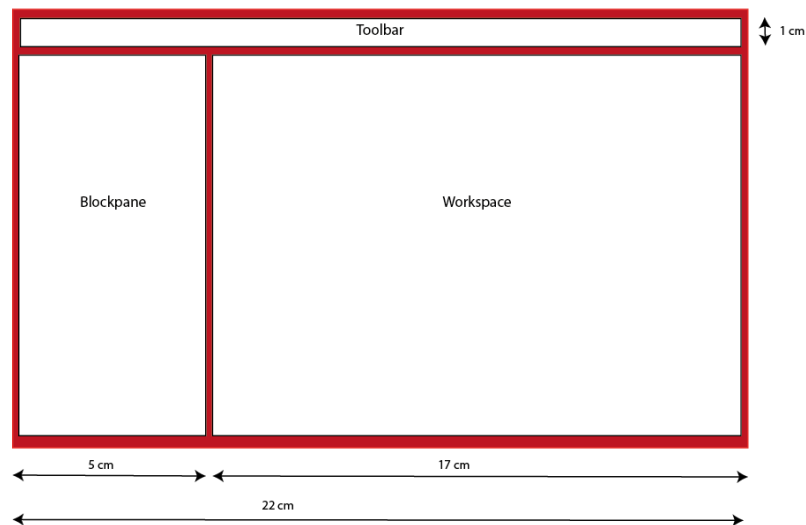


Figure 9: Graphical User Interface layout on the tablet which the prototype was tested on.

There are some general expectations, for the Graphical User Interface to fulfil.

- The Graphical User Interface(GUI) should correspond to the golden rules, as listed in [Shneiderman, 1998]. The golden rules are :

- consistency
 - shortcuts for experts
 - informative feedback
 - closed user-dialog
 - simple error handling
 - reversal of actions
 - locus of control
 - reduce memory load
- Putting every block on a workspace should be possible, even the blocks which won't work independently of other blocks (like blocks which represent boolean expressions). This is important for keeping user's impression of the workspace as a physical desktop, where everything can be put down everywhere, until user needs it. (as intended by Scratch [Resnick et al., 2009])
- Text entering should be avoided, spin buttons or scroll widgets should be used instead.

Workspace and pane ratio

The first decision is about the ratio between the block pane and the workspace. Since the workspace is the space where the most interactions will happen it should be maximized, which implies the minimization of the block pane. Because of a variety of different blocks and of the possibility to introduce new blocks in the panel - the optimum block panel width does not exist. The prototype which is described on page 31 has shown the width of 5 cm as sufficient.

Representation of block groups

The second decision is about organizing the blocks on the block pane. For the reasons described on page 18 introducing of a button for each block group is not recommendable. The main reason: the solution needs too much space, which is a rare resource in this project. Instead ScratchTab introduces a navigation widget to the left of the block pane, see figure 11a. The idea of this widget is to align all blocks vertically in the block pane. Since the the height of such an alignment exceeds the height of the device pane has to be scrollable. For immediate navigation to a concrete group the navigation stripe to the left of the block pane is given. Every point on the navigation stripe is mapped to a Y coordinate inside

of the block pane. The groups of blocks are present on the navigation stripe in form of colourful areas. A touch of the navigation stripe immediately scrolls the block pane to the mapped area.

To compare both solutions consider Fitt's law on figure 10 or [Fitts, 1992] for more details.

$$T = a + b \log_2 \left(2 \frac{D}{W} \right)$$

Figure 10: **T** is the amount of time required to complete the movement. **a** and **b** are values gained from direct observation. **D** is a measurement from the starting point to the end point (target object) **W** is the width of the target object

With this formula the time, which is needed to move the hand to an object with the width W , is calculated. According to Fitts's law objects with a larger width can be reached faster, because the pointing is easier. Especially the corners and the screen edges can be reached very fast, since the width of those objects can be assumed as infinite.

To compare two ways of block-group representation:

1. buttons seen on figure 6
2. and the navigation widget which is seen on figure 11a

Fitts law is used in this Project. The representation of each block group by a button makes the buttons shrink if the space for the whole-button widget is assumed constant or it makes the widget grow otherwise. In the first case the buttons will be increasingly more difficult to reach, the other case is not acceptable due to the deficit of space in the current project. The action sequence to reach a block inside of a particular group is the following: pointing the group button, touch the group button, pointing the block, touch the block.

On the other hand the solution depicted on figure 11a is placed along the left edge, which makes the navigation faster since the finger doesn't have to land on the widget, but has only to stripe the widget on the needed level. This virtually increases the width of the widget and decreases the access time according to Fitts's law. Another advantage of ScratchTab's approach is that every point on the ordinate is mapped to a point inside of the block panel. This means, that by striving the navigation panel on the

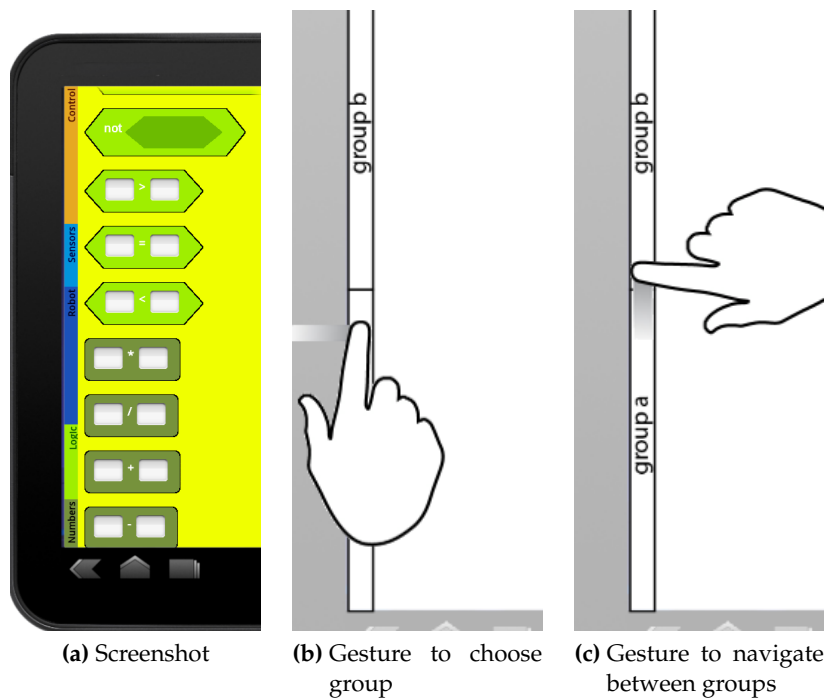


Figure 11: Block pane navigation widget

right level the user can directly navigate to the needed block inside the chosen group, so that when users hand is moved from left to right behind the navigation widget, the block appears directly under users finger. This reduces the "choose a block in a particular group" action by one pointing action, compared to the button approach. This saving is noticeable, since pointing one of the most time-consuming operations. (1.10 seconds according to the Keystroke-Level-Model ([Kieras, 2001])). This however requires some previous knowledge of blocks position within the group and can be seen as a shortcut for expert users, see [Shneiderman, 1998].

Additionally the user can navigate between the groups by sliding with the finger up and down (see picture 11c). The button solution requires pointing again to navigate between the groups.

Toolbar

The next thing is about introducing a toolbar where some basic interactions like start execution, stop execution can be represented as buttons and spending some valuable space as a consequence. The alternative would be giving up the toolbar and

Toolbar

implement all user interactions as gestures or displaying the buttons in form of overlays over the workspace.

The toolbar was introduced in ScratchTab because gestures, which are not always intuitive, will stay unrecognised if they are not visually presented as objects on the screen, which results in a perception gap, see seven stages of action in [Norman, 2002]. This decision was confirmed during the development by the following: during the development in one of the first versions of ScratchTab a double-tap gesture was used for the instantiation of new blocks on the block pane. Since the double tap gesture does not have any analogies in the real world and is not a part of usual user experience it is contra intuitive so many users had problems to recognize the necessary action for instantiating blocks. On the contrary no test person ever had problems with the buttons on the toolbar.

The solution with the overlay buttons was ignored, because this would make interactions with the workspace unnecessary dangerous, because the user would have to pay attention for not to unintentionally push a button above the workspace.

3.3 Blocks

Designing the blocks

In this part the block size and the art of nesting the blocks are explained.

Nesting the blocks

Scratch uses blocks of different width, which is perceived as messy by the users. This conclusion is a result of the following online survey: One hundred members of a German, technical forum "Maschboard"¹ were presented participated in the survey. The figure 15 was presented to the participants and three questions. The questions as they were asked and their English translation are presented on figures 12 13 14. The designations a,b,c apply to the matching part of figure 15. The numbers to the right of each letter indicate the amount of participants, which voted for the current version.

¹<http://www.maschboard.de/>

GER Welche Variante ist eurer Meinung nach am sympathischsten?

ENG Which version do you perceive as the most pleasant?

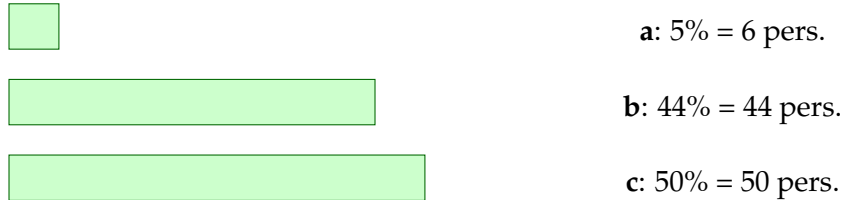


Figure 12: Question one

GER Welche Variante sieht akkurater aus als andere?

ENG Which version looks as the most accurate?

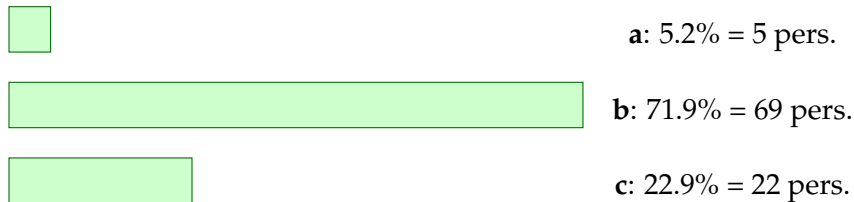


Figure 13: Question two

GER In welchem der drei Fälle kann man, an der rechten Seite eines Blockstapels, am besten erkennen – welches Blockende zu welchem Block gehört?

ENG In which of the three cases you can recognize the best, on the right side of the blockstack, - which block end belongs to which block?

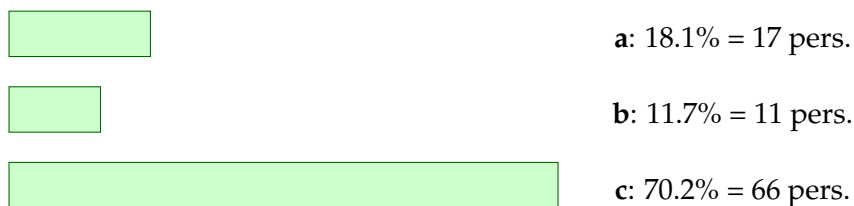


Figure 14: Question three

The survey indicates: people perceive the version on the figure 15c as most pleasant, so in order to maximize the likeability this way of nesting blocks would be the logical choice. However, this approach requires all blocks in the stack to be equally wide, which means that one large block can increase the whole stack dramatically and let it use much more space compared to the nesting approach on figure 15a. For this reason and because, as stated before, space is the most scarce resource in this project (limited by the hardware and for that cannot be extended) – the first nesting way is used in the current project. However if at any time in the future larger tablets should become available, then the blocks should be nested as seen on the figure 15c.

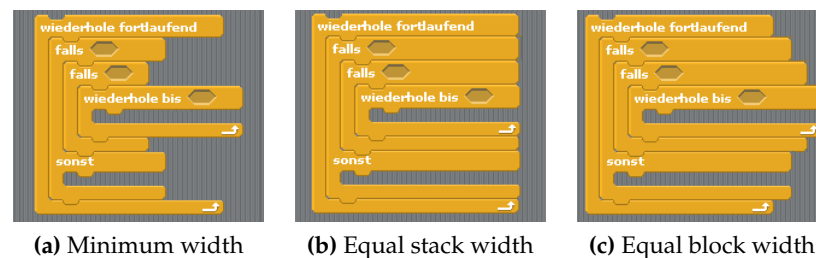


Figure 15: Three ways to nest the blocks where presented in the survey

Blocksize

As mentioned above, the greatest problem on mobile devices with a touch interface - is the lack of space. Determining the optimal object size, which can be touched easily and so minimizing the error rate - is critical for handling with the space problem. Unfortunately the optimal size for a touch interface object depends on too many factors (like screen density, device position, the amount of buttons on the screen, the test group etc.) so a general optimal touch-interface-button size seems non existent.

The table 1 lists papers, related to the search of a trade off between the pointing error rate and the button size.

The wide dispersion among the results confirms dependency of too many factors as that this question can be solved within of this project. For this reason the workspace with the programming blocks on it is zoomable in this project, so that every user can find his optimal zoom level. As a recommended minimum size for the touchable GUI objects - 1cm is used. For block layout details see Appendix 1.

Year	Device	Task	Keysize	Paper	
1990	Monitor	Using a Keyboard	20.0mm	land-on	[Gould et al., 1990]
1991	Monitor, ergonom. angle	Using a Keyboard	20.7mm	land-on	[Sears, 1991]
1991	Monitor, stabilized	Using a Keyboard	2.0mm	lift-off	[Sears et al., 1993]
2004	Monitor	Entering 10 digits	20.0mm	land-on	[Colle and Hiszem, 2004]
2007	Monitor	Using numeric keyboard	>10.0mm	land-on	[Schedlbauer, 2007]
2008	1 handed device	Pushing single button with thumb	> 4.0mm	land-on	[Parhi, Karlson, and Bederson, 2006]

Table 1: Software key size related papers

3.4 Gestures

Gesture related decisions

The interaction with the blocks on the workspace in reducible to simple touch gestures without conflicts. The input fields and dropdown widgets react on touch by letting the user enter some content. Touching the rest of the block results in a drag action.

Drag

Touch interactions within the block pane need more attention since the following conflicts occur: defining new blocks with unlimited width is possible in ScratchTab - the block pane is implemented as a vertically and horizontally scrollable container. Scrolling is done by touching the block pane and performing drag gestures. Dragging a block is done by touching a block inside the pane and performing drag gestures either which creates a conflict.

conflict

Scratch does not have this conflict, since it is controlled by a mouse, which provides a wider action space. On the figure 17 Scratches action diagram is seen. Dragging is done by clicking a block and pulling it somewhere, independently of whether the block is located in the sidebar or not, because scrolling is done by using the mouse wheel or the scrollbar.

ScratchTab's action diagram is seen on the figure 18. Because a touch driven interface allows fewer interactions, than a usual desktop computer's controls (no mouse wheel etc.) a unique double tap gesture was initially used to start a dragging process from the block pane. The experts review (see page 65) have shown, that the expected behaviour for a drag gesture performed on a

Action diagram

block, would be a start of a dragging process. This should happen, independently of whether the block is located in the block pane or on the workspace.

