Mobile Display of Knitting Patterns

Bachelor's Thesis

Bianca Ploch
540609

16 August 2016

Supervisor:
Prof. Dr. Debora Weber-Wulff

HTW Berlin
International Media and Computing (Bachelor)

Contents

Chapter 1

Introduction

Nowadays we find ourselves in a very different world than even 20 years ago. Following the rise of the mobile phone we got the smartphone, the handy pocket computer for young and old. With technological advance came better processing power, more storage for all the important pictures and videos in life, and the big display to view everything in nearly lifelike colours. Together with it came the advent of the mobile application, more commonly referred to as an app. The many apps available are helpful tools to manage our daily lives and needs. Google's distribution service for Android apps, the Google Play Store, surpassed the 1 million app mark in 2014.

But despite the many apps available, one is hard pressed to find apps concerned with knitting, and even fewer ones that present a clear user interface or adhere to modern design guidelines for apps. This thesis analyzed knitting related Android apps on the Google Play Store. Most of the apps I found there do not support the inputting or displaying of a knitting pattern chart.

The difficulty with displaying such a chart on a mobile device derives from the size of the device and its screen, which is a far smaller medium than a sheet of paper, on which pattern charts are normally printed. The charts are therefore too big to be viewed easily inside an app. The goal for this thesis is to research how a knitting pattern chart can be input and displayed on mobile Android devices, including both phones and tablets.

Chapter 2

Background

2.1 Definition of Knitting Pattern and Knitting Pattern Chart

A knitting pattern specifies a set of instructions outlining the steps necessary to create a knitted textile or fabric. For knitting a fabric the knitter uses two or more knitting needles and a long, continuous strand of yarn which he uses to form intersecting loops with, which in turn creates a textile or fabric. "Knitting is a conversion system in which yarn loops are interwoven to form a fabric" Raz 1993, p17. The type of loops the knitter uses as well as the kind of yarn determine the attributes of the knitted piece: elasticity, form and texture. A knitted fabric can be stretched in both horizontal and vertical directions, as well as the directions in-between. This makes it stand apart from woven fabric, which is created by layering two threads in an interlaced manner; the resulting cloth is very limited in its ability to stretch and be formed, excluding, of course, the usage of stretching threads. Knitting patterns can come in form of written instructions, usually with abbreviations used for the stitch terms, e.g. k2tog for the "knit two together" stitch, or in form of a pattern chart which consists of a grid filled with symbols. Both, written patterns and pattern charts, are generally split into rows, where each row has a definite number of stitches. Each cell in such a grid signifies a stitch in the pattern and the symbol displayed in a cell corresponds with the stitch that needs to be made in that place in the pattern. In what order the rows have to be knitted depends on the chart type; some charts display only the uneven numbered rows, which belong to the right side (RS) of the knitted fabric, and expect the knitter to knit the

return row on the wrong side (WS) inverse to the RS, i.e. knits would be knitted as purls and purls as knits. Other charts show all rows, the uneven numbered for the RS and the even numbered ones for the WS.

So far there doesn't exist an international standard for the symbols used in knitting charts or the abbreviations in written instructions. Symbols used by the industry usually vary depending on the region Raz 1993, p57. and it is the norm that a knitting pattern includes a glossary for the symbols and abbreviations used in the pattern. Singular exception to this is Japan, where there exists a Japanese Industrial Standard on knitting symbols used in the industry and for the hobby hand knitters: JIS L 0201-1995 Association 1995. This leads to the fact that Japanese knitting pattern charts are published without an index of the symbols used.

Other regional industry standards that Raz mentions in his book are the German Standard and the needle notation system, "the most explicit and accurate of all notation systems" Raz 1993, p58., which is solely used for industrial knitting and shows the positions of the needles of the knitting machine for each stitch.

2.2 Comparison of existing Android solutions

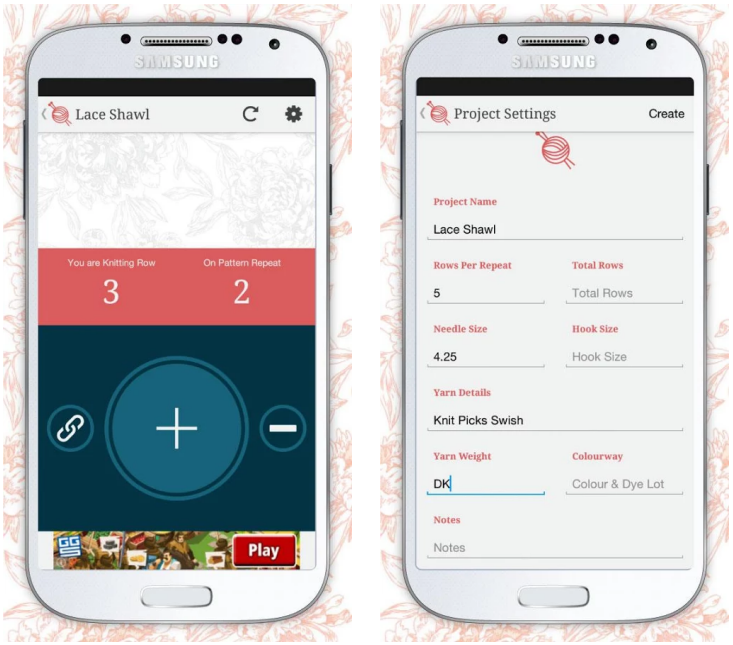
2.2.1 Android apps

When searching for the term "knitting" in the Google Play Store, Android's official source for Google-approved applications, few results will pop up. Next to a surprising amount of games about knitting, there are apps for knitting counters, knitting patterns and knitting instructions for those who wish to begin knitting. The following sections will look at the top five apps for creating and managing knitting projects with row counters and pattern display, as well as knitting chart creation.

knit tink — Row Counter by Jennifer K. Warren

Out of all most popular knitting apps, knit tink features the most modern and clean design. The app can be used for free or bought as an ad-free pro version. Features include the creation, editing, viewing, and deletion of projects, the setup of one row, and one repeat counter per project, as well as the unlinking of the row counter from the repeat counter. The free version of the app restricts the number of projects to three.

The developer announced an on-screen display of a knitting chart in PDF format as an upcoming feature.



(a) Row Counter (b) Project setup

Figure 2.1: Screenshots of the app knit tink

Knitting Counter by mkacki

Knitting Counter offers the same features a the knit tink app, the only differences being the layout of the user interface and the option to keep the phone from going into sleep mode, i.e., turning the phone screen off.

