



SILESIA UNIVERSITY OF TECHNOLOGY
FACULTY OF AUTOMATIC CONTROL, ELECTRONICS
AND COMPUTER SCIENCE

Engineer thesis

Development of accessible mobile sudoku

author: Adam Gajewski

supervisor: Krzysztof Dobosz, PhD

Gliwice, January 2019

Oświadczenie

Wyrażam zgodę / Nie wyrażam zgody* na udostępnienie mojej pracy dyplomowej / rozprawy doktorskiej*.

Gliwice, dnia 9 stycznia 2019

.....
(podpis)

.....
(poświadczenie wiarygodności
podpisu przez Dziekanat)

* podkreślić właściwe

Oświadczenie promotora

Oświadczam, że praca „Development of accessible mobile sudoku” spełnia wymagania formalne pracy dyplomowej inżynierskiej.

Gliwice, dnia 9 stycznia 2019

.....

(podpis promotora)

Contents

1	Introduction	1
1.1	Introduction into the problem domain	1
1.2	Settling of the problem in the domain	2
1.3	Objective of the thesis	2
1.4	Short description of chapters	2
2	Problem analysis	5
2.1	User characteristics	5
2.2	Description of existing solutions	7
3	Requirements and tools	11
3.1	Functional requirements	11
3.2	Non-functional requirements	12
3.3	Description of tools	13
4	External specification	15
4.1	Hardware and software requirements	15
4.2	Types of users	15
4.3	User manual	16
4.4	Working scenarios with screenshots or output files	16
5	Internal specification	19
5.1	Class architecture	19
5.2	Resume of important algorithms	19
5.3	Details of implementation of selected parts	21

6	Verification and validation	23
6.1	Testing paradigm	23
6.2	Test cases	23
6.3	Detected and fixed bugs	23
7	Conclusions	25
7.1	Achieved results	25
7.2	Path of further development	25
7.3	Encountered difficulties and problems	27

Chapter 1

Introduction

1.1 Introduction into the problem domain

Accessibility is a feature that lets blind people to use smartphones. Even if an application was not designed for blind people, they mostly still can use it thanks to screen readers like TalkBack (an accessibility service for Android) reading out loud the content of app to the user. It is a small help compere to many problems related to being blind, but we can increase this help, by creating applications that are more accessible friendly.

Due to the fact that in order to know what is the content of something that is currently displayed on the screen, a specific part of the screen must be tapped, the application should not use the same event as the chose action. In TalkBack this event is overwritten by double tap, so after first tap a button (for example) TalkBack tells the user what was just tapped. After a double tap the application calls the same action as if this element was tapped once with TalkBack turned off. Using TalkBack changes this and many other behaviors of applications, which can make many applications unable to work properly. To change this back to how the application worked originally without TalkBack, the application have to be designed either disable some of TalkBack features for parts of itself, or overwrite its reactions to touch events.

The easiest accessible thing to implement is some text to read. To any other elements should be provided appropriate labels. Thanks to that most of visual

features can be forgotten and the user can proceed to use the app. One of problems of this solution is time necessary for the audio to be played. Because of that application requiring faster response are a lot harder to be accessible, so there are not many accessible games on smartphones.

1.2 Settling of the problem in the domain

Making an accessible game we can take one of two approaches: Either we make the UI simple and decrease number of elements on screen to minimum, or we take the risk of making something more complicated. In the first case we just have to think of an idea that does not require from the user to chose from many options or to react very fast. It leaves us with really lot of options: Quizzes, text RPG's, crosswords, and so on. Simple design does not make a game boring, everything depends on the execution. In the second case we must put trust in the user and make a bit more complicated UI with more elements. When the user will get used to the game this difference in difficulty should decrease, but the time for learning how to play will be usually longer, than in the first case.

1.3 Objective of the thesis

The main objective is to create a working accessible Sudoku game that can be solved both by a blind person and someone fully sighted. The solution should be compatible with support software used by visually disabled users.

Communication with any external systems is not in the planning.

1.4 Short description of chapters



- Introduction - Short description of accessible solutions domain, and what the thesis is going to be about.
- Problem analysis - Problems in the game development accessible for people with visual disabilities, and description of similar solutions of mobile Sudoku applications.

- Requirements and tools - Requirements concerning behavior of the application, its main features, and compatibility with system and tools.
- External specification - Usage and requirements of the application.
- Internal specification - Description of the program and its structure. Analysis of the code and components.
- Verification and validation - Testing process and results of the performed tests.
- Conclusions - Conclusions and final results of the project.