To resolve the conflict between the scrolling gesture inside the block pane and the block-dragging gesture inside the block pane, the tap gesture was splitted up into "Tap and drag right" and "Tap and drag into other directions".

"Tap and drag right" gesture is recognized, when a block is moved from the block pane to the right, toward the workspace. Because the movement is never performed straight towards the workspace - a tolerance of 45 degrees was introduced, see figure 20b. As soon as a movement in some other direction is performed - the gesture becomes a "Tap and drag into other directions" gesture until the finger is taken up, see figure 21d.

Now "Tap and drag right" is associated with dragging of programming blocks from the block pane to the workspace. Such a mapping is natural, because blocks are dragged from the block pane with the only intention of putting them onto the workspace. After the redesign the ScratchTab's action diagram looks as depicted on the figure 19.

To make the separation complete the user should not perform "Tap and drag right" gestures in order to scroll the block pane. This is achieved by letting the block pane roll back horizontally as soon as the user takes the finger up. This is the reason why user always starts the horizontal scrolling by doing a movement to the left, which allows the recognition of the movement as a scrolling gesture, see figure 21.

Execution

To introduce a visible, unique gesture for running an executable block - a symbolic command representation strategy is used. A special start token is placed on the tool bar. A programming block stack is executed by dragging this token onto the stack, see figure 16. The action space can be extended to any dimension by using land on tokens, see figures 18, 19.

3.5 Other design decisions

Interface related ideas and decisions.

Level of abstraction for commands

For the purpose of testing the ScratchTab interface by controlling a robot, the introduction of many blocks with self-contained

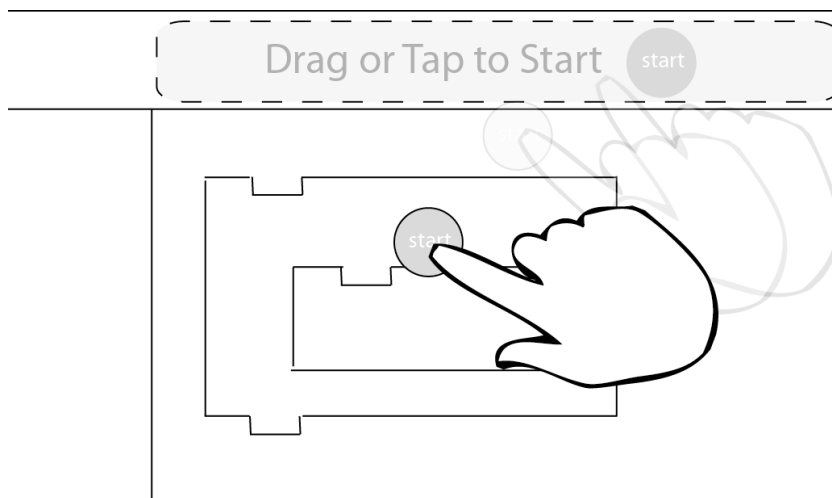


Figure 16: Blocks are executed by dragging tokens onto the blocks. The token appears, when the drag gesture is performed, starting on the start area inside the toolbar.

meaning on a higher level than motor control (e.g. not "turn on motor X" but "rotate the vehicle by 45 degrees") seems more suitable. This decision increases the number of blocks, but reduces the afford for composing basic actions again and again. Changing to a lower level of command abstraction should include the implementation of a stack copy command, to enable the reuse of existing commands, which are composed of lower level commands. Otherwise working with ScratchTab will result in boring recomposing of existing commands, if these commands are needed more than once.

3.6 GUI prototype

Graphical User Interface prototype do decide, whether the GUI can be implemented as described above.

After the foregoing user interface design decisions in this chapter an implementation of the first, non functional prototype was appropriate, in order to do the first evaluation of the decisions. To test the described GUI layout - the browser prototype technique has been audited as preferable, compared to the paper prototype technique.

A browser prototype is made of two parts:

1. The first part is a dynamic web page, run on a webserver of choice, simulating the prototype layout by using JavaScript.

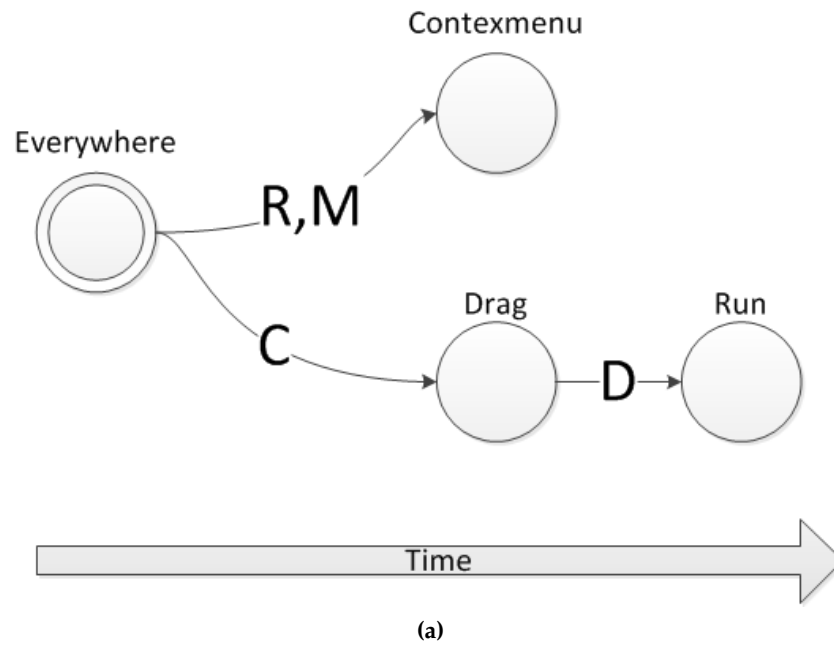


Figure 17: Scratches action diagram. **R** - rightclick. **M** - middleclick. **C** - click. **D** - doubleclick

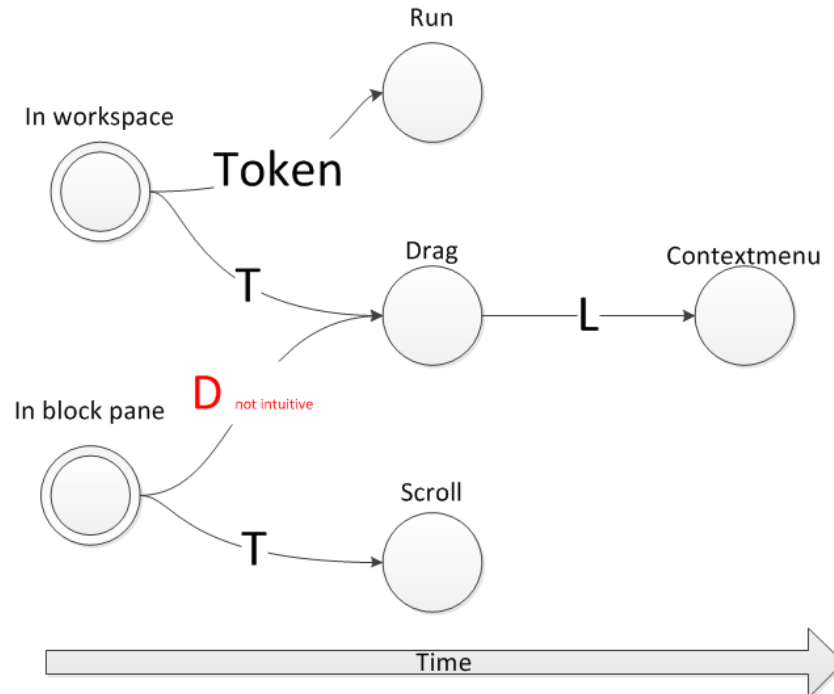


Figure 18: ScratchTab's action diagram. **T** - touch. **D** - double tap. **L** - long tap.

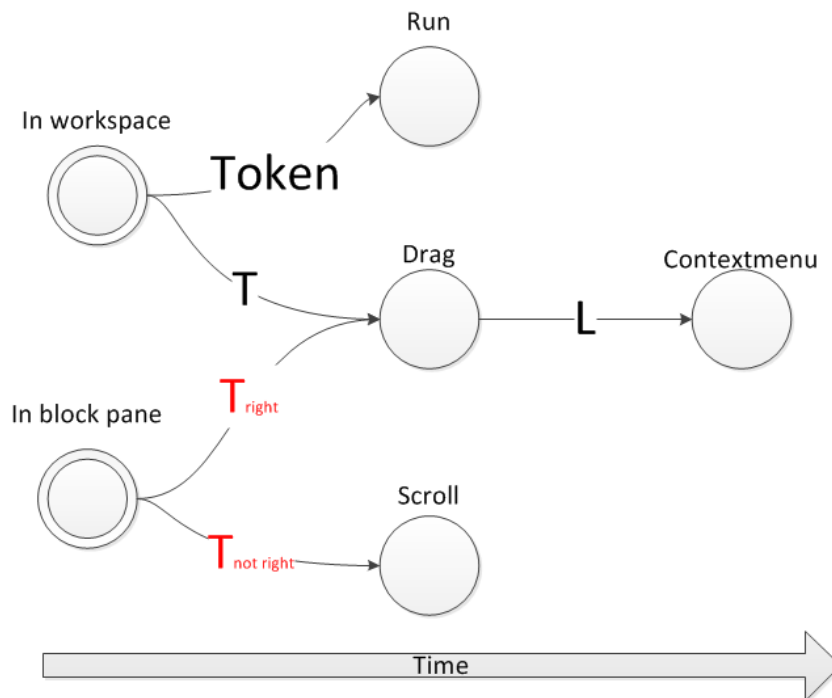


Figure 19: ScratchTab's action diagram after the redesign. T_{right} - Tap and drag right. $T_{not\ right}$ - Tap and drag into other directions. D - double tap. L - long tap.

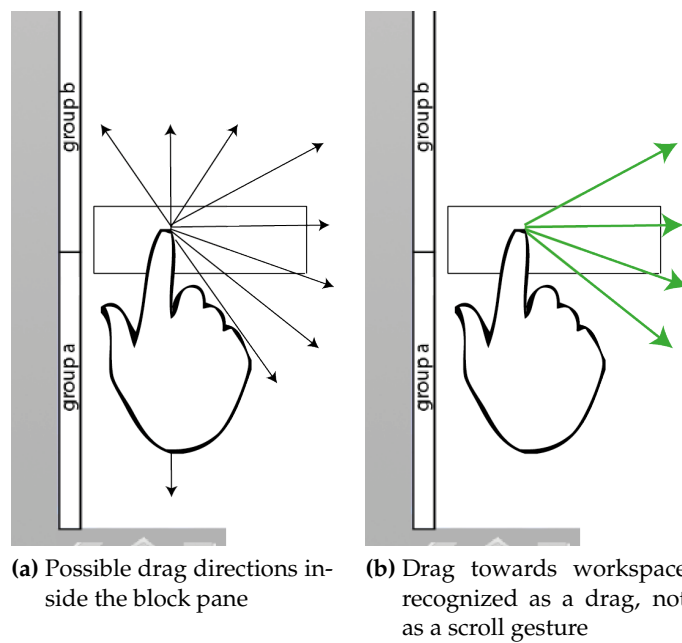


Figure 20: Dragging blocks from the block pane

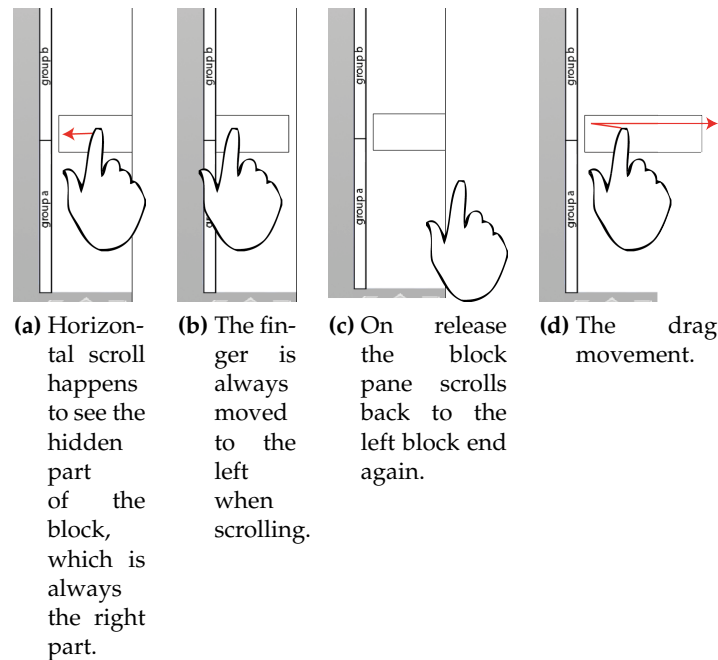


Figure 21: Scroll can not be recognized as a drag gesture. Every horizontal scroll gesture begins with a movement to the left. Scrolling back happens automatically, every time the user takes the finger up.

Since native JavaScript does not support some Tablet events like swipe, or sensor events - a JavaScript library named *jqTouch*², was used, which allows to react on orientation changes and the most touch events. Appendix 2, part 1.

2. The second part is a customized android-browser application, which opens the first-part-webpage and hides all the interface details which do not belong to the prototype. Additionally the browser application can support the development by visualizing the JavaScript errors, e.g. as a popup. For details see the source code from the Appendix 2, part 2.

It has been found, that a browser prototype brings a series of benefits compared to the paper prototype:

- easy adaptation for different devices, just by changing the website parameters.
- can easily be copied
- can be tested directly on a concrete device with the native look and feel

²<http://jqtouch.com/>

- can be created faster than a paper prototype, if the programmer is skilled in JavaScript, HTML, CSS

The layout described on the figure 9 was implemented, with some dummy blocks on the block pane. The whole layout was zoomable, the blocks could be dragged around by using a touch and drag gesture.

Four male students in the age of 24-29 were asked to use the prototype for dragging the block dummies from the block pane to the workspace. The following conclusions from their feedback.

- Zoom is very useful for fine operations.
- The block pane width of 5cm is sufficient.
- The text size of 10pt - 11pt is sufficient.
- Blocks should not be presented as a miniature on the block pane due to unreadable text.

3.7 Summary

[A short overview of the chapter.](#)

The current chapter defined the user interface for a block programming on touch-driven devices in detail. The implementation of the block programming language on project's platform - an Android driven tablet, is described in the next chapter.

4 Software architecture

The technical point of view to ScratchTab

The last chapter described the interface design of ScratchTab. This was done from the designer's point of view, without changing to the programming level.

This chapter describes ScratchTab, from the technical point of view, going into the implementation details.

The following notations apply within the current chapter:

Android classes are marked as: `classname`

ScratchTab classes are marked as: `classname`

4.1 ScratchTab as a framework

Introduction into the framework's basic terms

The whole ScratchTab application has become a framework with the aim of supporting the ScratchTab developer in designing new programming blocks which can then be combined by ScratchTab users to new scripts. To define a new block one must define

1. How the block should look like.
2. What command should be represented by the block.

The ScratchTab framework will then take over the positioning of blocks in the block pane, manage the interaction between the user and the system (e.g. by resizing the blocks on zoom, handle the touch and drag events etc.), regulate the execution of command stacks and distribute new threads.