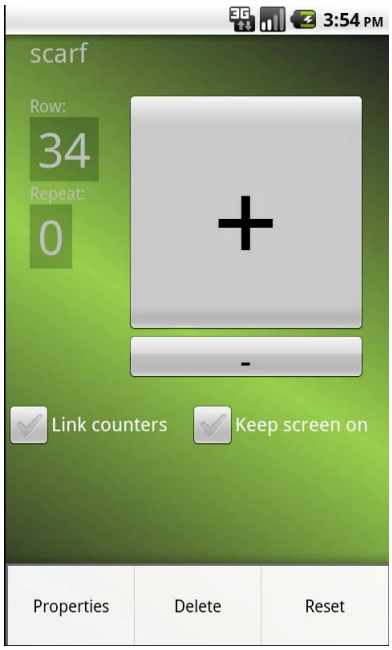


Figure 2.2: Counter of Knitting Counter

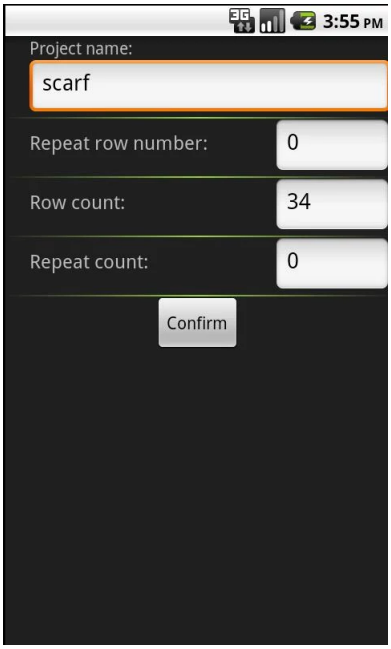


Figure 2.3: Counter set up screen of Knitting Counter

2.2.1.1 Knitting and Crochet Buddy by Colorwork Apps

The Knitting and Crochet Buddy contains a plethora of features related to knitting and crocheting. As is the standard with the previous apps, it offers the possibility to manage different knitting and crocheting projects, with each a row and a repeat counter per project. Users can also enter written instructions or add a picture of the pattern chart to be displayed on the counter screen.

Additional features include, but are not limited to: yarn and crochet charts, an abbreviation chart, size charts for knitting needles and crocheting hooks, a project timer, a ruler function, and a flashlight.

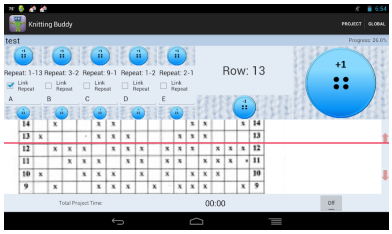


Figure 2.4: Counter of Knitting Buddy with written instructions

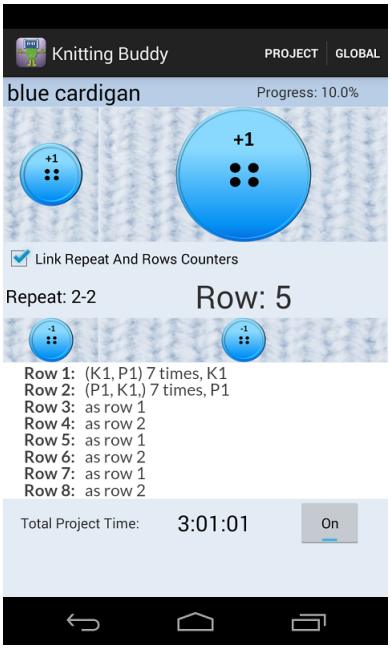


Figure 2.5: Counter of Knitting Buddy with chart picture

2.2.1.2 BeeCount knitting Counter by knirirr

BeeCount differs from the standard of one row counter per project in that it allows multiple counters. Those counters can be for parts of the knit piece that belong to the same project, as is the case for a knitted sweater, for example. These counters within a project can be linked together, so that increases or decreases in one counter affect the row number of another counter (figure 8). Furthermore, alerts can be set on counters to be triggered once the counter reaches a set number.

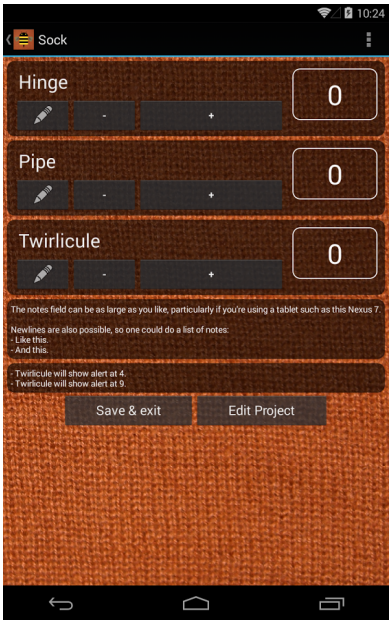


Figure 2.6: COunters of BeeCount

Figure 2.7: Counters set up of BeeCount

2.2.1.3 Knitting Chart Maker by Awesome Applications

When it comes to pattern charts, none of the aforementioned apps offers a solution besides the possibility to include a picture of a pattern. Therefore, I looked at Knitting Chart Maker, an app that focuses solely on the creation and editing of charts.

The app has over 30 stitch symbols that the user can use to create a pattern hart. The symbols are defined by the app and are not taken from a standard. The user can't devise their own stitch symbols. Symbols can be used by selecting the symbol in the left-hand menu and then transferred onto the grid by tapping on a cell. Alternatively, the user can select the paintbrush button on the top-left menu and use their finger to paint the symbols onto every cell touched in a swiping motion, not unlike drawing with a pencil. The whole grid is zoomable up to a certain zoom level.

While in-app, the user can purchase the pro version which allows them to save and export patterns. Charts can be exported in the form of written instructions or a picture. Included are also various sharing features, such as uploading the saved chart to Dropbox, or sharing a chart with a friend, who can then open that chart in their paid copy of the app.

The app is locked in landscape mode and the chart dimensions are limited to 50 x 50. The pattern chart grid is implemented using OpenGl's canvas and drawing images at the cell positions.

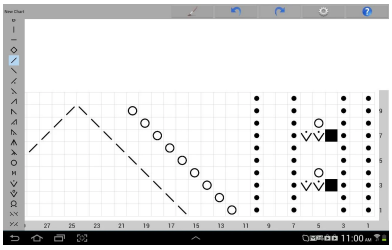


Figure 2.8: Chart editor of Knitting Chart Maker

2.2.2 Other

2.2.2.1 KnitML by Jonathan Whitall

KnitML is an XML based format for describing the knitting process from beginning to the finished product. With KnitML The project aims to establish an international

standard for knitting pattern expression (`http://www.knitml.com/blog/static.php?page=about-knitml`). He aims to do so by using the Knitting Expression Langauge, KEL, that he defined (`http://www.knitml.com/docs/users-guide.html#d0e246`). KEL is based on the Groovy programming language (`http://groovy-lang.org/templating.html`) for the Java platform and the GroovyMarkup architecture.

The following KEL expression

```
Pattern {
    generalInformation
}
```

Would result in

```
<pattern>
    <general-information/>
</pattern>
```

The project has not seen updates in any form since 2013 and I presume it discontinued. A beta of an editor program for KEL and its resulting XML can be found on the homepage of the project, knitml.com.

Chapter 3

Requirements

3.1 Functional Requirements

The requirements for this thesis are formed from interviews conducted with volunteers at the beginning of this thesis. Three participants, stemming from both the author's acquaintances as well as from a poster posted publicly nearby the HTW's campuses, have been interviewed. Prerequisite for a participant in such an interview was a proficiency and an interest in knitting. During a time frame of 45 to 60 minutes the participants were asked to answer a set of questions concerning their knitting experience as well as what features they would like to see in an app aimed to aid them during the creation and viewing of a knitting pattern chart. For a transcript of these interviews consult the appendix A. From these interviews a set of functional requirements was extracted and listed below Chapter 3.1.