Chapter 2

Problem analysis

2.1 User characteristics

There are a many different visual conditions that could limit a person trying to play a game. The primary conditions encountered in gaming are limitations in vision are:

- Blindness - usually defined as “the loss of vision, not correctable with lenses”
• People who are completely blind cannot play games that rely on visual cues to prompt a player. They must rely on sounds or special hardware such as force feedback to indicate when they need to act.
- Low Vision:
 - Low vision is related to blindness. A person with low vision can detect light, perhaps see some motion, but is very limited as to what they can see. A more detailed definition of low vision is “a visual acuity of 20/70 or worse in the better eye using a best-corrected spectacle correction, or visual fields of twenty degrees or fewer.
 - However, a more functional definition is that low vision comprises any vision loss that adversely affects the performance of daily activities.” A related term is “legally blind”. “Legal blindness is defined as visual acuity of 20/200 or worse in the better eye with correction, or visual fields of twenty degrees or fewer in the better eye“.

- In either of these cases, users could play games provided there was some degree of magnification of the screen. Users could respond to visual and sound cues, but their ability to see a wide area of the game may be restricted by the magnification of the screen.
- Color Blindness:
 - Color blindness is an inability to detect certain colors. It ranges from total color blindness, where the person perceives the world as shades of gray, to more common types where a person does not perceive the differences between red and green or yellow and blue correctly.
 - This condition is more common among men than women. Studies have shown that from 5 to 8% of all men and about 0.5% of women have some degree of color blindness.
 - The effects of color blindness are more pronounced under certain game color schemes. An example from the past was the Firaxis game “Sid Meier’s Alpha Centauri”. The primary color scheme was mostly red and green, which caused problems for some players. This was quickly addressed through the release of a patch, which provided alternate art for the game. Figure 2.1 Example of bad combination of colors.



Figure 2.1: Sid Meier’s Alpha Centauri - Example of bad combination of colors

Included informations from: [accessible.games\[3\]](https://accessible.games/).

The most popular advices in making accessible games for people with visual conditions are:

- Keep your color palette limited to 2 or 3 colors.
- Avoid using bad color combinations.
- Carefully select any contrasting colors and shades.
- Don't only rely on color to convey a message.

The smartphone accessible games have been available for a few years, and the conventional ones were more common for even longer. Puzzles like Rubik cube with braille alphabet on it can be found easily, and there exists a similar solution for the Sudoku puzzle, with numbers represented by small blocks with braille number on it, that can be putted into the puzzle. That solution can not be used in an electronic device not having any form of system showing braille alphabet. There exists a solution on the desktop computer using keyboard and audio reading content of the sudoku, but agin nowadays smartphones usully do not have keyboaeds.

2.2 Description of existing solutions

Durring working on this thesis there heve not been found any accessible sudoku applications. Figures 2.2 and 2.3 are presenting classical Sudoku applications.

Most of the tested applications could be used with TalkBack turned on, because all the game functions became disabled, and even if those functions would work as they were without TalkBack, a person with high visual disability could not use it, because TalkBack could not read the content of the application, and there was not any interface implemented instead to tell the user the content of the Sudoku.



The only Sudoku application found during work on this thesis was a simplified version of sudoku shown on figure 2.4. Despite the fact it is not an application designed for visually disabled people it still can be used by them.