Further the framework defines a group of allowed block types. At least the framework enforces the necessary feedback from blocks during the execution.

Shapes

The execution and the look and feel are maintained by two parallel structures in ScratchTab: the shapes and the blocks. The

shapes are responsible for drawing. The shape basic class, which all shapes inherit from is called `Shape.class`. The drawing is done on a 2D canvas of `Block.class`. When it is time to draw a block - it passes the available canvas to the shape and shape does all the magic.

Further the shapes are responsible for positioning slots because shape's internal data, like height, width, form is required to decide where and how many children should occur in the block belonging to the current shape. To sum up: shapes decide for blocks which children the block will have in it.

Blocks

The executional part is handled by the block structure. The execution of commands, handling of touch events, drag events are all managed by blocks. The basic class, which all shapes inherit from is called `Block.class`. There are two special blocks in ScratchTab: Slots and Labels.

Slots

Slot - a ScratchTab user should know, where he can drop a block into another block to connect them. Such connection spaces inside of a block are marked by slots. A slot is a container which marks the place, where blocks of a special block type can be dropped in. Slots are implemented as objects, which can contain a pointer to the block, which was dropped into the current slot - the infill block. The most events, input etc. the slot redirects to its infill. When the slot is asked for its size - it returns the size of its infill block. When the slot is asked to draw itself - it draws its infill object.

For empty Slots, which do not have any infill yet, to be drawn and measured correctly every slot has a dummy object. This dummy object is the visual representation of an empty slot - empty slots measure and draw their dummies instead of infill.

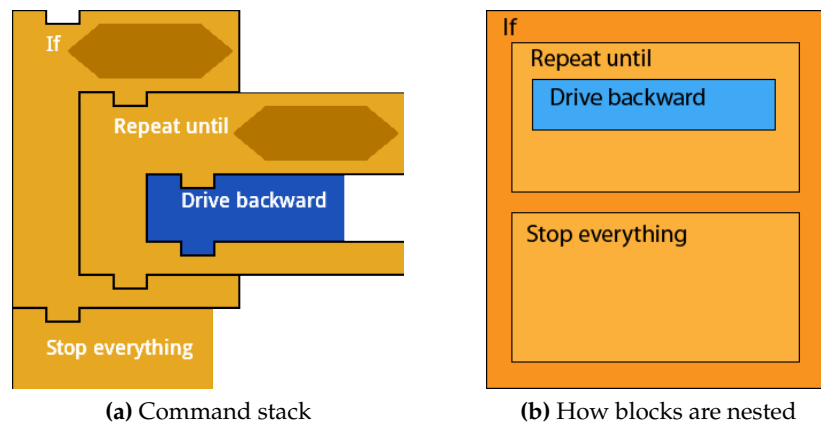


Figure 22: Nested blocks

Labels

Labels - are containers which can contain text and slots. Because many Blocks in ScratchTab have the same form and drawing behaviour - they can be reused, by adding different labels with different text and different commands to them. The basic forms, which were used in ScratchTab, are listed on page 39. Using labels increases the reusability of code. By using labels a ScratchTab developer can define block's shape only once and reuse by decorating it with different labels.

Command stacks, scripts

Scripts (alias command stacks) - constructs made out of programming blocks. In ScratchTab there is no special container which would represent a stack. Instead the blocks can contain each other recursively: if a block A was dropped into another block B - A becomes the child of B. The same happens, if one block is appended at the bottom of another block, see figure 22. This approach makes it much easier to move stacks around, because nested blocks are always drawn relative to its parent and they do not bother about repositioning, so only the top most block has to be repositioned.

4.2 How to use the framework

Using ScratchTab to develop new blocks

Creating an own block in ScratchTab

To define an own block, two java classes need to be created: a shape and a block class. Here is a step by step instruction for creating a new class:

1. First, block's shape should be created, since the shape is needed to define a block. For that choose the package which describes the meaning of the new block the best or create a new one. Inside of the package create a new class which should extend the class `ShapeWithSlots.class` and implement all the necessary abstract methods. It may be easier to extend one of the abstract subclasses of `ShapeWithSlots` which are illustrated by the following images:

`ShapeWithSlotsDiamond` on image 23,

`ShapeWithSlotsHead` on image 24,

`ShapeWithSlotsRectangle` on image 25,

`ShapeWithSlotsSingleLevel` on image 26,

`ShapeWithSlotsSingleLevelLast` on image 27,

`ShapeWithSlotsDoubleLevel` on image 28,

`ShapeWithSlotsTrippleLevel` on image 29.

These shapes describe the standard block-forms with the usual slots. If one of the listed shapes is suitable - extend it and the only thing you will have to do then is defining the right labels and putting them into predefined slots. For an example see any shape from the packages `com.binaryme.ScratchTab.Gui.Shapes.*`

2. Next step is the creation of the new block. Choose the right package for the new block and create a new class which should extend the class `ExecutableDraggableBlockWithSlots<S,E>`. The first generic parameter `S` should be the new shape, defined in the previous step. The other generic parameter `E` should be the type of the object, which the current block will return after execution. Implement all the necessary abstract methods, especially the method `initiateShapeHere()` should return an instance of the shape defined in the previous step. The method `executeForValue()` should be used to implement the block functionality.
3. To make the block appear on the block pane - add the complete path to the new block into the lists inside of the class `BlockPane.getAllAvailableBlocksAsGroups()`

The `ShapeWithSlotsDiamond` on pic. 23 should be used for blocks of the Boolean type. One Label is available for personalizing the shape.

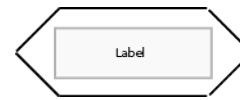


Figure 23: ShapeWithSlotsDiamond

The `ShapeWithSlotsHead` on pic. 24 should be used to implement first blocks in the stack. These blocks have only one command block at the bottom, the block itself cannot be dropped anywhere except of the workspace.



Figure 24: ShapeWithSlotsHead

The `ShapeWithSlotsRectangle` on pic. 25 should be used for blocks of the Data type. For an example see Shapes from the [Numbers](#) package.



Figure 25: ShapeWithSlotsRectangle

The `ShapeWithSlotsSingleLevel` on pic. 26 should be used for blocks of the Control type. There are two Slots in this shape, one for a Label, the second one at the bottom for the next block.

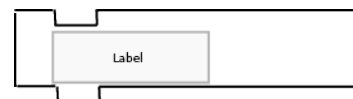


Figure 26: ShapeWithSlotsSingleLevel

The `ShapeWithSlotsSingleLevelLast` on pic. 27 should be used for control blocks like the `ShapeWithSlotsSingleLevel`. The difference is that there is no slot at the bottom.



Figure 27: ShapeWithSlotsSingleLevelLast

The `ShapeWithSlotsDoubleLevel` on pic. 28 should be used for blocks of the control blocks type. There are three Slots in this shape - one for the label at the top and two command slots at the top and at the bottom.

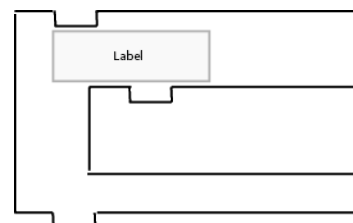


Figure 28: ShapeWithSlotsDoubleLevel

4.3 Source code










[A glance at the code structure](#)














The `ShapeWithSlotsTripleLevel` on pic. 29 should be used for blocks of the control blocks type. There are five Slots in this shape - one for the label at the top, one for the label in the middle and three command slots at the top, in the middle and at the bottom.






Figure 29: `ShapeWithSlotsTripleLevel`





In this section the cornerstones of the `ScratchTab` code are introduced which are necessary for better understanding and orientation. First the package structure of `ScratchTab` is explained here:

- ▷  **com.binaryme.AndroidSensors** manages tablet's sensors.
- ▷  **com.binaryme.Bluetooth** responsible for establishing a Bluetooth connection to a Bluetooth device. Fires events on Bluetooth connection, disconnection.
- ▷  **com.binaryme.DragDrop** contains classes, which organize the dragging, which know about the block being dragged currently, which start and stop the drag process.
- ▷  **com.binaryme.LayoutZoomable** the classes which are related to the zoomable workspace are here.
- ▷  **com.binaryme.ScratchTab** main package with the application controller and application controller related classes which catch unhandled exceptions, trigger the initialization of static classes in the application.
- ▷  **com.binaryme.ScratchTab.Config** contains classes with shared resources and application wide configurations.
- ▷  **com.binaryme.ScratchTab.Events** here the `ScratchTab`'s events (for triggering chosen head block's execution) are located.
- ▷  **com.binaryme.ScratchTab.Exec** classes to control block execution and to handle exceptions.
- ▷  **com.binaryme.ScratchTab.Gui** GUI related classes are located in this package

-  **com.binaryme.ScratchTab.Gui.Blocks** the basic classes of the block hierarchy are here, see block. Every Blocks.* package contain non abstract blocks which handle a concrete task.
-  **com.binaryme.ScratchTab.Gui.Blocks.Control** blocks with controlling structures.
-  **com.binaryme.ScratchTab.Gui.Blocks.Dummies** dummy blocks which are used by slots.
-  **com.binaryme.ScratchTab.Gui.Blocks.Logic** blocks of the type "boolean".
-  **com.binaryme.ScratchTab.Gui.Blocks.Numbers** blocks of the type "data" which return numbers on execution. (not strings)
-  **com.binaryme.ScratchTab.Gui.Blocks.Robot** NXT robot related command blocks.
-  **com.binaryme.ScratchTab.Gui.Blocks.Sensors** NXT robot related sensor blocks.
-  **com.binaryme.ScratchTab.Gui.Datainterfaces** Interfaces which allow handling different, data returning blocks, by the returned data type. E.g. the blocks which allow addition, subtraction, division, multiplication implement the [InterfaceNumDataContainer.class](#) and their output can be retrieved by getting the result [getNum\(\)](#)
-  **com.binaryme.ScratchTab.Gui.Shapes** the basic classes of the shape hierarchy are here, see shape. Every Shapes.* package contain non abstract shapes which can draw a concrete block.
-  **com.binaryme.ScratchTab.Gui.Shapes.Control** shapes for the control blocks
-  **com.binaryme.ScratchTab.Gui.Shapes.Dummies** shapes for the dummy blocks
-  **com.binaryme.ScratchTab.Gui.Shapes.Logic** shapes for the logic blocks
-  **com.binaryme.ScratchTab.Gui.Shapes.Numbers** shapes for the numbers blocks

- ▷  **com.binaryme.ScratchTab.Gui.Shapes.Robot** shapes for the robot blocks
- ▷  **com.binaryme.ScratchTab.Gui.Shapes.Sensors** shapes for the sensors blocks
- ▷  **com.binaryme.ScratchTab.Gui.Slots** this package contains all slots which are used in the ScratchTab, which are not only the slots for all existing block data types.

Additionally there are slots for labels, for android widgets like dropdowns(spinner) and slots for text fields which cannot receive dropped data but display some text (used to display sensor data). Labels, spinners, text fields cannot be used for interaction by drag and drop - they are added to the slot during the block initialization programmatically once and forever. It would be logical to add these data to a block directly without using a slot, but this would break the consistency in navigating through the block hierarchy.

- ▷  **com.binaryme.ScratchTab.Gui.Widgets** here subclasses of the android widgets, changed to be resizable, are saved.
- ▷  **com.binaryme.ScrollViewDual** classes which allow scrolling the workspace in two dimensions are here.
- ▷  **com.binaryme.tools** a toolkit responsible for converting measurement units from pixel to cm and back, handling colors etc.
- ▷  **icommand.navigation** classes from the framework capable to navigate an NXT as a vehicle, starting and stopping two wheel motors synchronously. See page 60 for further explanation.
- ▷  **icommand.nxt** classes from the icommand framework responsible for retrieving data from the NXT robot.
- ▷  **icommand.nxt.comm** system classes of the icommand framework
- ▷  **icommand.scratchtab** this package defines a connection port between ScratchTab and icommand framework.

Here the most important classes from the android API are listed and explained shortly. These classes are important because their functionality was used to implement the block structure.

*View*¹ - the class used to implement Graphical User Interface parts. It introduces a measure, layout, draw cycle. Drawing a tree of views requires two passes: one for measuring the views with respect to it's children, the second for positioning the Views recursively. Read the ViewGroup documentation of the android framework for better understanding.

read the Layout part in the View documentation of android for better understanding.

*ViewGroup*² - a special View which can contain other Views as children. Read the ViewGroup documentation of the android framework for better understanding.

*AbsoluteLayout*³ - a special ViewGroup which can position other Views inside of itself by using absolute coordinates. Read the AbsoluteLayout documentation of the android framework for better understanding.

Block.class - This class represents the basic building block for graphical user interface in ScratchTab. It is a direct successor of AbsoluteLayout, so it takes advantage of View's measure-layout-drawing cycle to draw the GUI.

ViewGroup's functionality allows nesting the Blocks to build complicated structures.

AbsoluteLayout allows positioning the blocks freely in each other.

The most Blocks have an own Shape, which the block will draw on its place, when it is asked to draw itself.

The Block has methods to position itself inside of an Absolute Layout, with respect to the current scale level.

Blocks can resize themselves when a scale event is happening, so the most blocks do not have to reimplement the scaling.

¹<http://developer.android.com/reference/android/view/View.html>

²<http://developer.android.com/reference/android/view/ViewGroup.html>

³<http://developer.android.com/reference/android/widget/AbsoluteLayout.html>

Measure, Layout, Draw cycle.

The distribution between blocks and shapes during the measurement and layout cycle is not obvious, so it is described in detail here. As mentioned above for dispatching of the measurement, layout and drawing events the android framework's classes are used.

Before a block hierarchy can be drawn it has to be measured and laid out. Measuring blocks is critical because interdependencies between children and parents make it complicated. The usual dependencies are:

A height of a block, which should graphically wrap another block should be measured. It cannot be measured until it's child's height is measured, see picture 30a

A width of a block, which should graphically wrap another block should be measured. It cannot be measured until it's child's width is measured (picture 30a)

A slot should be as wide as the label of the wrapping block (picture 30b). So the block's width sometimes depends on it's children's width, and children's width sometimes depends on it's parent's width - a cycle occurs.

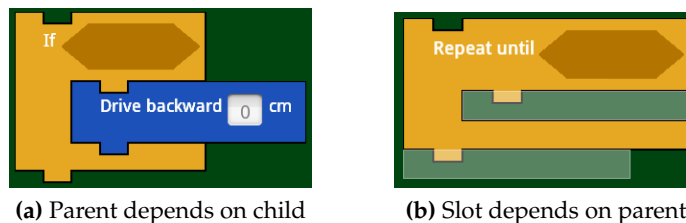


Figure 30: Drawing dependencies

To avoid cycles the drawing of a stack should happen as following:

1. The labels of the top block are measured before going deeper, so that the slots can use the measured values.
2. The children are measured as next and can use the label width.

3. When all children of a block are measured the root block's width can be determined as the maximum of the width of all labels and the width of all children.