#	User Story	Functional Requirement
1	As a knitter I want to be able to see the knitting pattern chart on my phone while knitting	Display of knitting pattern chart that is easily usable while knitting
2	As a knitter I want to create my own charts in the app both in a grid format and a row format	Pattern editor for creating and editing patterns

#	User Story	Functional Requirement
3	As a knitter I want to transcribe charts from paper into the app with both grid and row formats	Pattern editor
4	As a knitter I want to have a list of all the patterns in the app and add and remove patterns from that list	CRUD for patterns and showing list of patterns
5	As a knitter I want to convert metric units for needle sizes, yarn weight and length to imperial and vice versa Unit converter in app	Unit converter in app
6	As a knitter I would like to enter a set of written knitting instructions and be able to see each individual instruction while knitting and jump to the next instruction with a button press	Editor for written instructions and view of them to be used while knitting with button or voice command
7	As a knitter I want to use my phone to count the rows I knit	Row counter
8	As a knitter I would like to be able to look up the explanations and visual instructions for different kinds of stitches while inside the app	Glossary of stitches with explanations and instructions
9	As a knitter I want to have a way to jump to the row I'm currently on in my knitting pattern and to get back to the default zoom level	Button for resetting the zoom level and to jump to current row when viewing a pattern
10	As a knitter I want to be able to take pictures of the finished, knitted products of a pattern	In-app camera and function for adding images from disk
11	As a knitter I want to be able to see pictures of the knitted products of a pattern	Gallery for knitted products from a pattern

#	User Story	Functional Requirement
12	As a knitter I want to have all my knitting projects with their details (pattern, required needle size and yarn, etc.) easily accessible in one app	Knitting project management functions
13	As a knitter I want to be able to use the row counter with another app in the foreground	Have row counter increase and decrease button in notification bar when knitting app is not the active app
14	As a knitter I want my screen to stay on until I exit the app	Force screen to stay on while in-app

Within the context of this thesis the focus lies on the functional requirements #1, 2, 3, 4, 8, and 7. The protoype of the app will present a functioning editor as well as a viewer for knitting pattern charts. Two input styles will be available for both viewer and editor: a grid style and a row style. The in the app generated pattern will be stored on disk and will be accessible with CRUD operations within the app. The viewer will have a row counter next to the displayed pattern chart. Buttons for switching between the view styles will be present in the editor as well as the viewer. The option to import and export pattern files will be available as well in case the user wants to move their patterns to or from a different Android device.

After these requirements will have been fulfilled and time allows, further features for the app will be: a button for resetting the zoom level and jumping back to current line in the pattern, a row counter increase and decrease button outside of the app and the option to force the screen to stay awake while within the app.

3.2 Non-functional Requirements

The prototype will store the pattern files locally on the device the app runs on. All prototype operation will run locally, connectivity to the internet is not needed. Internet connectivity is an option for a later version of the prototype, e.g. for backing the pattern files up to cloud storage and sharing patterns. Since this thesis focuses on the User Interface (UI) part of an app, storage will be restricted to simple, local solutions. This is also done to better fit the time restraints placed in this thesis.

Chapter 4

Design

The desired outcome of this thesis will be an usable Android app prototype with working CRUD functions for knitting chart patterns and a row counter functionality while viewing a pattern. This prototype is intended as an aid for knitters of all backgrounds during their respective knitting projects. Patterns will be saved locally as a JavaScript Object Notation (JSON) file on the device's internal storage. It would also be an option to store patterns on a server and let the app play the role of client, but that would not fit within the time constraints of this thesis. To give the user the ability to backup their patterns the app will support the import and export of pattern from external storage. For a detailed explanation of Android's concepts of internal and external storage see section 5.2.5. Patterns exported will be accessible by the user and can be handled in whatever way the user sees fit to, for example, share or upload a pattern.

A chart pattern will consist of a set number of rows and columns. Each cell of the grid contains a symbol representing a knitting stitch. Created pattern are stored locally on the device and can be manipulated by the user in-app, as CRUD operations apply. Creating, editing and viewing patterns will be based on two shared visual formats for the pattern: a grid format and a row format.

The grid format will display the pattern in a grid, simulating the most common form of commercial distribution for knitting chart patterns on both analogue and digital media. Manipulation of the pattern content will be possible through a software keyboard containing the stitch symbols. A symbol can be selected and then applied to cells in the grid via touch. The symbol will stay active until the user selects a different symbol. The grid size can be changed with a button which opens a dialog where the desired amount

of rows and columns can be entered. On confirmation the grid will shrink or expand to the set dimensions. Any symbols lying outside of the new bounds will be deleted, whereas new cells will be empty.

Similar to the grid format the row format will display the pattern rows, but will forego the representation of the columns. Instead the cells of a row will be summarized in such a way, that consecutive, identical symbols will be represented by a number value equal to the count of the symbols and followed by the stitch symbol. Rows in this format can be edited like a conventional text editor - a movable cursor to show where further user input will be inserted and text selection functions for multiple character deletion, copying and pasting will be available. The software keyboard corresponding to this format will consist of the stitch symbols and a num pad, as well as an enter and a backspace key.

Viewing a pattern will come with a row counter below the actual chart pattern. This counter can be increased, decreased and reset by utilizing buttons. The current row will be indicated through a highlight on the pattern, marking the corresponding row in both grid and row format.

While editing or viewing a pattern, the row and the grid format will allow the user to 2D scroll, meaning both vertical and horizontal scroll. Additionally, the grid view can be zoomed and reset to default zoom and scroll. Switching between both formats while editing and viewing a pattern will be supported with a button. Upon switching the pattern will be saved. Renaming, deleting, saving, and exporting the pattern will be possible from within both formats with menu entries.

On app launch the list of patterns saved on the device will be shown. While viewing that list menu entries for exporting all patterns and importing a single pattern will be available. For the import the user can choose a file on the device from a file chooser. Exporting will export files to a set directory on the publicly accessible storage of the device. A list item will consist of the pattern name, an edit, and a delete button. A click on the pattern name will open the pattern in the viewer in row format with the default dimensions of 10 columns and 10 rows. Below the list will be a button to create a new pattern which will open a dialog for entering the new pattern's name. After confirming a name, the editor will open with the row format.

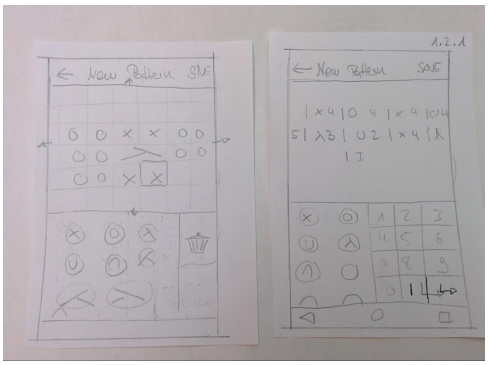


Figure 4.1: Editor screens for grid and row format with pattern name and save button in navigation bar

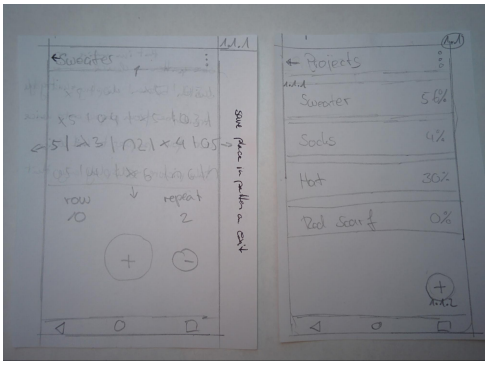


Figure 4.2: Viewer screen for row format and selection screen for stored patterns