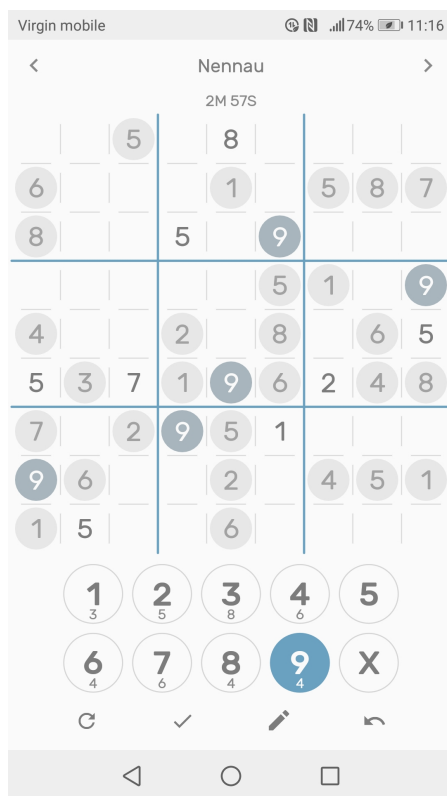


Figure 2.2: Sudoku app1

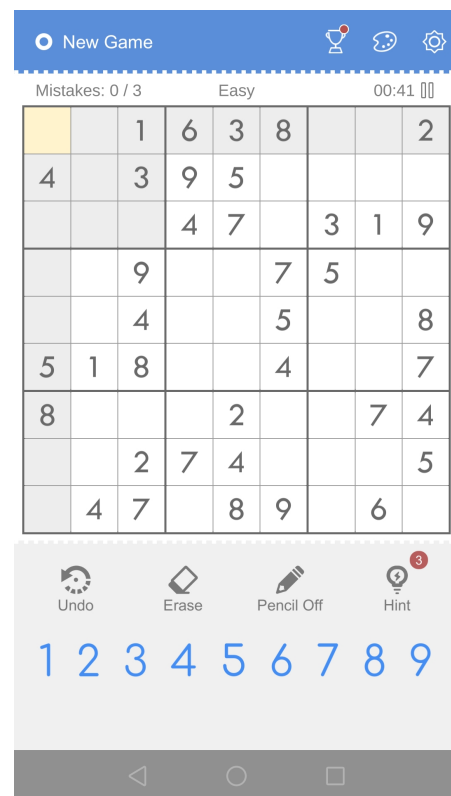


Figure 2.3: Sudoku app2

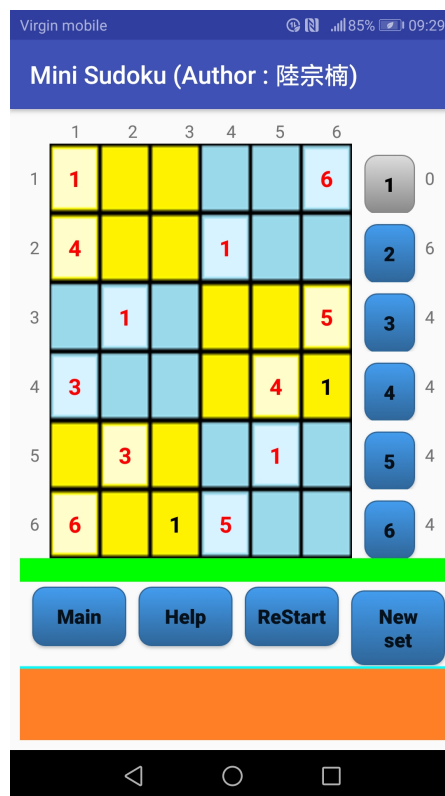


Figure 2.4: Small Sudoku app

Chapter 3

Requirements and tools

3.1 Functional requirements

There have been chosen to use both of previously mentioned approaches in the game, by creating a tutorial level with Sudoku 6 by 6 instead of 9 by 9. This way number of elements is decreased by over 55%, so there are supposed to be two algorithms implemented: one for 6 on 6 and one for 9 on 9. Beside that the application will also need a validation to see if the Sudoku was solved correctly. The use case diagram is shown in 3.1

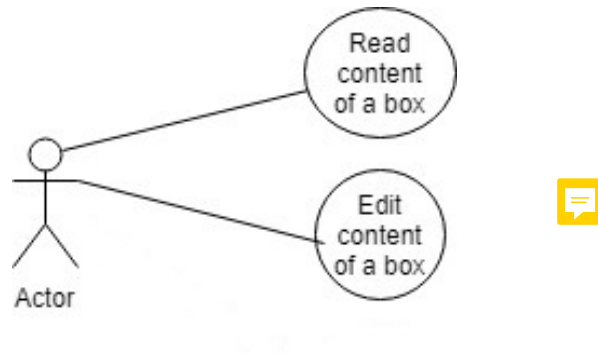


Figure 3.1: Use case

Usage scenario:

1. Instruction on how to play is played to the user (he can skip it if he wants to).
2. User chose size of the Sudoku.
3. User fills in the Sudoku.
4. The program validates if the Sudoku is correct.

3.2 Non-functional requirements

The user interface of the application should work similar to the way that Talk-Back is working:

- tapping an element of Sudoku (number) should result in the interface reading the label to the user;
- to edit an element, user must hold finger on it for a longer period of time, then the content of the element will be changed to the number that the element was tapped;
- in both cases, when the user try to edit a starting element (element that the game have started with), or when the user taps an empty element, a short message will be played to inform about it.

To make it easier for blind user to find number on the screen of smartphone, and be aware of its localization,

- it should be possible to zoom in to one cell with only 6 or 9 elements (depending on the type of Sudoku);
- it would be possible to move between cells by swipe events, and another after zooming out the swipe event can be used to go back to the menu.