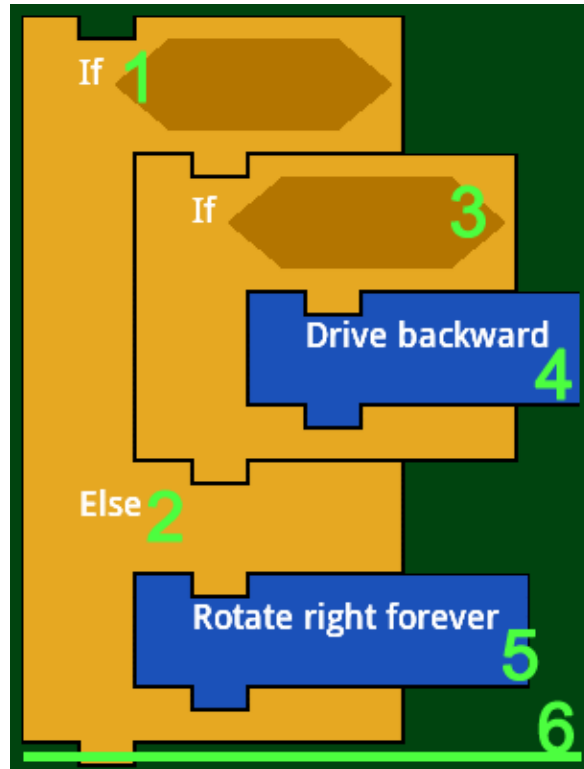


Figure 31: Measuring order of an example stack

On the picture 31 you see the measuring order of an example stack. The items number 1 and 2 are top level labels. Then measuring cycle is going deeper and measures the first child. The child measuring is started by measuring its label - item number 3. Going deeper, measuring the child number 4. Now return to the top level and measure the other child fork - number 5. At least measure the whole block as the maximum of all children and labels - number 6.

UML

In this part the basic decisions in code design are explained and the UML diagrams of the code ScratchTab are presented. On the picture 32 the most complete UML diagram is presented. The intention behind this picture is to give a general overview over the class structure. For more detailed information consider reading

the documentation, see Appendix 3 on page 84. The block hierarchy on the picture 33 and the shape hierarchy on the picture 34 are highlighted in the second and the third separate diagram for easier orientation.

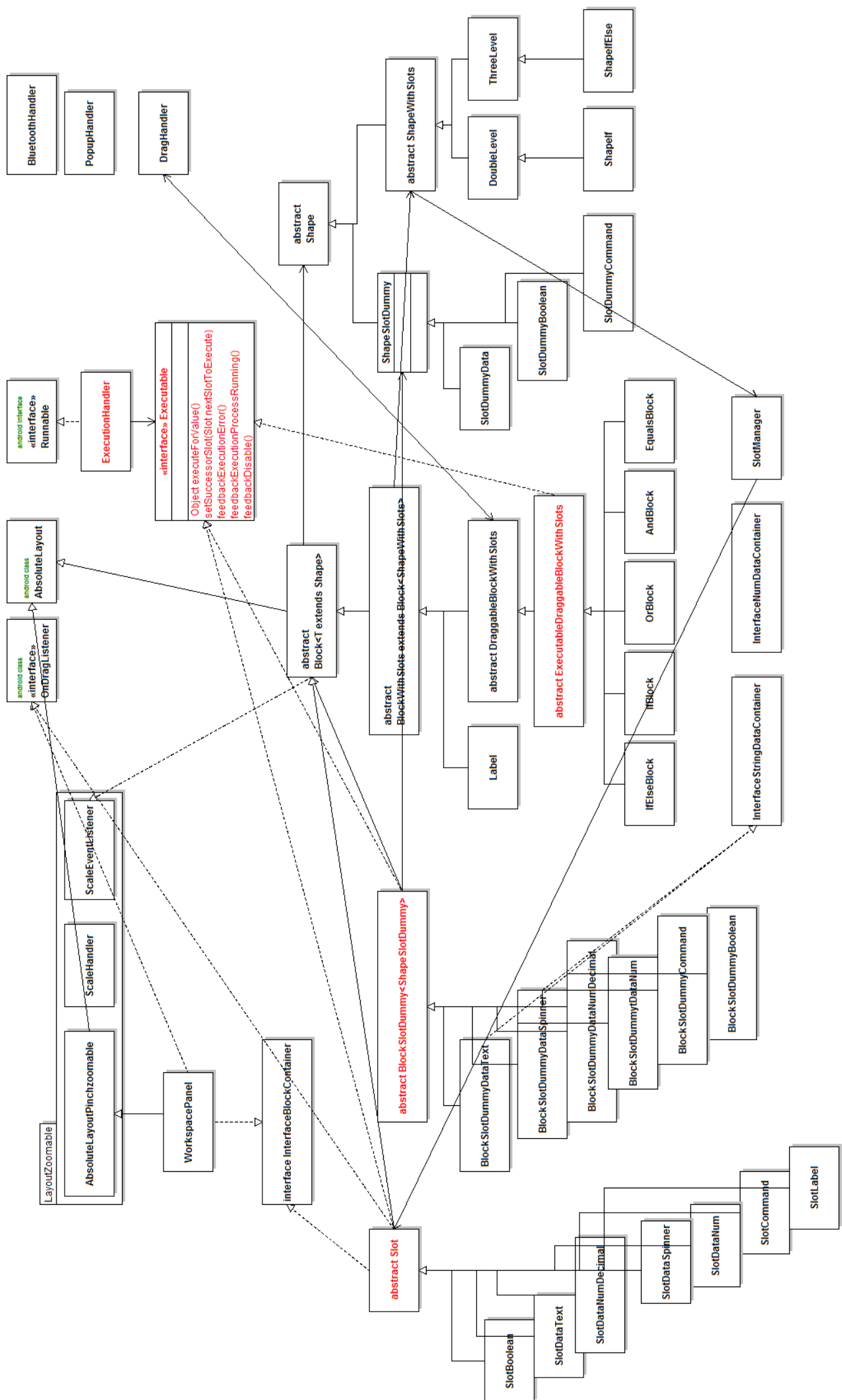


Figure 32: The core structure.

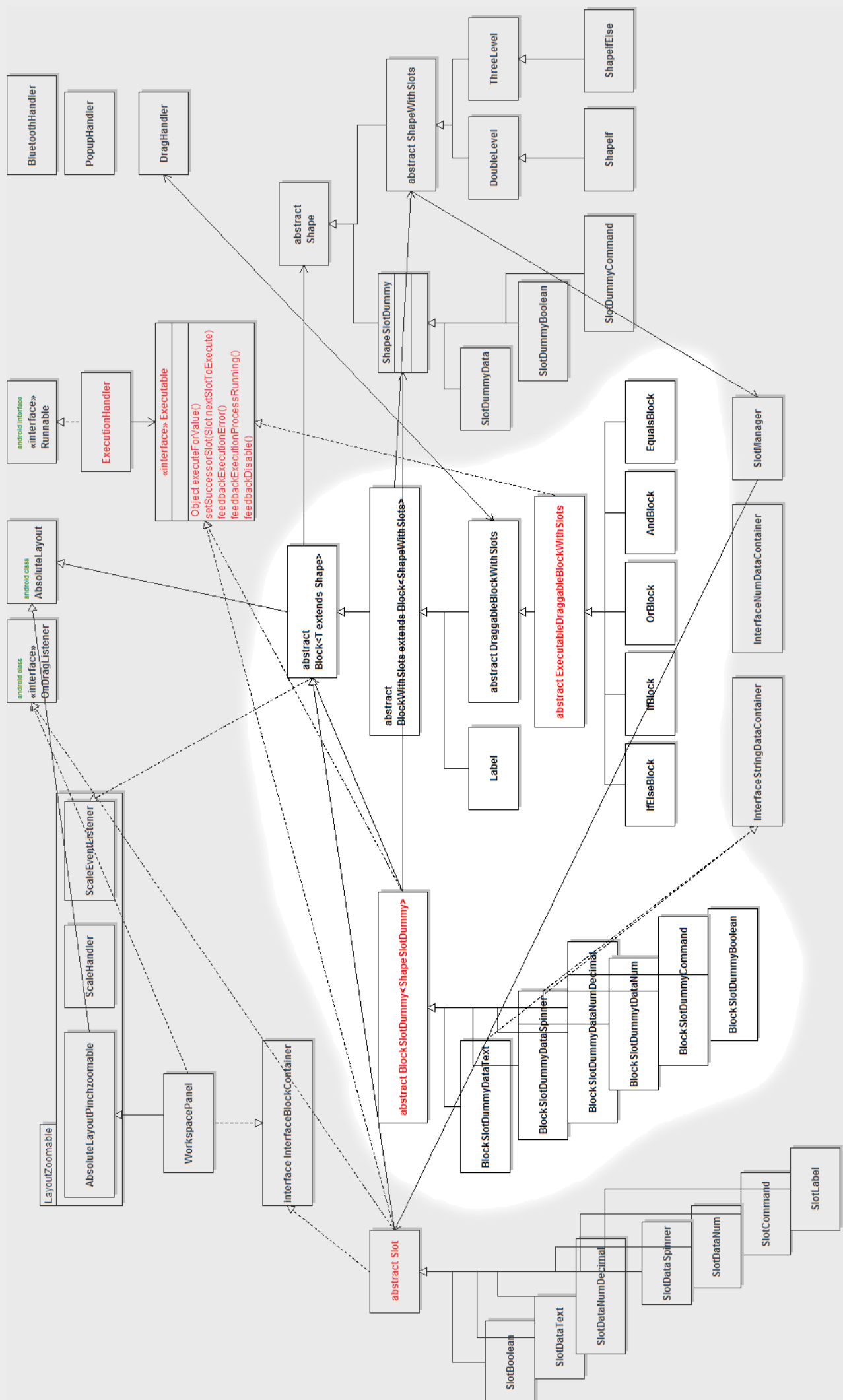


Figure 33: The block structure.

Howto navigate between blocks and shapes

For the purpose of clarity the methods in the UML diagram were hidden. However I am aware about the problem of finding the right methods to navigate inside the application structures. Navigating through a block stack has become non-transparent during the development. For that reason the methods used to navigate through a stack are handled here separately and in detail.

The methods are explained by using a simple stack as an example, see figure 35. As a hierarchical structure, where all invisible containers are depicted, it looks more complicated. See figure 36.

- The text "If" and the diamond-formed boolean-slot are not located in the block directly - they are wrapped by a label which aligns them horizontally.
- The label is not located directly in a block. It is wrapped by a label Slot.
- The command Slot filled by the "drive forward block" has both - a dummy and an infill.
- Empty diamond-formed boolean-slot has no infill - only a dummy.

On the figure 35 the methods, which can be used to navigate inside of the concrete example structure are depicted as arrows. All methods are relative to the class, which the arrow with the method starts from. The colors have no special meaning except of clarifying the arrow direction. The squares represent instances of classes which names are seen inside the squares. If a square A is connected with another square B below by a grey line - then the matching Block A is a child of B.

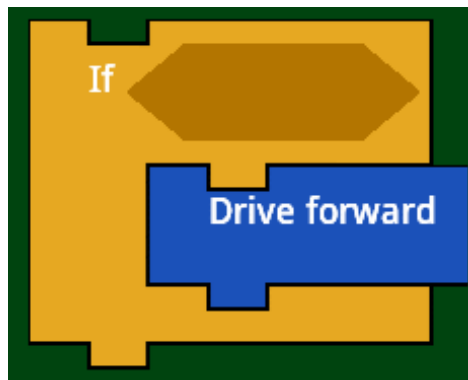


Figure 35: Example stack for understanding navigation methods.

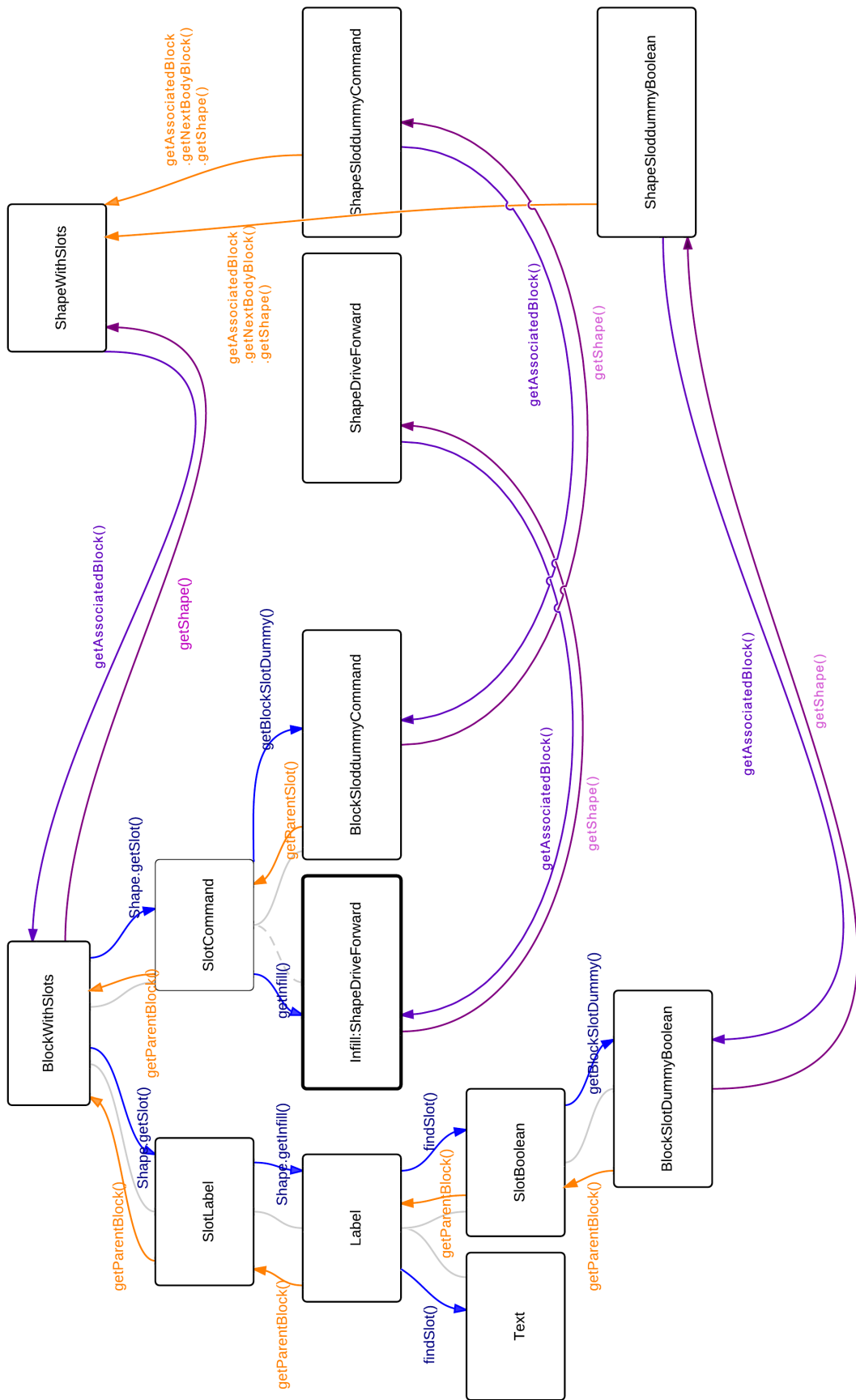


Figure 36: Methods used to navigate through a Stack

Execution of embedded commands

Maybe the most important framework part is the execution part. Here the implementation of the semantic block part is explained.