Figure 4.3: Screens for editor and viewer without Android elements

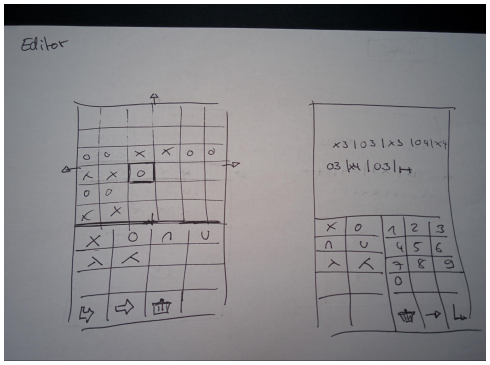


Figure 4.4: Editor screens for grid and row format

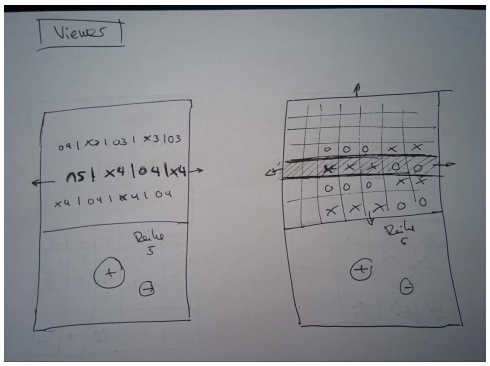


Figure 4.5: Viewer screens for grid and editor format with row counter

Chapter 5

Android Basics

5.1 Basic structure of an Android App

Android is an open source operating system for mobile phones and tables based on the Linus kernel (Android Developers 2016c). Android apps are distributed on Google Play, a service owned by Google. To build and Android app Google offers the Android Software Development Kit (SDK), containing sample projects, necessary Android libraries and an Android emulator. Additionally, Google recommends to use the official Android Integrated Development Environment (IDE) Android Studio, which is based on IntelliJ IDEA and offers many useful tools, including testing frameworks, a Graphical User Interface (GUI) for screen layouts, and the build tool Gradle (Android Developers 2016j). The concepts and Android components discussed throughout this chapter are taken from the Android Developer reference (Android Developers 2016k), training (Android Developers 2016h), and Application Programming Interface (API) guides(Android Developers 2016i) found online.

An Android app consists of at least one activity which can contain one or more fragments — both activities and fragments are basic Android application components. An activity, in the Android context, is a Java class that extends the Activity class. It manages a screen and is needed to display any user interface. Fragments implement a specific user interface or behaviour and should, ideally, be modular so they can be reused with multiple activities and in different screen configurations. Both components have their own lifecycles and events during the apps runtime. Activities feature methods such as onCreate(), onStart(), onStop(), and onFinish(), which can be overwritten

to implement logic that is executed at different points in it's life. Same applies for a fragment, but where an activity can stand alone, a fragment always needs to be attached to an activity; it is connected to that activity's lifecycle — is the activity stopped, the fragment will stop as well. When containing fragments, the activity's job is that of managing those fragments through getters and setters and orchestrating the communication between fragments, which is done with callbacks defined in the fragments.

Activities and Fragments can house ViewGroups and Views, which define different UI components for Android. Such a view would be a TextView, which displays one multiple lines of text. Viewgroups and views can be extended in custom classes to implement non-standard behaviour.

5.2 Basic Components of an Android App

5.2.1 Activity

The Activity class is needed to display any user interface and as such usually has a single purpose — handling a login would be such a purpose. An app consists of one or more activities that are in some way connected to each other (Android Developers 2016b). An activity can embed multiple fragments — those fragments then live in a ViewGroup inside the activity's own view hierarchy (Android Developers 2016g).

5.2.2 Actionbar

The actionbar located at the top of an app and has several important functions. It displays the application name or the title of an activity, houses the action buttons, and the action overflow. Action buttons should contain the most commonly and important actions used in an app (Android Developers 2016a). The action overflow contains action buttons that are hidden from plain view, either because the actionbar was not wide enough to show all buttons or because of a deliberate design decision. Such a decision is usually made when the button in question is connected to an action that is rarely used, e.g. renaming something, or when the action has far-reaching consequences and shouldn't be near buttons that are used frequently, lest the user accidently hits it. Such an action could be the deletion of the currently edited pattern, in case of an knitting app.

5.2.3 Fragment

The Fragment class ususally implements a specific user interface or behaviour and should, ideally, be modular, so that they can be reused within multiple activities or in different screen configurations. Just like an activity a fragment has its own lifecycle, see **Figure 5.1**.

A fragment's creation is always embedded in an activity (Android Developers 2016g) — forcing the fragment to pause or stop alongside its parent activity's lifecycle. Fragments can also house viewgroups and views and are intented to function as interchangeable modules, e.g. as UI modules for an app that runs on devices of varying sizes and that wants to present the user with a dynamic UI fit to suit the screen size (see **Figure 5.2**).

Android offers different fragment subclassse with predefined behavior, such as the DialogFragment class, that opens a fragment as a floating dialog by default (see **Figure ??**). Communication between different fragments needs to be handled by the activity, which manages the fragments. An activity communicates with a fragment by keeping a reference to the fragment and calling its public methods. On the other hand the fragment should not possess a reference to its parent activity — instead the parent activity should implement a callback interface defined inside the fragment (Android Developers 2016f). A good practive to enforce the implementation of a fragments callback interface is to check for its existence when the fragment is attached to the activity — example code proposed by the Android Developer guide concerning how to check for this can be seen in *Listing 5.1* (Android Developers 2016f).

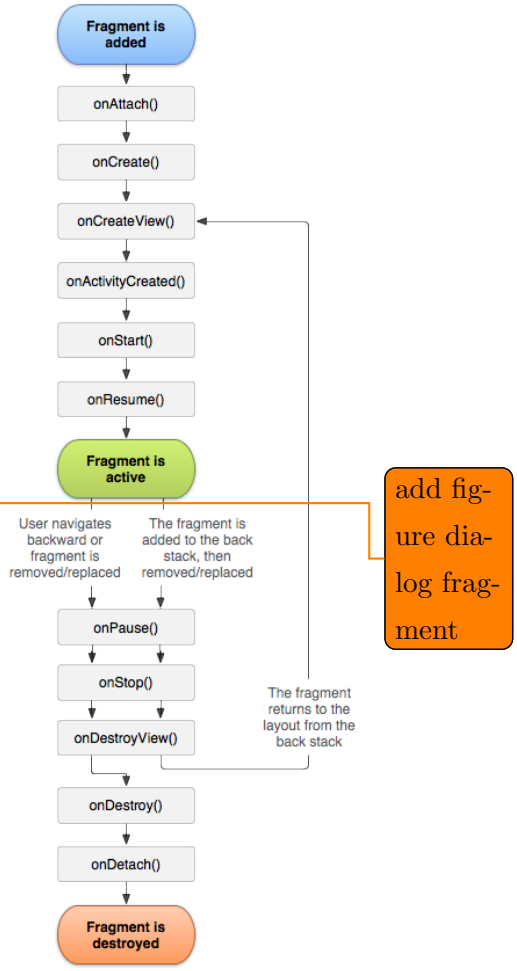


Figure 5.1: The Fragment life cycle

```
1 public static class FragmentA extends
  ListFragment {
2     OnArticleSelectedListener mListener;
```

```
3       ...
4       @Override
5       public void onAttach(Activity activity) {
6           super.onAttach(activity);
7           try {
8               mListener = (
    OnArticleSelectedListener) activity;
9           } catch (ClassCastException e) {
10              throw new ClassCastException(
                    activity.toString() + " must
                     implement
                    OnArticleSelectedListener");
11          }
12      }
13      ...
14  }
```