The executable blocks inherit from the `ExecutableDraggableBlockWithSlots.class` which implements the Interface `Executable`.

The semantic part of the block - the block command, is implemented in the interface's method `Executable.executeForValue()`. This method is implemented by the one, who creates a new programming block - the ScratchTab developer.

The implemented `executeForValue()` method is one of the most critical parts during the ScratchTab runtime, because for different blocks it is written by different people and may be not well tested. That means that someone has to handle the exceptions, which occur during the execution of blocks. Further every stack should run on its own thread, so that the execution of an own thread won't block the whole application. This and all other necessary execution management, like triggering feedback on blocks is done by an instance of the class `com.binaryme.ScratchTab.Exec.ExecutionHandler.class`. Every time when a new stack is executed - a new instance of an `ExecutionHandler` is created. This instance is passed to the `executeForValue()` method, during the recursive execution blocks.

The execution itself is a recursive process which is started on the root block of a stack. The following happens during the execution:

- The new `ExecutionHandler` instance calls the method `executeForValue()` of the root block.
- If needed, the `executeForValue()` method of the root block will execute root's children (block which are currently in roots slots). Read more about this below.
- When the `executeForValue()` of the current block is ready the `ExecutionHandler` will trigger the execution of the next block. The next block is located in the successor Slot, a pointer to which the ScratchTab programmer defines in the hook `ExecutionHandler.getSuccessorSlot()` during the implementation of a block.

During the implementation of the `executeForValue` one may need to know the values of nested blocks. These values can be retrieved by finding the relevant Slot with the needed successor and executing it by running `Slot.executeForValue()`. Since this happens during the implementation of the `executeForValue()` method - the first recursive fork is triggered here by going deeper into the block. It is important, that the `executeForValue()` is called on a Slot object, because the Slot reuses the `ExecutionHandler` to execute it's infill.

Every possible infill implements the `Executable interface` – this allows the Slot to ask it's infill for the result by executing it. Different infill types are executed differently:

- If the currently executed Slot is empty – the dummy is executed and returns the default value for it's type, or the value of the input field if the dummy has an input field.
- If the currently executed Slot is filled by another block – the infill block is executed. Executing the infill block may imply the execution of another blocks etc.

Example:

consider an executable block, see picture 37, which implements the addition. It has two executable children. The results produced by the children should be added. The children can be just numbers or another

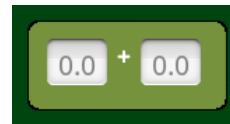


Figure 37: Executable block for adding two numbers

blocks. Inside of it's `executeForValue()` the addition-block finds the slot where the first child should be and executes it, retrieving a value. If the Slot was empty - it retrieves the value of the dummy. The same happens with the second child.

Previous to the first execution an `ExecutionHandler` instance is created by the object which has caused the execution. This object is passed from `executeForValue()` to `executeForValue()` between executables. Slots are responsible for reuse the `ExecutionHandler` instance recursively, to launch the next object on the `ExecutionHandler` thread. In the case of the addition-executable it will be the `SlotDataNum`, because this slot is capable for receiving numbers as dropped blocks or as numbers. See the source code of `SlotDataNum` for better understanding about the reuse of the `ExecutionHandler` instance.

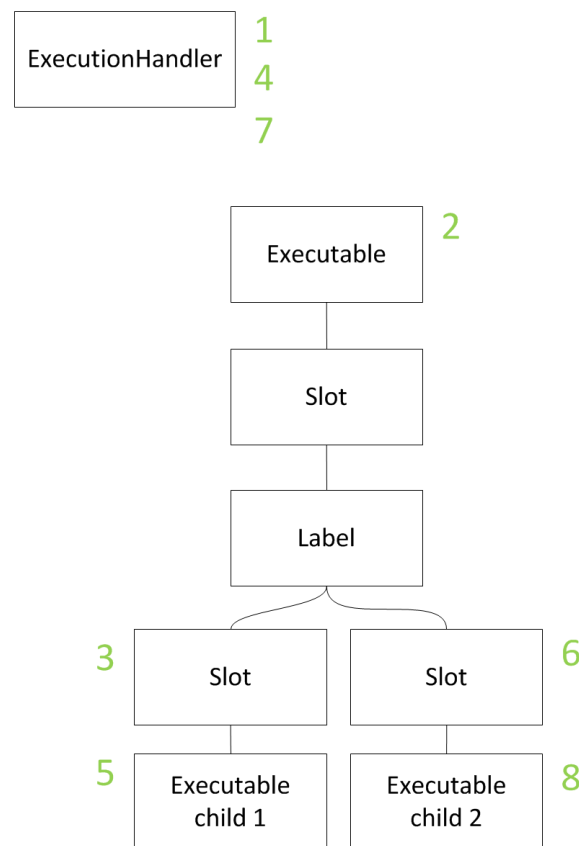


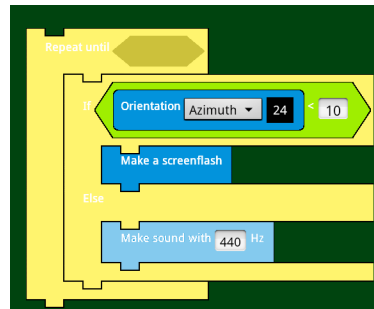
Figure 38: Illustration of execution process

On the picture 38 you can see the additon-executable and its block hierarchy. The green numbers indicate the object order, in which the objects occupy the ExecutionHandler thread. The label-Slot does not need to execute anything, instead it passes the thread on it's children slots, which are current inside of the label. The slots do not execute anything either, however they do not pass the thread to their children directly. Instead they ask the ExecutionHandler to execute the children. If an error occurs now the ExecutionHandler will catch it and fall back to the previous cycle.

Artificial slowdown of execution

In order to support user's understanding about the current happening inside of his program, the executed blocks are highlighted for a short period of time during the execution of a program(as seen on figure 39). Since many commands are executed imperceptibly fast, all commands are executed with with a minimum execution time. The slowing down is done by the

`ExecutionHandler.class`, since it handles as a wrapper for all executed commands.



(a) Screen size

Figure 39: Highlighting of blocks during the execution. The Blocks "Repeat until", "If" and "Make Sound" are currently highlighted.

4.4 Drawing

Things related to the displaying of the blocks

Device
independent
measurement
units

Because ScratchTab interface contains fine details which are manipulated by touch, the usability of ScratchTab strongly depends on the size of the GUI details, for that the GUI details size should not differ on different devices, especially it should not depend on screen density (Figure 42).

For the purpose of unambiguousness the terms resolution, density and screen size are introduced here:

Def 6: Resolution

Resolution

Resolution is measured in pixel per inch and indicates the total number of physical pixels on the screen, see figure 40a.

Def 7: Density

Density

Density is the number of pixels on a specified area, here one inch 40b.

Def 8: Screen size

Screen size

Screen size is the physical size of the screen. Resolution grows on ascends together with the screen size. Density is screen size independent 40c.

To enable density independent measurement Android introduces density-independent pixels(DP). Density independent pixel number on a concrete screen is calculated as following: device screen density is first mapped to one of the four general density cases (120dpi 160dpi 240dpi 320dpi) depending on it's physical density. **Depending on the mapping** the available pixels per inch(dpi) are then converted to the density-independent pixels by dividing them through 160. So if the screen is mapped to 120 dpi, one density-independent pixel equals $120/160 = 0.75$ pixels, $160/160 = 1$ pixel when the screen is mapped to 160 dpi and so on, see picture 41

Now, because Android does not work directly with the physical screen-density, but maps the physical screen density to the four general cases. For two screens within one general case but with different physical density (e.g 149dpi and 169dpi will be mapped to the same case of 160dpi) one density-independent pixel would not have equal size on both screens, so the T on the Figure 42 would be displayed differently, if it would be defined in Android's density-independent units.

Because of the described situation ScratchTab uses own density independent units "density-independent centimetres", which are simply defined as physical density-per-inch divided by 2.54 since inch equals 2.54 centimetres.

The zoom in ScratchTab is managed by the class `ScaleHandler.class`. Since the `Block.class`, which is the root class for all Blocks, can already adopt it's size on scale events triggered by the ScaleHandler - all Blocks in ScratchTab are zoomable. The zoom changes within the $[0.5;3]$ Intervall. The default zoom level is 1. Every zoomable object in ScratchTab encapsulates 4 different size variables: UnscaledWidth, UnscaledHeight, Scaled-Width, ScaledHeight. The UnscaledWidth and UnscaledHeight variables contain the default width and height with, which apply when the zoom level is 1.

Zoomable blocks

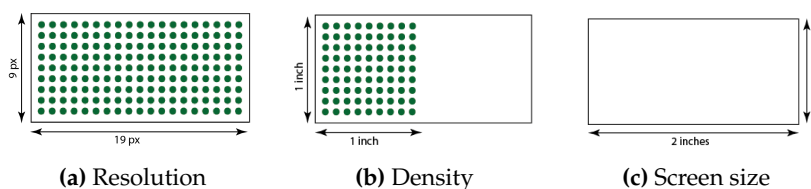


Figure 40: Resolution, Density and Screen size

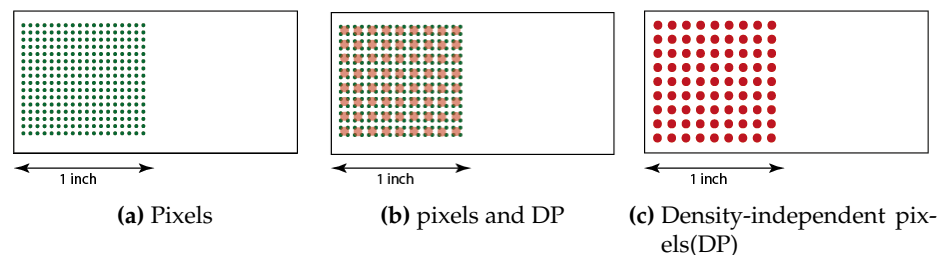


Figure 41: Screen independent units

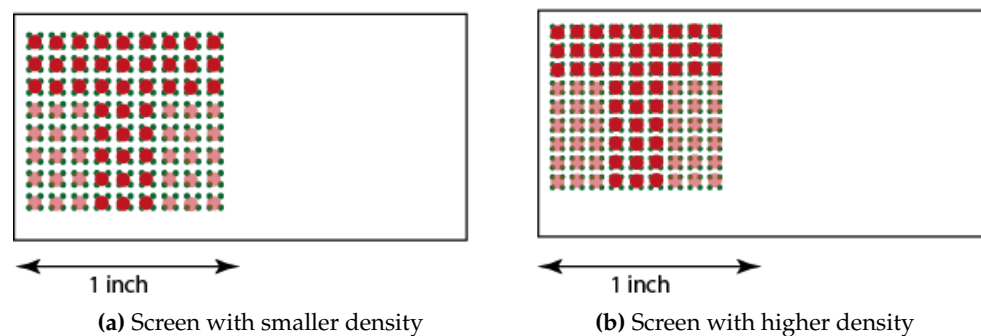


Figure 42: Different sized T, displayed on screens with different density within one general density case

The `ScaledWidth` and `ScaledHeight` contain the actual object size, with respect to the zoom level. They are computed by using the `UnscaledWidth` and `UnscaledHeight` on every zoom event. This approach was preferred to the approach of one width/height pair for every object, because keeping a pair of standard values and calculating the size by using the standard values reduces the rounding errors, which otherwise would accumulate. To sum up, after a scale event it should always be true:

$$ScaledWidth = ScaleHandler.getScale() * UnscaledWidth$$

$$ScaledHeight = ScaleHandler.getScale() * UnscaledHeight$$

Workspace

For implementation of a scrollable workspace, Android provides vertical scrolling widgets³, horizontal scrolling widgets⁴ but there is no scroll widget which can scroll in both directions. It is possible to build such a widget from the `ScrollView`¹ and `HorizontalScrollView` widgets mentioned above, but there are some things about event passing to know.

Motion events in Android are composed of event actions. Every motion event has a starting action(`ACTION_DOWN`²) and a closing action action(`ACTION_UP`²). In between different other actions can follow. Within of the scrolling widgets the events are distributed in a two pass process.

First events are passed top-down through the view hierarchy by using the `onIntercept`³ method. At this stage every view can intercept the current event, so that no view below in the hierarchy will get any action of the current event.

In the second pass, events are passed bottom up by using the `dispatchTouchEvent`⁴

On default the `ScrollView` widget intercepts every event with any component. Because people never move their finger straight during a touch gesture nearly every event is intercepted and not passed to the views below. To change this behaviour the `onInterception` method should be overridden in both scrolling widgets. After that it becomes possible to combine the scroll widgets into one widget with a capability of vertical and horizontal scrolling by nesting them. Now the event passing in `ScratchTab` looks like in figure 43.

³<http://developer.android.com/reference/android/widget/ScrollView.html>

⁴<http://developer.android.com/reference/android/widget/HorizontalScrollView.html>

¹<http://developer.android.com/reference/android/widget/ScrollView.html>

²<http://developer.android.com/reference/android/view/MotionEvent.html>

³<http://developer.android.com/reference/android/widget/HorizontalScrollView.html#onInterceptTouchEvent%28android.view.MotionEvent%29>

⁴<http://developer.android.com/reference/android/view/View.html#dispatchTouchEvent%28android.view.MotionEvent%29>

Single touch events, which hit a block on the workspace are handled as related to this block e.g. as drag events. Multi touch events are handled as pinch-zoom events by the workspace. Single touch movement events, which do not hit the block are handled by the scroll widgets.

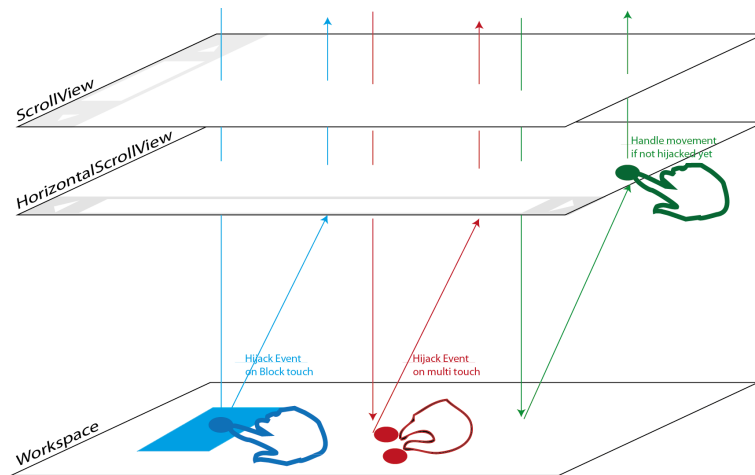


Figure 43: Motion events are distributed in a two pass process. They are passed through vertical and horizontal scrollview widgets to the workspace and handled on different interface levels.