Listing 5.1: Example code for enforcing the implementation of a callback interface

5.2.4 View

Views are the most basic block that the UI is built from (Android Developers 2016n). A view's bounds are always rectangular and its position is defined by its top and left coordinates with the point of origin at the top left. It is the view's job to handle its drawing and event handling. For this the view has the predefined methods onDraw() and onTouchEvent(), respectively. The view class is also the base for viewgroups which in turn are the base for layouts, containers for other views or viewgroups. The views from a window are arranged in a tree structure. Views can be added to this tree statically, by specifying them in an Extensible Markup Language (XML) layout file, or dynamically, from code. Android comes with plenty of view subclasses, specialized in acting as controls or displaying

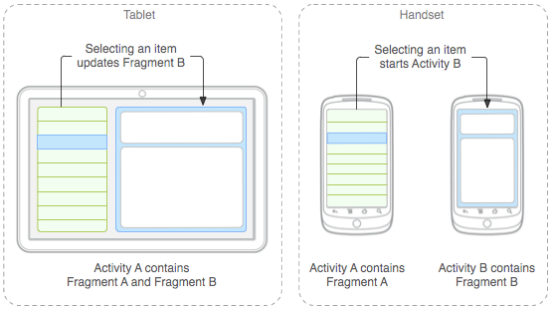


Figure 5.2: Two fragments of one activity and their layout on two different screen sizes

specific types of content, e.g. text or images. If the pre-existing views don't match a developer's needs, they can also implement a custom view to take control of the drawing and the event handling as it fits their requirements. For this the onDraw() method can be overridden and custom operations can then be executed on the Canvas object that is contained in the method parameters.

Android ships with many subclasses of View, specialized for displaying text (TextView), or a button (Button). The view class is also the basis for ViewGroups, to which the layouts, e.g. LinearLayout and RelativeLayout, belong to. Views in a window are bundled together as a tree with a layout being the top-most root. To add views to an activity or fragment the views can be declared in the corresponding XML layout or from code.

5.2.5 Storage

File storage in Android devices is separated into "internal" and "external" storage — this refers to the fact, that Android devices often times have a built-in, non-removable memory and an external, removable medium in the form of an SD or a micro SD card (Android Developers 2016e). This storage separation even exists on devices with only built-in memory — in such cases the storage is partitioned into "internal" and "external" partitions. This assures that the concept of two storages persists across all devices and API levels. The internal storage is inaccessible by the user under normal circumstances — exception to that is when the user has root privileges, e.g. on a rooted phone. This storage houses, among other things, files from apps, e.g. databases. These files are only accessible by the app that originially places them in the interal storage — neither user nor other apps can access them. Files are removed when the app they belong to is uninstalled. The external storage on the other hand is more public. Files placed here can be read and written by user and other apps and they even remain after the app they orignated from is uninstalled. When working with the external storage it is important to check that it is not currently used as Universal Serial Bus (USB) storage by a computer the device is connected to. Apps have by default read and write access to the directory they are installed in on the internal storage, but to access the external storage the app requires that the user grants the app a specific permission. This permission needs to be declared in the app's manifest file, a XML file that every app must have. This file contains information required by the Android system to allow the app to run, such as the activities contained in the app and the permissions the app requires. Beginning

in Android 6.0 (API level 23) apps targeting that Andoir version need to request and aquire dangerous permissions at run time (Android Developers 2016l), whereas before the app was given all permissions listed in its manifest upon agreeing to a dialog popup when installing the app. Dangerous permissions cover access to the user's private data or to affect areas where the user stores their data or data that other belong to other apps (Android Developers 2016l). Since the user can revoke permissions for apps at any given time the developer needs to take measures to keep the app running even when some features need to be disabled because of missing permissions.

Chapter 6

Implementation

Google's IDE Android Studio 2.1.2 was used for the implementation of the Android app prototype

6.1 Stitch symbols

There are different ways to display a stitch symbol in an Android app — this section will give an overview of the possiblities and the solution chosen for the prototype. Since the View class already defines a canvas object with dedicated functions for drawing image and text content in its onDraw() method, it offers a good starting point. To decide between using the image or text format for disoplaying stitch symbols, a further look into what each format entails is necessary. Using image resources

Image content needs to be specified in an Android project in the res directory under the drawable directory. This directory contains the image resources of the app, the drawables — this applies for icons, custom images, and other image content, excempting the launcher icon of the app, which is located in the mipmap directory. One way to use stitch symbols in an Android app would be to add every symbol as a drawable resource, and then draw those resources to the canvas of a view. For this a drawable would be needed for each individual stitch symbol, as well as way to map these drawable files to values that can be efficiently stored in a JSON file. The usage of many drawables in an app would also lead to an increase in app size, resulting in longer download times and larger storage demands. Both are an inconvinience to the user and can be problematic on older devices with less powerful hardware, making the app unsusable in the worst

case scenario. The other option for displaying symbols is to create a custom True Type Font (TTF) with glyphs for stitch symbols that is applied to text in the app. This is the solution used in the prototype. It offers several advantages over using image resources. For one, when using drawables it might be necessary to include several versions of the same file to assure that they are displayed correctly on devices with different screen densities (**android˙densities**). This is not needed for text with a custom font: the glyphs are defined by Beziér curves, which are correctly rendered by the system for the individual screen densities. The usage of a font also allows to write each stitch as a characters, simplifying the process of saving a pattern to a JSON file. For OpenType fonts, which TTF belongs to, Microsoft recommends 64000 as the maximum number of glyphs a font should contain (Microsoft Corporation 2014). This allows for a plethora of stitch symbols. The open source program FontForge [1] was used creation of the custom TTF knitting font used in the protoype. The knitting font contains a selection of 16 stitch symbols whose glyphs were defined by the author herself. The glyphs and their corresponding UTF-8 characters can be found in the Appendix A: .

ref here?

ref here!

6.2 Pattern

Using a custom font for the stitch symbols allows to represent patterns as combinations of characters. The actual pattern chart is saved to JSON as an Array of Strings, where each String represents one row in the chart. The Within the app prototype I use three different pattern formats: one for the Pattern POJO (Plain Old Java Object) which is used for the persistent storage, one for the row format, and one for the grid format.

The grid format uses a two-dimensional String array of columns and rows and can be likened to the way a pattern chart is displayed on paper. Each cell is filled with a String representing a stitch.

The row format is one String that is used with Android's EditText widget. The format implements the shortened pattern rows described in the design chapter of this thesis. An EditText can display multiline text and does so by adding a line feed
n at the appropriate place in the String. Rows can be distinguished by the
n following a combinations of letters.

The Pattern POJO contains a field for a String Array of pattern chart rows that,

[1] `https://fontforge.github.io/en-US/` (last accessed: 08-10-2016)

like the pure row format, use the shortened rows for representing a pattern. The pattern is converted between the different formats by using the class KnittingParser. This class provides static methods for the conversion between grid, row and POJO format.

6.3 Displaying a Pattern

6.3.1 Grid Format

When considering presenting data in a grid format, the most obvious solution is to first look at Android's own implementations of grids. Promising starting points for that are the TableLayout and the GridView class. After a brief investigation into the TableLayout, it quickly becomes apparent, that this layout is intended more as a way to position views, than to represent data in a grid. It can be likened to the HTML table tag (Android Developers 2016m), which is also used to position views within the constraints of cells, columns, and rows.

Android's GridView class, on the other hand, is designed with notion of displaying data. Each cell represents one data entry in a collection of data, the displaying of which is handled by an Adapter class attached to the grid view. Android ships with a specialized adapter for lists [2], as well as a BaseAdapter that can be subclassed for a custom handling and presentation of data. While this is a fitting solution for displaying symbols in a grid, it does not meet the requirements this project sets for the grid format. It is required that the column and row numbers are displayed next to the grid as axes. These axes should scale and scroll together with the grid, but should not be scrolled outside the visible area, since the user would not be able to know the cell position then. For one, the GridView class does not support frozen cells, columns, or rows — as far as the author of this thesis was able to research. If the column numbers were to be displayed in the first row of the grid, they would move offscreen upon scrolling the grid. A possible solution to this would to use text views to act as axes to the grid and to display the column and row numbers next to it. Problematic with this approach would be the fine-tuning required to math the visuals of the text view to that of the grid. Line height, text size and the synchronization with scrolls and zooms performed on the grid would need to match perfectly. Since this does not classify as intended behavior for

[2] `https://developer.android.com/reference/android/widget/ListAdapter.html` (accessed: 11-08-2016)

these views, a cohesive UI cannot be guaranteed. Furthermore, Android does not offer two dimensional scroll on any view except its WebView[3] – the GridView class natively only supports vertical scroll.

Therefore, in order to fulfill all requirements it makes the most sense to create a custom implementation of a view that supports the display of text in a grid, two-dimensional scroll, zoom, and axes that stick to the view bounds. Google's sample project *Interactive Chart*[4] and the corresponding training path[5] present an implementation example and were used as a guide line for the implementation of the class PatternGridView.java . This class keeps a reference to a two-dimensional array of strings which represents the pattern with its columns and rows. The grid and its axes are drawn by overriding the onDraw() method, calulating the postion of the correspoinding lines and numbers with the dimensions of the view and pre-defined, hardcoded values for cell width and margin. The current version of the PatternGridView uses a SimpleOnGestureListenerto compute two-dimensional dragging. Android defines two different scrolling types for views: dragging and flinging (Android Developers 2016d). Dragging is executed by draging a finger across the device's touchscreen and results in a moving of the view corresponding to the dragging direction and speed. This means, that no matter the velocity of the dragging gesture, the view won't keep moving once the user lifts the finger involved in the gesture. A fling on the other hand will keep moving the view with a speed and duration exponential to the velocity of the fling gesture, where the duration is calculated by introducing a friction to the scroll. This gesture can also be describe as a swiping motion performed on the touchscreen. Even after the finger has been lifted of the screen the fling continues to execute until it either runs its course or is interrupted.

reference here

code example 2d array to show columns and rows

ref here

The current version of the PatternGridView implements a simple dragging gesture and does not support flinging as of yet. For this a variable of type Pointis saved locally to represent the offset scrolled. It is initiliazed with the value (0,0) and updated on every motion event that is recognized as dragging. The necessary calculations for these updates are done by overriding the onSCroll()method in a custom SimpleOnGestureListener. This method has access to the horizontal and vertical distance scrolled, which is substracted from the offset. This is done because the value of the distance is positive

ref docs point class

ref

[3] `https://developer.android.com/reference/android/webkit/WebView.html` (accessed: 11-08-2016)

[4] `https://developer.android.com/shareables/training/InteractiveChart.zip` (accessed: 11-08-2016) A digital copy of this project can also be found on the CD attached to this thesis.

[5] `https://developer.android.com/training/gestures/scale.html#drag` (accessed: 11-08-2016)

when dragging towards the point of origin (the top left corner) and negative when dragging away from it. Therefore, the canvas needs to be translated in the opposite distance. The offset is then clamped to minimum and maximum values, to ensure that the grid will never completely move offscreen:

$$offset_{min} = (0, 0) \tag{6.1}$$

$$\begin{aligned} offset_{max} = (&width_{view} - width_{content} - 2 * margin, \\ &height_{view} - height_{content} - 2 * margin) \end{aligned}$$

where margin is the distance of the grid from the top and left view edges that is reserved for the axes text.

Similiarly to dragging, scaling is implemented with a SimpleOnScaleGestureListenerwith is connected to the view's onTouchEvent(). The scaling factor is then clamped at pre-defined maximum and minimum values and saved in a variable. During the drawing operations of the grid, axes, and text the scaling factor is then used to compute and draw the scaled visuals. On user touch on the grid the touched cell is calculated from the pixel position of the touch event and the currently selected stitch string is saved to that position in the two-dimensional pattern array. Following this the view is invalidatedand re-drawn with updated pattern.

ref here

ref to method in docs here

6.3.2 Row Format

requirements: editor should have standard text editor functionalities: copy, paste, cut, select, multi lines, scollable horizontally and zoom problem: texteditor has standard functions (cut, copy, select, paste) also has multi lines as well as horizontal scroll but doesn't support multi line and hori scroll at the same time if scroll, then only single line if multi line, then vertical scroll and wrap at the end of the screen, no content outside of visible area edittext default opens soft keyboard solving attempts: using custom edittext to suppress opening of soft keyboard create custom edittext override ondraw for drawing line numbers implement custom scroll zoom only scaled the canvas, but not the text realize now: shouhld have set text size, then it would have scaled correctly use library twoDScrollView as parent view to standard edittext only supports on child touch events got mixed up? use linear layout with custom scroll put textview

for line numbers inside put edittext inside next to textview calculate container width from edittext calculation often times wrong trying to scroll to cursor when typing goes wrong at end of line: edittext still calculates new cursor position in next line from default wrapping behavior when adding line at bottom: should scroll new line so that is visible: not working, edittext doesn't now new height bevor drawing also problem upon fragment first showing: edittext doesn"t now how many lines it has upon instantitian with text can't set container height and can't set line numbers can't scroll in viewer, can only scroll in editor after editing something as the edittext then updates and knows its height

6.4 Persistent Disk Storage

import: check file type??? not handled yet, app crash

There are multiple ways persistent storage can be implemented in Android. A common choice is to use a SQLite database, which Android natively supports. Another would be to save files to internal or external storage, which many phones have by way of SD card. I decided to save the patterns as files to internal storage — the pattern is saved in the default local directory Android gives the app. For this the Pattern POJO is serialized using Google's JSON library Gson, which handles the writing, reading and conversion of the files to Java objects and vice versa. Gson supports the usage of Java generics and can map JSON data to POJOs while maintaining inheritance hierarchies.

The Pattern POJO defined in the prototype contains fields for the pattern rows, the total number of columns and rows, and the current row from the row counter with getters and setters for all of the fields. A pattern object's default state after initialization contains a pattern of size 10 x 10 that is filled with the character representing an empty stitch. The empty stitch character in the prototype is 'u'.

It inherits from the class Metadata which contains both a UUID representing the pattern id and the name the user gives the pattern. When storing on disk Pattern POJOs are converted to JSON using Gson and saved with the id as filename. The metadata from the pattern is then added to an ArrayList of Metadatas which is saved in the same local app directory and acts as an index of all patterns saved on the device. Using an index file increases performance, since to get a list of the existing patterns not all pattern files have to be loaded, but just the one file containing the ArrayList of Metadatas. Individual pattern can then be loaded using the id saved in the patterns Metadata.