Since the blocks on the workspace are positioned by dropping, it is appropriate to use a view with absolute coordinates as the workspace, so that the blocks can be drawn at the given drop coordinates. Since there is no such a view with zoom support, it is implemented in this project by inheriting from [AbsoluteLayout](http://developer.android.com/reference/android/widget/AbsoluteLayout)⁵

Zooming is implemented by hijacking multi touch events on the workspace level and introducing a scale event. Every zoomable GUI detail implements the [ScaleEventListener](#) interface and changes its size and position according to the new global scale level. Zooming is implemented by hijacking multi touch events on the workspace level and introducing a scale event. Every zoomable GUI detail implements the [ScaleEventListener](#) interface and changes its size and position according to the new global scale level.

4.5 Lego NXT robot

How the NXT robot is connected to ScratchTab.

The ability of controlling the Lego NXT robot was added to ScratchTab in order of giving a playful and vivid task to the

⁵<http://developer.android.com/reference/android/widget/AbsoluteLayout.html>

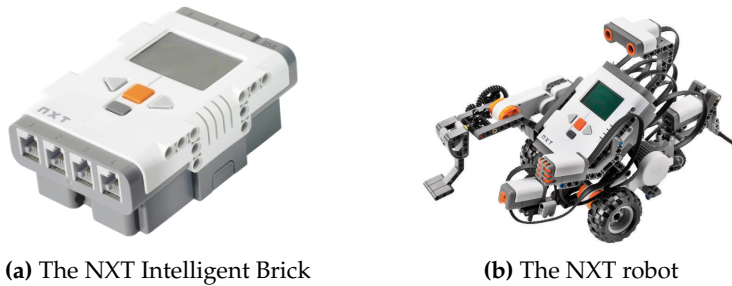


Figure 44: The NXT robotic toolkit

ScratchTab programmers. In ScratchTab there are several Blocks with embedded Lego NXT commands.

Lego NXT is a robotic kit. It's components are:

- The NXT Intelligent Brick 44a, which is a brick-shaped computer, which is capable to control up-to 4 connected motors.
- Step motors
- Sound, touch, light and ultrasonic sensors. Within the framework of this project the upper components were used in the combination which is depicted on figure 44b.

Protocol

In order to keep the mobility of the project platform, wireless communication was considered as suitable for the data exchange between ScratchTab and the NXT Intelligent Brick. Bluetooth protocol was chosen as a concrete protocol. The commands, which are transferred via Bluetooth, are formulated by using the "direct commands". These commands are specified in [Mindstorms, 2006].

As stated in [Mindstorms, 2006], direct commands were created in order to provide a simple way of controlling the NXT bricks. The control over the NXT brick includes: turning motors on and off, for a limited or for an unlimited time, stopping the motors immediately or just turning off the power allowing the motor to move by inertia, receiving the sensor input etc. Each direct command is a set of bytes, concatenated to a package according to the following pattern: Byte's meaning inside the package is determined by byte's position. First two bytes specify the length of

the package. They are not included into the byte enumeration by the NXT specification.

Length Byte 1 and 2 Not included into the enumeration by the NXT specification.

Byte 0 specifies the type of the command. Possible command types are:

0x00 Direct command telegram, response required

0x01 System command telegram, response required

0x02 Reply telegram

0x80 Direct command telegram, no response

0x81 System command telegram, no response

Byte 1-N the commands body, which is specified in Lego's Bluetooth development kit, see [Mindstorms, 2006].

The replies to the commands, which the NXT Intelligent block is capable to send back via Bluetooth are similar bytes strings. As stated above, the bit 0x02 which follows after the two length bytes, makes command-replies identifiable as such. The figure 45 outlines the order and determination of bytes inside of a commands and replies.

Length Byte 1	Length Byte 2	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	...	Byte N
------------------	------------------	--------	--------	--------	--------	--------	-----	--------

Figure 45: The direct commands consist of bytes, where byte's meaning is determined by byte's position.

ICommand

In ScratchTab the byte-package-commands were not composed manually. A framework named "iCommand", developed by the leJOS team⁶ was used for these purposes. "iCommand" is a Java framework, which encapsulates the direct commands protocol and provides a higher level commands API. In 2009 the development of this framework was stopped, however the it is still available sourceforge⁷.

⁶<http://lejos.sourceforge.net/>

⁷<http://sourceforge.net/projects/nxtcommand/files/>

Comments

During the project two main issues of the NXT-control via direct commands were noticed: "The motors perform inaccurate, when they are controlled via the direct commands. They always turn too far, compared to the given numbers of rotations. "The NXT Intelligent block provides an automatic mechanism for correcting the previous movement, see [*Mindstorms NXT Toolbox*]. For example if a motor turned 2 rotations too far after the last command, it will reduce the next movement by 2 rotations. This leads to problems, when the robot is manually repositioned, because the robot behaves unexpectedly by doing the movement with respect to the previous action. However for the purposes of the current project the precision provided by direct commands was sufficient.

4.6 Summary

[A short overview of the chapter.](#)

This chapter described the system architecture of ScratchTab. The important points were about extracting of the drawing-related classes into a separate structure, about controlling the execution of embedded commands and about controlling the NXT Robot.

The following chapters deal with the evaluation of the prototype, which resulted from the current chapter.

5 Interaction

How the system is used

After the description of ScratchTab's design and implementation in the foregoing two chapters first user experience were made with the project's working prototype.

This chapter contains a short description of an observation, about the way in which the prototype was used by the first testers. Further the consequences, which this observation had to the system are described in this chapter as well.

During the development of ScratchTab a group of people was repeatedly asked to use the system. After some time those people (4 male students) knew the system quite well and their behaviour in system usage was different from the behaviour of people who was using ScratchTab, it was more structured and could be roughly separated in two stages:

1. In the first stage the blocks were pulled down from the block pane and put together to a script.
2. In the second stage the input of data was performed: numbers, text etc.

The whole process was iterative, sometimes the first phase included more sometimes less blocks.

The screen requirements of the both stages are not the same. The second stage needs the virtual keyboard to be fold out, for the user to type in the information. The first stage needs the workspace to be as big as possible to enable comfortable manipulation with blocks, so the keyboard should be hidden.

ScratchTab supports the two phases, by showing the keyboard as soon as the user touches an input field, the right type of the keyboard for the right type of the input field and by hiding the keyboard as soon as the user touches a block on the panel or on the workspace, because it is assumed that the user passes to the next stage at this moment.

6 Evaluation

Prototype evaluation and results

In the previous chapters the system design and implementation were described. All the forgoing descriptions were implemented in a working prototype.

This chapter is about the evaluation of the prototype, created during the forgoing project. There were two stages of evaluation: an expert review, and user tests.

6.1 Expert review

The first stage of the evaluation process

Previous to the user tests the prototype was presented to people with rich experience in the field of interface development. Some Improvements could be done after this review.

Problem It was found that consistency of tap-gesture should be favoured over the safety of a unique double-tap gesture, when initializing blocks from the pane.

Solution Double tap gesture was replaced by a "drag to the right" gesture as described in the interface-design chapter on page 23.

Conclusion During the user tests the new gesture was used without any problems.

6.2 User tests

The second stage of the evaluation process

At the end of the project some user tests have been conducted to evaluate the user Interface and to estimate how the whole idea, in its current implementation, is perceived by the user. For that 7 students between the ages of 22 and 29 with different level of prior relevant experience as stated below, were asked to use ScratchTab and to fill in a form after that. The form contained questions about the experience with the touch interface and about

Nr	Gender	Age	Field of study	Exp.	2	4	3
1	M	24	management	6	✓	✓	
2	M	22	informatics	9	✓	✓	✓
3	F	25	pedagogy	5	✓		✓
4	M	27	engineering	9	✓	✓	✓
5	M	25	food technology	6	✓	✓	
6	M	29	management	0			
7	F	26	biotechnology	5	✓		✓

Table 2: The user tests participants. The Exp. column reflects previous user experience with touch interfaces and programming. 2 Points were given for any previous usage of touch devices. 4 points for regular usage of touch devices. 3 points for programming experience.

programming experience. To evaluate people's feelings about the system the JoyOfUse, as described in [Hatscher, 2001] was computed. For this purpose the AttrakDiff form (see [Hassenzahl, Burmester, and Koller, 2003]) was used, with the word pairs as seen on the figure 48.

The users

The seven test participants were students of various disciplines. To determine their previous knowledge several questions about their previous technical experience with touch interfaces and programming were asked (The complete form with questions is available in the Appendix 4, see page 84). From their answers it was derived, if they have ever used a touch device, if they regularly use one and if they have previous programming experience. In order to introduce a scale of technical experience for the current project, previous experience of participants was weighted according to the subjective importance for the current project:

- Previous usage of a touch device: 2 points
- Regular usage of a touch device: 4 points
- Programming experience: 3 points

In the table 2 the participants with the matching field of study and the relevant technical previous knowledge are listed.

The test

The user test started with a short introduction into the system. The interface was explained (the block pane, the workspace, the toolbar) and the blocks were explained as a visual representation of commands. Each of those commands was introduced. Two random blocks were instantiated and combined on the workspace to explain how the combination of commands works. After that the explaining part was over and the users were asked for questions.

The second part of the user tests had the goal of confronting the users with the system, so that they can judge it. For that a set of missions with a rising complicity were given to the users. Those missions were all about navigating the robot around a box in the middle of a room see figure 46. Different users accomplished the given missions with different speed and success. As soon as people have been spending half an hour on their missions no new missions were given. All missions were filmed by a camera, which was placed on user's chest, in order of analysing user's activity after the test.

The missions for the user tests

1. The first mission was about navigating the robot along the box and stop after the robot has passed the box. People were told, that they can use any approach they want for this mission. The goal of this mission was to let people make first experiences with the system. The first mission is seen on figure 46a

Every participant has completed this mission. The participants 3 and 6 with less previous experience started the robot to drive forever and stopped it with the "stop" button on the toolbar. The other participants started the robot to a limited distance after which it stopped automatically.

2. The second mission was about navigating the robot around the box to it's opposite side. The second missions is seen on figure 46b.

Every participant has completed this mission as well. Participants 3 and 6 exceeded the half an hour during this mission.

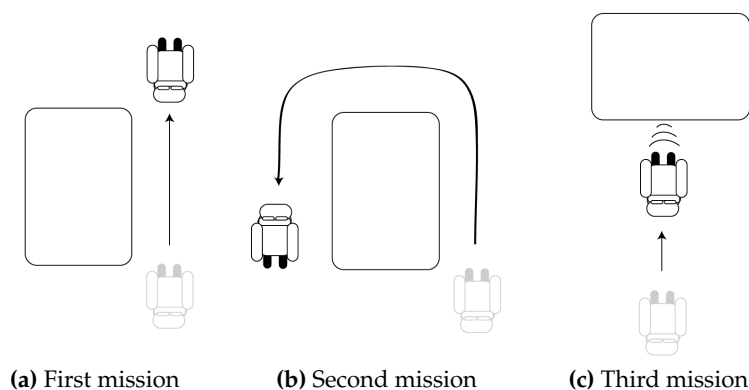


Figure 46: The missions, proposed to the users during the user tests.

3. The third mission was about letting the robot drive against an obstacle and letting it stop automatically by using the distance sensor, as seen on the figure 46c.

Nearly everyone, except of the participant 2 who studies Informatics, requested some help during this mission. Everyone exceeded the deadline of the half of an hour during this mission.

After the last mission people, who completed this mission successfully, seemed to feel motivated and played around with the system on their own. They were not distracted until they were ready. After that they were asked to fill in a form which is available under Appendix 4 (see page 84), to evaluate their technical affinity and their feelings about the system usage.

Finally, a short retrospective testing interview was conducted to uncover the intentions behind some of user's doings. The user's intentions mentioned below were clarified in theses interviews.

About the AttrakDiff form

In order to compute users's JoyOfUse three factors were computed using the AttrakDiff questionnaire. The first factor is the system's "pragmatic quality" (PQ), which indicates the usefulness of the system for the user. This factor is very close to the definition of usability by DIN EN ISO 9241-11. The second factor is the system's "hedonic quality", which addresses the user's

need for change and novelty and should indicate whether the system is desirable and exciting. The "hedonic quality" is additionally separated in "hedonic quality – stimulation" (HD-S) which is responsible for measuring whether the system stimulates the user to acquire new skills and "hedonic quality – identity" (HD-I) which should evaluate whether the user can identify himself with the system. The third "attractiveness" factor (AT) is the general attractiveness of the system to the user. All word pairs from the AttrakDiff form are seen on the figure 48. Since not every user could speak English, the test was translated to German. The user had the possibility to choose from between the two extremes which were marked with the words from the matching pair. The extremes were rated as -3 and +3. The whole form as it was presented to the users is available in the Appendix 4, see page 84.

6.3 User test results

The conclusions which were made during the user tests

The results of the user tests consist of the evaluation of the AttrakDiff form, responses which were of interest and the identified problems with the proposed solutions.

AttrakDiff evaluation

The questionnaire was evaluated by calculating the average of all ratings for every word pair. The calculations are present as Appendix 4, the average values and their standard deviation are presented in the table 3. The average values and the standard divergences in the form of a graph are presented on figure 48.

The pooled average values give the following factors:

- Pragmatic-quality of 0.7
- Hedonistic-quality of 1.8
- Attractiveness of 1.7

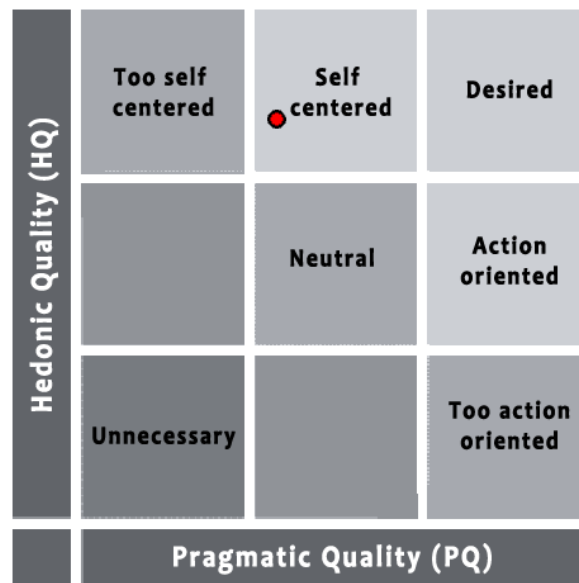


Figure 47: Visualization of the AttrakDiff form evaluation. This is an english translation of the visualization, available in the report on <http://www.attrakdiff.de/>

When visualizing the evaluation as done by the authors of AttrakDiff in their online tool ⁸ ScratchTab was guided as too self-centred which means that it has to improve it's usability, see figure 47. This result probably was influenced by the bugs, which were revealed during the user test of the prototype. On the other hand the "hedonic-quality" was guided as quite good even. Indeed the results are strongly scattered (see the standard divergence on page 72), for more detailed and statistically relevant evaluation of how the idea of ScratchTab is perceived by the users - in-depth user tests are necessary. However this goes beyond the scopes of the current work.

Statements

To describe the atmosphere during the test, some exemplary statements, made by users during the tests are listed here. This may be useful to judge the relationship of the users to the ScratchTab idea (see the conclusion on page 76).