6.5 Keyboard

The row and the grid editor use different keyboards each. The keyboard of the grid editor features stitch symbols and a delete button. The symbols and the delete button can be toggled to be in the active state. While a symbol is active, touching the grid will lead to the active symbol being added to the pattern at the touched cell. Since empty cells contain the designated empty character '.', deletion works in the same way as writing. The keyboard is implemented using a Gridview, a default Android component to display a collection of items in a grid with equal spacing between the items, that also features scrolling. A grid item consists of text set on a KnittingFontButton, a custom view that I defined that extends Android's own Button view. The custom button sets the custom knitting font defined for the project.

In the row editor there are three sections of the keyboard. One for the symbols, one for numbers and one that includes the enter and backspace buttons. Pressing a symbol and number button will append the character or the number to the editor. The enter and backspace button call the system's enter and backspace key events from Android's software keyboard. The symbols sections is also a Gridview, although with less columns than in the grid editor. For the numpad I use the Viewgroup class CalculatorPadLayout form Rahul Parsani. The viewgroup takes a number of child views, in this case KnittingFontButtons, as well as arguments for row and column count. It then calculates the size of the child views so, that they are all equal in size and distribution. There are multiple reasons for using a custom Viewgroup here instead of a Gridview. A Gridview uses an Adapter class to populate the items in it with a collection and handle the onClick event of these items. It supports changes in the collection and offers methods to notify the adapter to redraw the grid after data changes. Writing such an adapter class is unnecessary since the number buttons in the numpad will not change. Each button also calls the same method, an individual handling of onClick events is therefore superfluous. Additionally, a Gridview always makes its items scrollable by default, something that we don't want with a numpad even on different screen sizes. A LinearLayout, a Viewgroup that displays its children one after another, either vertically, or horizontally, could be used there too, but would increase the number of layouts that need to be rendered on the screen. That is generally something to avoid, since views are quite resource intensive in Android and can cause lag in the app. Lastly, the enter and backspace buttons are displayed in a single LinearLayout.

All buttons use a drawable for their background to indicate pressed and clicked states with a filled circle round the pressed button.

6.6 Row Counter

counts rows in a pattern buttons for increase and decrease row counter can be reset with overflow action button ideally: current row should be highlighted in pattern in grid and row format + upon current row change current row should be scrolled into center of screen if not in visible area only works in grid view row format still has issues with knowing how high it is upon instantiation see row format setion

6.7 Grid pattern format: GridEditorView

For displaying the grid format I first looked into Android's own grid implementations, the Gridview and the TableLayout. The TableLayout is similar to HTML's table tag and is displays elements in rows and columns. It should only be used if the underlying views or data will not change. A Gridview, as used for the symbol keyboard, seems an optimal solution for a grid. Unfortunately it is not possible to implement frozen cells, so that the first row and column stick to the edges of the screen to display the row and column numbers.

I then took inspiration from the app Knitting Pattern Maker, introduced in the background section, and Google's developer guide to dragging and scaling, for which they published a sample project implementing those features on an OpenGL canvas that included axis labels.

For the prototype I implemented my own subclass of View. In it I draw the canvas with the line and row numbers, the lines for the grid and the stitch symbols. For the symbols I use the canvas' drawText() method, which takes a Paint parameter that has the custom knitting font set. The position of touched cells is calculated from the pixel input of the touch event. The finger pinch zoom is implemented using Android's ScaleGestureListener interface. The onScale method takes the a ScaleDetector as parameter which contains the coordinates relevant to the scaling and the scale factor. The scale factor from the event is then saved and used when drawing the canvas in onDraw to scale the drawing operations. The SimpleGestureListener interface is used for the two-dimensional dragging of the grid. The dragged distances are clamped in the on-

Scroll method and then applied as translation during the drawing of the canvas. The GridEditorView also has a method to scroll the current line to the center and draw a highlight on the current row in the onDraw method which are used when the grid is displayed in the viewer.

6.8 Row pattern format: RowEditorLinearLayout

The row editor consists of a the RowLinearLayout class, a custom LinearLayout. This layout implements two-dimensional scrolling which uses a native Scroller. The scroll distance is calculated inside the onTouchEvent method and then scrolled by the scroller. It houses one custom EditTexts, which comes with text operations such as selecting, cut, copy, and pasting and a custom TextView. The layout is set programmatically to extend beyond the visible screen and make itself scrollable when the children are bigger than the visible area.

The LinedEditorEditText extends EditText to draw the background for the odd and even lines and to set the knitting font, as well as the highlight for the current row when in the viewer. On the left of the EditText is the LineNumberTextView, a custom implementation of Android's TextView that contains the line numbers. When a new line is added to the editor, the line numbers get a line added as well.

6.9 Editor Activity

This activity simply contains a FrameLayout to programmatically add the grid and row editor fragments to is and allow easy switching between the visible fragments. The menu contains buttons for switching the editors, saving, the overflow menu, which are the three vertical dots, and, when the grid is visible, to set the grid size. Buttons for deleting and renaming the pattern are located in the overflow menu.

Before the views are switched the activity calls the savePattern method. This method checks whether the pattern has been edited and then saves it to local storage. When the user tries exits the activity with the software back button or the home button it is checked again for changes. Are any found then the activity will display a dialog to ask the user if he wants to save the changes before exiting.

6.9.1 Editor Fragments

There are two fragments in the Editor Activity: each of the two editor formats has its own fragment that displays the corresponding keyboard. The fragments handle saving, loading, and updating of the pattern data as well as the keyboard presses. The grid format fragment also displays the dialog for setting a new grid size.

6.10 Viewer Activity

The Viewer activity houses the row counter a placeholder for the grid and row fragments. The counter consists of a simple TextView and Buttons, which show the current line number and increase or decrease the counter. In the activity's menu top the user can also scroll to the current line, a feature that only the grid implements so far, and find a button to reset the counter in the overflow menu to the right.

The fragments are set during runtime programmatically and can be switched with a menu button. Both fragments indicate the current row number with a highlight on the row.

Chapter 7

User Test

After the implementation of a working prototype in accordance with the requirements set at the beginning of this thesis, the interview participants were invited to test the prototype. One participant had no experience nor interest in using knitting pattern charts and was therefore not included in the prototype test. The participants were asked to execute tasks derived from the requirements inside the prototype app during which their reactions were observed, with the main goal of determining the usability of the two pattern chart formats. After the completion of the tasks the user's feedback concerning the prototype was discussed. Users were asked to to note wether the prototype met the expections they expressed during the interview and were it failed to do so. The users were also asked to judge the overall usability — the ease of use of the features, as well as the unambiguousness of the user interface. All user tests were held separately without communication between the participants. The results of these user tests are summarized in the following paragraphs.

Both participants were able to create and name a new pattern without problems. When first confronted with the default pattern in the row editor both expressed initial confusion about the shortened row format. After a brief explanation on what the number and symbol combination signified in the row format and how it would be displayed expanded when viewed in the grid format both participants were quickly able to work with the row editor and accurately produce results they were aiming for. For this they at first relied on switching to the grid editor to see how their changes in the row editor would play out — they were only able to grasp entirely how the row editor functioned after seeing the changed they did to the pattern in the expaned grid format. The participants

did not expect the editor to support the standard editor funtionalities such as selecting, copying, cutting and pasting text. Both participants had problems when asked for the first time to edit the pattern in the grid editor: they expected the same typing behaviour from the symbol keyboard that they encountered in the row editor. It took a few tries and an explanation of the toggling mechanics to enable them to successfully use the grid editor. In both tests the participants found and used the buttons to switch editors and save the pattern without problems. The same holds true for the menu entries to rename and delete the pattern currently open in the editor. At the time of testing viewing the pattern in row format still had some bugs: larger patterns were not scrollable and the current row would not be highlighted upon increase or descrease of the row counter. The testing of the row viewer will there be disregarded for this user test iteration. All functions of the row counter and the grid viewer were used and understood by both participants from the start.

Following the instructed testing the participants were asked to express their feedback concerning the prototype. Both participants expressed the desire to see a tutorial or introduction to the formats upon first use of the app since it was not clear from the beginning that there were two formats available to present a pattern chart. After being confronted with the row format on opening a pattern in the editor, the participants expected the grid editor kyeboard to behave like the one found in the row editor. They had problems understanding how to use the toggle symbols in the grid editor keyboard and were confused why nothing happend when they touched the grid. For the wish to have a symbol pre-selected was voiced, to indicate that symbols have to be selected to be set on the grid and to help the user understand that touching a cell leads to the selected symbol being set to that cell. One user also mentioned that the trash can icon for the button designed to erase a cell was misleading, they expected the button to delete the whole pattern — an eraser icon would be better suited. Another problem was the missing background behind the line numbers in the grid editor, when the grid was scrolled it was hard to read the numbers. To improve this a solid background should be added to the line number sections.

The row viewer did not fulfill the participants' expectation at all — both agreed that at the very least the pattern needs to be scrollable and the current row should be indicated. The wish to set the current row in the viewer by tapping the current row number in the counter section was also expressed.

The participants agreed that the protoype met the expectations set by the preliminary interviews, excepting the row viewer. They compared the process of creating, editing and viewing a pattern chart to their current methods, modifying a spreadsheet to take the form of a knitting pattern chart, and found the prototype to be much easier and efficient to use. They deemed the ability to edit and view a pattern in two formats very valuable, since same stitch repeptitions could be quickly entered in the row format without the hassle of entering every stitch individually in the grid editor. The grid editor offered the ability to easily view the whole pattern and to spot and correct mistakes in it, as well as input more varied stitches in a pattern. Both participants preferred the prototype to their current pattern editors and viewers. One participant expressed the wish for a prototype supporting the same functions with colors instead of stitch symbols for the creation and viewing of colored pattern charts.

Chapter 8

Evaluation and Discussion

requirements reached: yes, except row viewer

- as is state:
- patter file can be imported
- pattern file can be exported to set directory from viewer and editor
- pattern can be renamed from inside editor
- export directory cannot be set by user
- all patterns can be exported for backup purposes
- pattern can be created, edited, viewed, saved, deleted
- row counter exists and is connected to pattern and persisted
- current row in viewer is highlighted in grid viewer

known bugs:

- pattern title has no limit
- imported files aren't checked for tile type, cause crash
- row format is a mess
- pattern title not visible: too many action buttons in actionbar

- should start with grid format instead of row

Issues:

- Edittext not intended for 2D scrolling -¿ solutions: add custom scroller to edittext or place edittext in scrollable layout or add custom scroller to layout and place edittext in it
- Canvas in grid lags: onDraw naive implementation -¿ change implementation to only draw what is visible
- Edittext opens softkeyboard: suppressing is awkward and not intended -¿ used edittext lib that suppresses behavior programmatically as best as posibble, declare soft keyboard state in manifest, then admit defeat
- Row editor doesn't scale -¿ implement custom on scale gesture listener and scale edittext and textview and container layout accordingly
- Disk operations do not block UI -¿ add spinner to indicate disk operations, are very fast tho
- Edittext doesn't read line number correctly -¿ figure out lifecycle of edittext and read line numbers at appropriate time
- Edittext sets width incorrectly -¿ edittext is not intended to support multiline text that extends beyound screen. wrestle with edittext implementation to try to solve this?
- Orientation change not supported yet -¿ support is with persistent fragfment state = fragments wont be recreated on orientation change
- Different screen sizes not supported -¿ create xml files for different device sizes

edittext issues:

- could write own editor, but did not because time was short and implementing own solution for selecting text, cutting, cpoying, deletion, etc typical text editor functionalities was deemed too much effort when there already is a working text editor view which is standard for text editing on android.

- The surprise shown by the participants of the user test may indicate that the standard text editor functionalities — selecting, cutting, copying and pasting text — are helpful features, but not necessary in a knitting pattern row editor. This would allow for a custom built editor better suited for the horizontally scrollable multi-line text used in the row editor that foregoes those functions.

Chapter 9

Outlook

Outlook:

- Ligatures and multi character stitches
- Pictures
- Panning in grid editor
- Support of bigger dimension with canvas optimization
- Focal point when scaling
- built in functions for sharing patterns: email, dropbox, etc
- repeat counter next to row counter for repeated patterns with highlighting functionality for repeats

hello glossary should be here):

List of Figures

Listings

Bibliography

Android Developers. *Action Bar*. URL: https://developer.android.com/design/patterns/actionbar.html (visited on 08/09/2016).

— *Activities*. URL: https://developer.android.com/guide/components/activities.html (visited on 08/09/2016).

— *Android, the world's most popular mobile platform*. URL: https://developer.android.com/about/android.html (visited on 08/09/2016).

— *Animating a Scroll Gesture*. URL: https://developer.android.com/training/gestures/scroll.html#term (visited on 08/09/2016).

— *Choose Internal or External Storage*. URL: https://developer.android.com/training/basics/data-storage/files.html#InternalVsExternalStorage (visited on 08/09/2016).

— *Creating event callbacks to the activity*. URL: https://developer.android.com/guide/components/fragments.html#EventCallbacks (visited on 08/09/2016).

— *Fragments*. URL: https://developer.android.com/guide/components/fragments.html (visited on 08/09/2016).

— *Getting Started*. URL: https://developer.android.com/training/index.html (visited on 08/09/2016).

— *Introduction to Android*. URL: https://developer.android.com/guide/index.html (visited on 08/09/2016).

— *Meet Android Studio*. URL: https://developer.android.com/studio/intro/index.html (visited on 08/09/2016).

— *Package Index*. URL: https://developer.android.com/reference/packages.html (visited on 08/09/2016).

— *System Permissions*. URL: https://developer.android.com/guide/topics/security/permissions.html#normal-dangerous (visited on 08/09/2016).

Android Developers. *Table*. URL: https://developer.android.com/guide/topics/ui/layout/grid.html (visited on 08/09/2016).

— *View*. URL: https://developer.android.com/reference/android/view/View.html (visited on 08/09/2016).

Association, Japan Knitting Certificate. *JIS L 0201-1995: Letter symbols for knitting stitch. Standard booklet*. Nov. 1995. URL: http://www.webstore.jsa.or.jp/webstore/Com/FlowControl.jsp?lang=en&bunsyoId=JIS+L+0201%3A1995&dantaiCd=JIS&status=1&pageNo=0 (visited on 04/04/2016).

Microsoft Corporation. *Recommendations for OpenType Fonts*. May 1, 2014. URL: https://www.microsoft.com/typography/otspec/recom.htm (visited on 08/09/2016).

Raz, Samuel. *Flat Knitting Technology*. Heidenheim: Druck Repo Verlag, 1993.