- "Ah, it's a shame that I can't copy this... (stack)"

⁸<http://www.attrakdiff.de/>

	Average value		Standard divergence
technical	-1.714	human	1.979
complex	0.714	simple	1.979
impractical	1.857	practical	0.833
laborious	0.857	directly	1.641
unpredictable	1.000	predictable	0.926
confusing	0.714	clear	1.578
unruly	1.571	manageable	1.178
insulating	1.714	connecting	0.881
layman	1.000	professional	1.414
bad style	0.143	stylish	0.990
inferior	1.714	valuable	0.700
exclusionary	1.857	inclusive	0.990
separates me from people	1.714	connects me with people	0.700
not presentable	2.571	presentable	0.495
conventional	2.571	original	0.728
unimaginative	2.429	creative	0.495
cautious	1.571	brave	0.495
conservative	2.714	innovative	0.452
lame	1.714	absorbing	1.278
harmless	1.429	challenging	1.917
conventional	2.571	novel	0.495
pleasant	1.857	unpleasant	0.833
ugly	-0.143	beautiful	1.726
unlikeable	1.857	likeable	0.833
rejecting	1.857	inviting	0.639
bad	2.286	good	0.881
repulsive	2.286	attractive	0.700
discouraging	1.714	motivating	1.278

Table 3: Evaluation results of the AttrakDiff form. The average values and their standard divergences are computed by using the ratings of all word pairs by all users. The values are from the range -3 and +3.

- "Cool, it's like Kinect ⁹ or Wii ¹⁰ ..."
- "On the PC it would not be so interesting"
- "Somehow I am afraid of trying the commands out"
- To the robot: "Hey, I said 45 degree..."

Design issues

The following design issues were recognized during the tests:

Intuitive drop Problem

⁹<http://en.wikipedia.org/wiki/Kinect/>

¹⁰<http://en.wikipedia.org/wiki/Wii/>

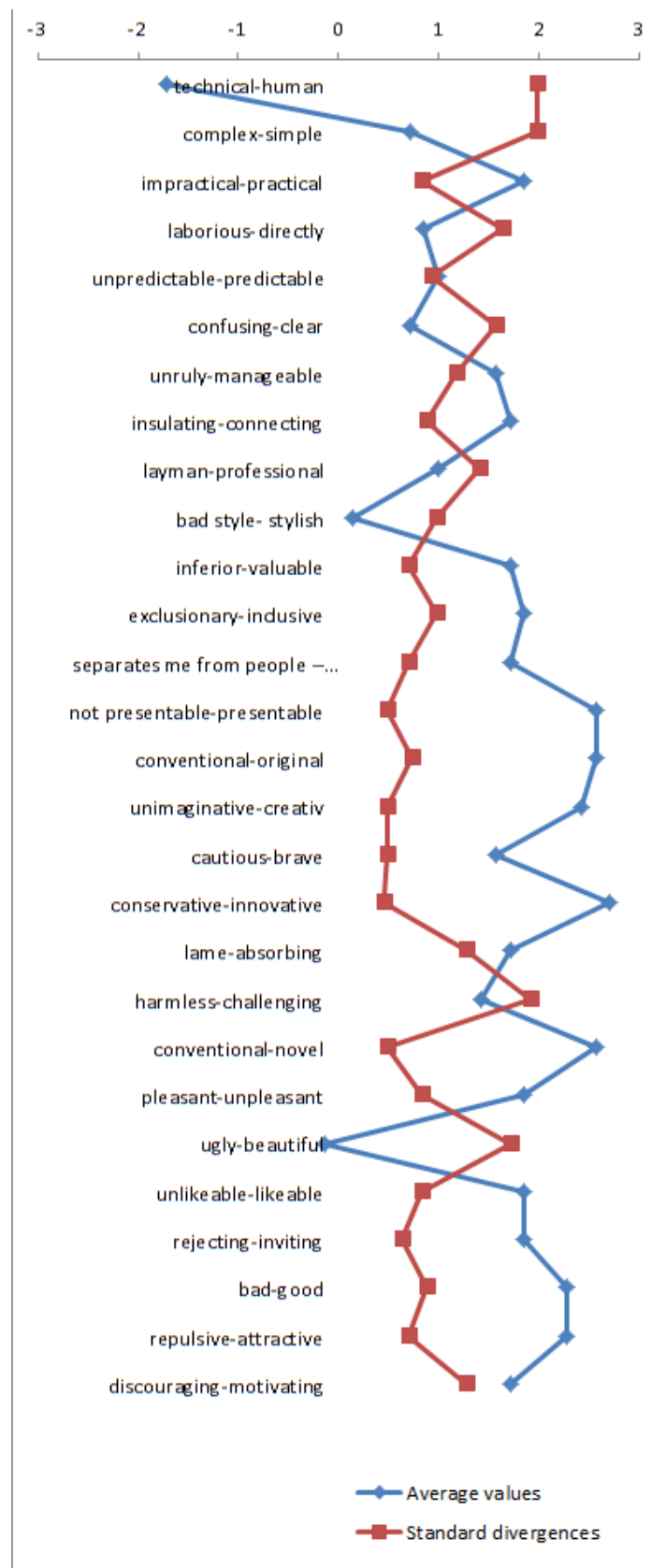


Figure 48: Average values and standard divergences from the AttrakDiff form. The german version of this AttrakDiff word pairs was presented to the users during the tests. The german version together with the precise values is located in the Appendix 4. PQ is pragmatic quality. HQ-I is hedonic quality – identity. HQ-S is hedonic quality – stimulation. ATT is attractiveness.

Problem The main issue in ScratchTab prototype usage seems to be in snapping the blocks together. During the user tests the system was designed to trigger the drop of one block into another, when user's finger has been moved and released inside of the drop area (slot), see figure 49a.

The tests have shown that nearly every user looks at the dragged block and its constraints, rather than at the own finger. Because of a frequent occurrence of the "drop one block into another" this usability issue has probably negatively affected the usability ranking the most and should be repaired in the next prototype.

Solution During the drag operation blocks notch, which is a usual constraint of the most blocks, is automatically moved under user's finger. After this change the location of the finger should have been consistent with the location of block's constraint, see figure 49a.

Result1 The drop has become more intuitive, since the error where user moved in vain the block's notch to block's projection in order to connect two blocks did not occur any more, but the finger was now on the drop area, which hid the slot, when drop was performed, so that the user did not see the visual feedback which was performed by the slot.

Solution2 The block have been made more transparent to switch user's focus from the block to the finger and to uncover slots during the drop operations.

Result2 This improved the situation, since now the feedback performed by the slot was only covered by the finger. However, the problem of covered slot feedback still occurred.

Conclusion To solve this problem the block should be visible during the drag operation, since the user seems to expect the slots to react on the dragged block, not on the finger which performs the drag. The blocks should snap together when the notch of a block touches another blocks projection. The finger should be located in the center of the currently dragged block, so that the finger won't cover the place of contact with other blocks. For both see figure 49b.

Building blocks bottom up Problem

Problem Two of 7 users have tried to build a stack bottom up, by connecting the dragged block from the top to the block on the workspace.

Conclusion In the next prototype it should be possible to snap blocks together by connecting the bottom-projection of the dragged block with the top-notch of the block on the workspace. In the prototype which was used for the user tests it was only possible to snap the blocks together by joining the top-notch of the dragged block with the bottom-projection of the block on the workspace.

Mark the stack as a unit Problem

Problem Repeatedly there were problems with the execution of the stack. The blocks were dropped to the workspace, under other blocks, so that the blocks stayed unconnected. However the users assumed the blocks as connected because there is no way to see whether blocks, which are already on the workspace, are connected or not.

Conclusion There should be some visual feedback, which can mark the stack as a unit. A line going around the stack would be appropriate mark according to the gestalt law of closure, see [Soegaard, 2010].

Touch workspace to close keyboard

Problem During the user tests a short tapping onto the workspace was performed by users, which had previous touch interface experience, in order to close the virtual keyboard. This feature was not implemented at that time.

Solution The expected gesture was implemented after the third user test.

Conclusion In the following user tests this gesture was used repeatedly by the touch experienced users.

To eliminate the "intuitive drop" and the "building blocks bottom up" issues, a redesign of the ScratchTab architecture is required. Both scenarios require drag shadow feedback to know the position of the top-notch and the position of the bottom-projection in order to let them interact with the block on the workspace. The

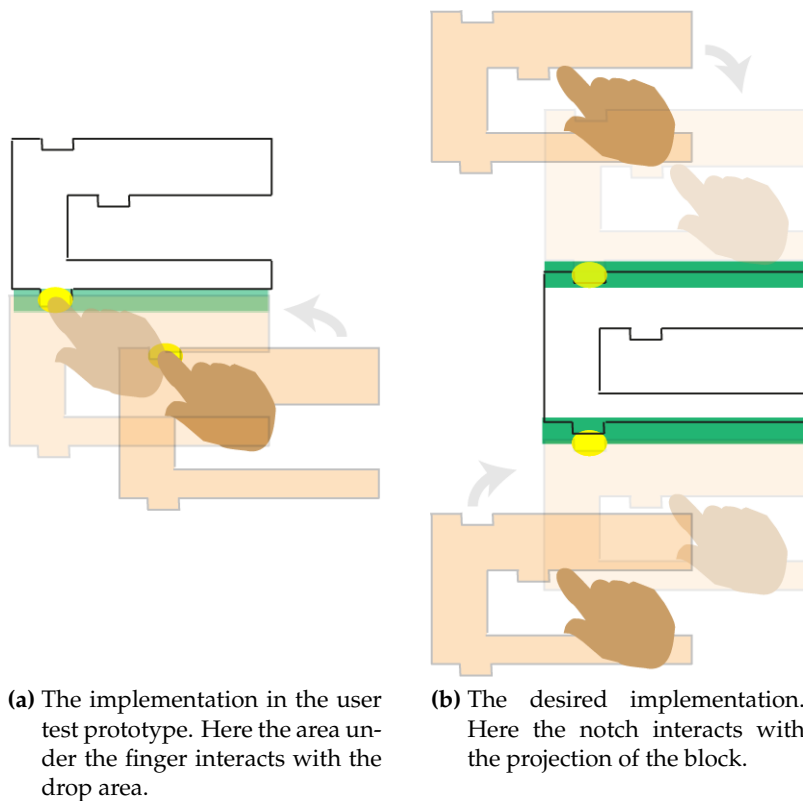


Figure 49: Block snapping gestures. The green area is the drop area of the block on the workspace. The yellow area is the point, which interacts with the drop area.

prototype, which was used for the user tests, uses the android implementation of drag shadow where interactions with the drag shadow are not possible. The redesign of the "drag and drop" gesture would rise the usability of ScratchTab and would probably move the project into the "desired" part of the diagram 47.

7 Conclusion and Outlook

This chapter addresses the research question and discusses the future works of ScratchTab.

In the foregoing chapter the evaluation of the working prototype was described, including the results of the user tests. The evaluation was the final part of the project.

The first part of this chapter summarizes all the previous chapters and refers the research question.

The second part of this chapter contains the outlook which suggests the direction in which ScratchTab may develop in the future. Listed proposals may be implemented in the next ScratchTab prototypes. The outlook is separated in two parts too: It starts with a short summary of the improvements, which revolved themselves as necessary during the user tests, as discussed in the previous chapter. The second part of outlook contains system proposals for the graphical interface and changes which are not a direct part of the graphical user interface.

7.1 Conclusion

The project results according to the research question.

Research question

Here the information from all the previous chapters is summarized to answer the research question of the current work: "Is Scratches building block metaphor transferable to tablets and are possible users more motivated using a mobile device instead of a static desktop-pc?"

Summary

This project has made the preliminary work in creating a working prototype and doing some basic evaluation. The evaluation results are presented in detail in the evaluation chapter on page 65. They shown, that users were attracted by the idea of ScratchTab, some of them compared the prototype with innovative games consoles like Wii or XBOX-Kinect demonstrating that the process of programming was perceived as playful.

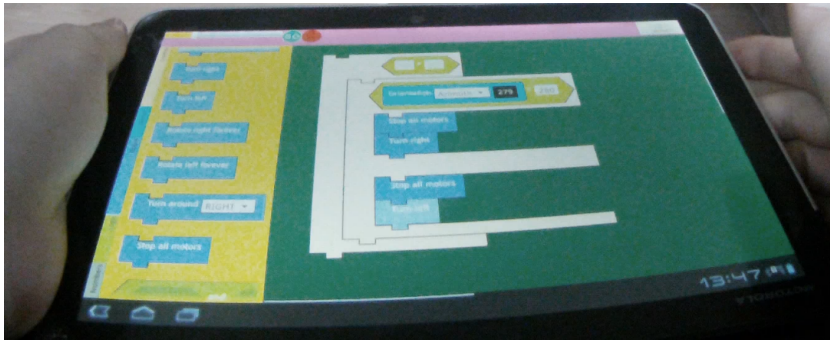


Figure 50: ScratchTab during the user tests. Quite complicated constructions were build during the user test.

There were even statements about the superiority of ScratchTab's way to learn programming compared to the way of learning programming on PC, without any previous addressing of this topic by the user test conductor.

Further the prototype was graded quite high on the "hedonic quality" scale(see figure 47), which indicates the high desirability and excitement factor of the system. However the usability was graded as medium, perhaps because of the existing prototype restrictions (All design issues are described in the evaluation chapter on page 65 and in the future-works chapter on page 78). Those restrictions have probably deflected user's focus from the idea of ScratchTab to the prototype's design issues. For this reason further development cycles are recommendable, in which the existing usability issues can be eliminated and more detailed user tests can be performed, with more statistically relevant amount of participants.

At the current stage of development the above average attractiveness of the project (1.67 on the Scala from -3 to +3 and the high hedonic factor of 1.83 on the Scala from -3 to +3 demonstrate, that the idea of ScratchTab is perceived as exciting. Further, since relatively complicated constructions (see figure 50) were created during the user tests without any problems, except of the removable ones which were described previously - one can say that the block metaphor is transferable from the PC to the touch interface. The comparison between the PC and the ScratchTab's interface metaphors requires the elimination of the design issues and can be answered during the later design cycles, when a design-issue-free prototype will be available. But even at the current stage of development users have stated, that they find it more exciting to learn programming by using ScratchTab than by using a PC.

Conclusion

7.2 Outlook

What can be done in the future to improve the current project.

The Outlook is separated in two parts as following: First part starts with a short summary of the improvements, which revolved themselves as necessary during the user tests, as discussed in the previous chapter. The second part contains system proposals, which are not a result of the user tests. The second part is further separated in changes to the graphical interface and changes, which are not a direct part of the graphical user interface.

GUI improvements after user tests

Intuitive drop This problem occurs when user connects two blocks, see page 73 in the previous chapter. Interactive drag shadows are required in order to trigger the drop, when specific parts of the shadow touch the blocks.

Building blocks bottom up This is a logical extension to the existing possibility of building the blocks bottom up, as described in the previous chapter on page 73. The drop should occur, when the drag shadow touches the block at the top. Again an interactive shadow is required.

Mark the stack as a unit This would add the missing feedback, when the combination of blocks fails, as described in the previous chapter on page 74.

Other GUI improvements

Nesting of blocks Current block nesting is perceived as messy, as shown by the survey, described on page 26. The desired implementation of a uniform block width was not implemented due to too small size of the prototype device. If bigger devices will be available for future prototypes – the uniform block size should be implemented for stacks to be perceived as more attractive.

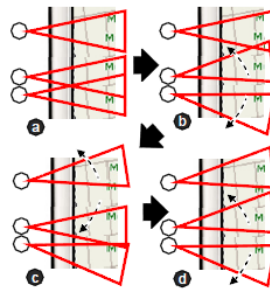


Figure 51: Displaying off screen objects with "wedge". For further informations about see [Gustafson et al., 2008]

Local popups over workspace It is not yet possible to display a block feedback with respect to the block's location, near the block on the workspace. This feature should be implemented to allow the feedback from more than one block simultaneously. For the current prototype the feedback is displayed as a global tooltip at the bottom of the screen.

Workspace extension The workspace, which in the current prototype has a fixed size should grow if necessary, e.g. if a block is placed near the end of the workspace. This feature will be necessary for large ScratchTab programs with multiple stacks. To allow such programs - the stability of the prototype should be increased and the save feature implemented, to allow working on one program for a long period of time.

Wedge The problem of location of stacks on the workspace did not appear during the user tests in the current project, probably because of too low level of difficulty. For more complicated and long-termed tasks this problem probably will become more relevant. To support the location of stacks on the workspace the workspace can use a pattern as background, which the user can use for orientation. Additionally the "Wedge" pattern can be used, see figure 51 or [Gustafson et al., 2008] for details.

Programming concepts

Variables The concept of variables was not yet implemented because the accent of this project was on the interface development. For this purpose the Variables were not necessary. If projects with the accent on teaching the programming

basics or on developing more complex programs by using ScratchTab variables will more important.

Proposed non graphical features

Save This feature is a requirement for the development of complex programs, since the development of such programs will probably be interrupted by pauses.

Undo This may be an important feature to increase the perceived safety of the system, for users to be braver when trying out ScratchTab features. Such a problem occurred during the user tests as seen in the user statements during the user tests.

Copy Blocks Important feature, which was mentioned in the user statements. This feature is required to reduce the specialisation level of the blocks on the block pane, which would reduce the number of blocks at the same time. During the user tests the specialisation of the blocks in the ScratchTab prototype was quite high. E.g. there were blocks to move the robot forward and backward, turn left and right etc. Instead the blocks to control three motors could be introduced, which would be used to compose the special movements. However the construction of higher level blocks would take some time. In order to not letting the users construct the movements again and again - the copy feature is required. With ability of copying blocks the needed construction would be done once and then copied as often as needed.

For more precise evaluation of the ScratchTab concept and for statistically more significant results - wider user tests may be necessary. Previously to the wider user tests the known design issues, listed under "graphical interface improvements resulting from user test" should be removed.

8 Related work

Many languages [Kelleher and Pausch, 2005] do exist which position themselves as easy to learn and suitable for beginners. Many of them were examined during introduction into field, this section provides a summary on projects and papers, which influenced ScratchTab the most.

Scratch

Since this project is based on Scratch[Resnick et al., 2009] - this is the project which influenced ScratchTab the most. Scratch is an educational language which uses the building block metaphor to overcome the most common problems which people do have when they learn programming. Above Scratch was already described in detail, so here the Scratch main characteristics are listed shortly again:

- Scratch prevents users from making syntactical errors, by using shape constraints.
- Scratch has a very uncomplicated one screen interface, which does not need much introduction.

This project tries to be aware of these qualities and not to lost them when customizing the metaphor for the tablet PCs.

8.1 Google App Inventor

Google App Inventor¹¹ influenced ScratchTab as a negative example. This project uses the block metaphor to build applications as Scratch does. Unfortunately blocks in App Inventor do not wrap the contained slots, instead they have the slots on the sides so that the execution order of the blocks is not that obvious as in Scratch, where first all the blocks inside the current block are executed, and then the next block in the top-down order is visited. App Inventor's approach introduces two possible execution directions: top-down and aside.

By fully wrapping the contained blocks, Scratch makes the child blocks looking more related to their parent which can be explained by the gestalt law of closure, law of common region. App Inventor do not benefit from this effect.

¹¹<http://info.appinventor.mit.edu/>

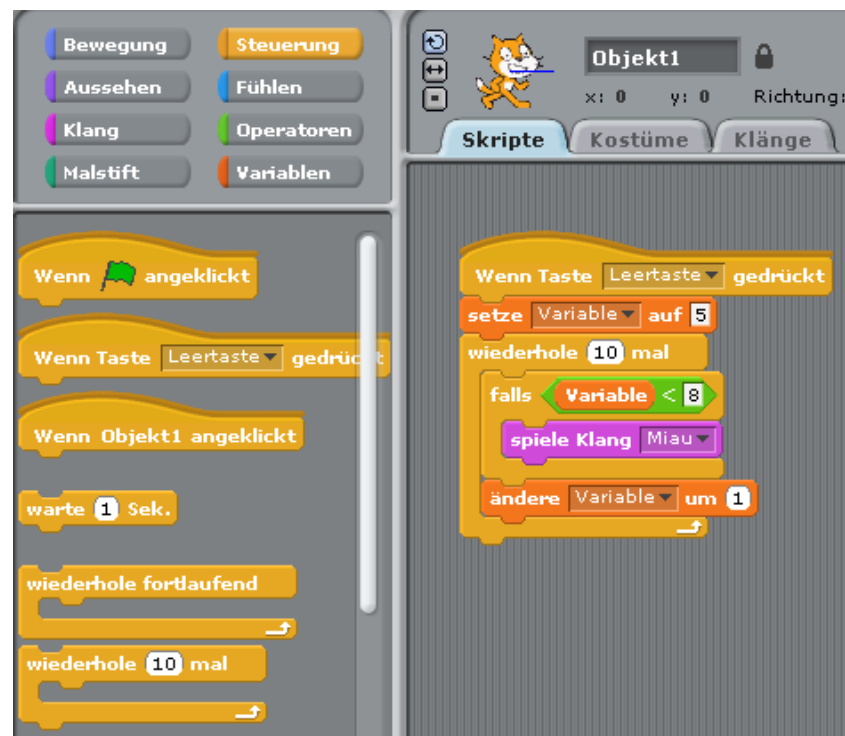


Figure 52: Scratch. On the right side of this picture you see a program, put together by dragging puzzled blocks to the virtual desktop.

Further the App Inventor gives up the most important advantage of the building block metaphor: it allows the combination of syntactically inappropriate blocks, so that syntax errors unnecessary become possible (see Figure 53).

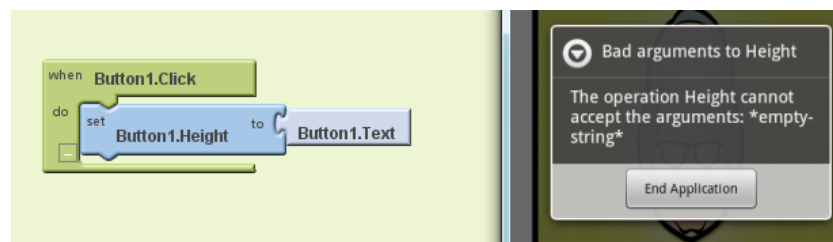


Figure 53: Google App Inventor. Combining syntactically inappropriate blocks is possible.

During this project Google gave up the support App Inventor.

8.2 Open Blocks

Open Block¹² on picture 54 is a open-source Java library for creating programming blocks based languages. Google App Inventor uses this library. During the initial development stage ScratchTab

has used some Ideas in organizing the block stack. However during the development the the stack structure has lost any commonness. Since issues like suboptimal child-wrapping of the Google App Inventor are anchored in Open Blocks, it was not taken over for the current project and an own solution was developed.

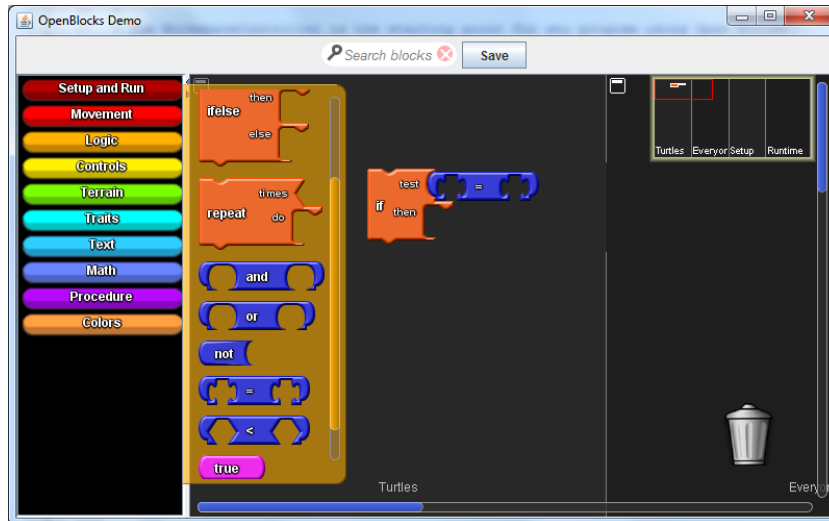


Figure 54: Open Blocks. An open source block library.

8.3 TouchDevelop

TouchDevelop¹³ is a Microsoft project which has the objective to create a script programming language for Windows Phones. In so far as this is known, it is the only serious project about programming languages for touch driven mobile devices, which is under active development today.

¹²<http://education.mit.edu/openblocks/>

¹³<http://research.microsoft.com/en-us/projects/touchdevelop/>

9 Appendices

9.1 Attached disc

The data located on the attached disc.

- Appendix 1 - Block layout
- Appendix 2 - Browser prototype
- Appendix 3 - JavaDoc
- Appendix 4 - Usertests Evaluation
- Appendix 5 - UML diagrams ¹
- Appendix 6 - Latex sources
- Appendix 7 - The thesis as PDF
- Appendix 8 - Project Sourcecode
- Appendix 9 - Application as APK

¹Project's UML diagrams were created by using an open source UNL editor available under sourceforge.net/projects/violet

10 Bibliography

Printed References

- [AdMob, 2011] Google AdMob (2011). "Tablet Survey". In: URL: <http://services.google.com/fh/files/blogs/AdMob%20-%20Tablet%20Survey.pdf>.
- [Colle and Hiszem, 2004] Herbert Colle and Keith Hiszem (2004). "Standing at a kiosk: effects of key size and spacing on touch screen numeric keypad performance and user preference." In: 47.13, pp. 1406–1423. URL: <http://www.ncbi.nlm.nih.gov/pubmed/15513716>.
- [Fitts, 1992] Paul M Fitts (1992). "The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement". In: *Journal of Experimental Psychology: General* 121.3, pp. 262–269.
- [Gould et al., 1990] John D. Gould et al. (1990). "Using a touch-screen for simple tasks". In: *Interact. Comput.* 2, pp. 59–74.
- [Gustafson et al., 2008] Sean Gustafson et al. (2008). "Wedge: clutter-free visualization of off-screen locations". In: *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pp. 787–796.
- [Hassenzahl, Burmester, and Koller, 2003] Marc Hassenzahl, M. Burmester, and F. Koller (2003). *AttrakDiff: Ein Fragebogen zur Messung wahrgenommener hedonischer und pragmatischer Qualität*.
- [Hatscher, 2001] Michael Hatscher (2001). "Joy of use - Determinanten der Freude bei der Software-Nutzung". In: *Mensch & Computer*.
- [Kelleher and Pausch, 2005] Caitlin Kelleher and Randy Pausch (2005). "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers". In: *ACM Comput. Surv.* 37, pp. 83–137.
- [Kieras, 2001] David Kieras (2001). "Using the Keystroke-Level Model to Estimate Execution Times". In: *University of Michigan*.
- [Maloney et al., 2010] John Maloney et al. (2010). "The Scratch Programming Language and Environment". In: *Trans. Comput. Educ.* 10 (4), 16:1–16:15.
- [Mindstorms, 2006] Lego Mindstorms (2006). *Lego Mindstorms NXT Direct Commands*. Version 1.00. <http://mindstorms.lego.com/en-us/support/files/default.aspx>.

- [Monroy-Hernández and Resnick, 2008] Andrés Monroy-Hernández and Mitchel Resnick (2008). “FEATURE: Empowering kids to create and share programmable media”. In: *Interactions*, pp. 50–53. DOI: <http://doi.acm.org/10.1145/1340961.1340974>. URL: <http://doi.acm.org/10.1145/1340961.1340974>.
- [Norman, 2002] Donald A. Norman (Sept. 2002). *The Design of Everyday Things*.
- [Parhi, Karlson, and Bederson, 2006] Pekka Parhi, Amy K. Karlson, and Benjamin B. Bederson (2006). “Target size study for one-handed thumb use on small touchscreen devices”. In: *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*. MobileHCI ’06, pp. 203–210.
- [Raskin, 2000] Jef Raskin (2000). *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley Professional.
- [Resnick et al., 2009] Mitchel Resnick et al. (2009). “Scratch: programming for all”. In: *Commun. ACM*, pp. 60–67. DOI: <http://doi.acm.org/10.1145/1592761.1592779>.
- [Schedlbauer, 2007] Martin Schedlbauer (2007). “Effects of Key Size and Spacing on the Completion Time and Accuracy of Input Tasks on Soft Keypads Using Trackball and Touch Input”. In: *Proceedings of the Human Factors Ergonomics Society 51st Annual Meeting* 51.5, pp. 1–5.
- [Sears, 1991] Andrew Sears (1991). “Improving Touchscreen Keyboards: Design Issues and a Comparison with Other Devices”. In: *Interacting with Computers* 3.3, pp. 253–269.
- [Sears et al., 1993] Andrew Sears et al. (1993). “Investigating Touchscreen Typing: The effect of keyboard size on typing speed”. In: *Behaviour & Information Technology* 12, pp. 17–22.
- [Shneiderman, 1998] Ben Shneiderman (1998). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 1. Addison-Wesley.
- [Stanley, 2011] Morgan Stanley (2011). “Tablets demand”. In: URL: www.morganstanley.com/views/perspectives/tablets_demand.pdf.

Online References

- [Facebook], Facebook. *Facebook statistics*. URL: <http://www.facebook.com/press/info.php?statistics> (visited on 04/20/2011).
- [RWTH], Mindstorms NXT Toolbox RWTH. *Mindstorms NXT Toolbox*. URL: <http://www.mindstorms.rwth-aachen.de/trac/wiki/MotorControl#Limitationsofdirectcommandsformotormovements>.

[Soegaard, 2010] Mads Soegaard (2010). *Gestalt principles of form perception*. URL: http://www.interaction-design.org/encyclopedia/gestalt_principles_of_form_perception.html.

[Wikipedia], Wikipedia. *Wikipedia statistics*. URL: <http://www.wikipedia.org/> (visited on 05/06/2011).

[Youtube], Youtube. *Youtube statistics*. URL: http://www.youtube.com/t/press_statistics (visited on 04/20/2011).