Other requirements:

- turning on the TalkBack should not interfere with usage of the application;

- taking into account that this application can be used also by partially blind people, there can be made a facilitation for them, by displaying white numbers on black background. With this contrast they will be easier to see.

3.3 Description of tools

The chosen design tool is Android Studio with Java as the chosen programming language. Android Studio have been chosen as the most advanced tool in the field of creating android applications at that time. It is also equipped and compatible with many other useful tools, like emulators that can be used for testing, or plugins generating diagrams. Java was chosen due to the popularity of this programming language, which ensures wither range of existing solutions of many problems.

The TTS (Text to speech engine) refers to the ability of device to read text aloud. A TTS Engine converts written text to a phonemic representation, then converts the phonemic representation to wave forms that can be output as sound. TTS engines can be used with different languages, dialects and specialized vocabularies through third-party publishers. Android is fully capable of using them[2].

According to Tiobe, Java has been the number 1 or 2 most popular language basically since its creation in the mid-90's. The popularity of Java helps to ensure its future popularity, thanks to a huge community of users[1].



Chapter 4

External specification

4.1 Hardware and software requirements

The application requires a device with API 15 or higher version of android. Because the whole program uses only basic android functionalitis, there is no point in setting any higher version, this way the application will be available on more devices.

The installation is based on installing the APK file to the device and running the installation procedure appropriate to this device (agreeing on installing applications from unknown source could be needed).

4.2 Types of users

This application have been designed having in mind three groups of users:

- Users with no visual disabilities.
- Partially blind users.
- Completely blind users.

This application has only one universal mode for all types of users. The Sudoku will still have simple visual elements, so the game could be played relying only on the eyesight. The content is read to the user that lets play the game without

looking at the screen. And the white on black background contrast could help the partially blind players. This way the application is made as much accessible as it is possible.

4.3 User manual

All the basic instructions are shown and played every time after starting the application, and can be skipped with tapping the screen.

Editing the content of Sudoku number is enabled by a long tap, after long tapping the number it can be incremented by tapping it, after reaching the limit the number goes back to zero. Moving between cells to access more numbers is done by swiping a finger on the screen.

When the TalkBack is on, the swipe must be done with two fingers and the long tap have to be preceded with one normal tap as a double tap. All the other functions of the game are the same as with TalkBack off.

At the beginning user is given a simple instruction on how to play the game:

Welcome to blind Sudoku!
Fill in the boxes so every number will be placed only once in every row, column, and cell. Swipe your fingers to navigate between the cells, and tap a box to hear its content. Long tap the box to enable editing the number, and tap the box to increment the number. Double tap to continue. Have fun!

The way the user interface works was intended to be as similar as possible to the way TalkBack works. So user used to the TalkBack behavior would not have many problems with using it.

4.4 Working scenarios with screenshots or output files

This application uses two views:

- Instructions (Figure 4.1) - short explanation on how to play the game and use the app. It shows up when the application starts, and can be skipped to go to the game.
- Game view (Figure 4.2) - the cell filled with numbers from sudoku where zeros are the empty places to be filled in. It starts in the left upper cell of the whole Sudoku matrix.

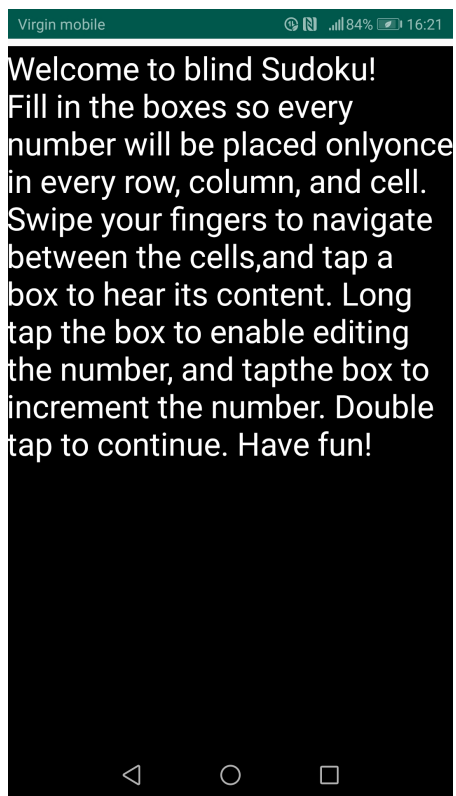


Figure 4.1: Instructions

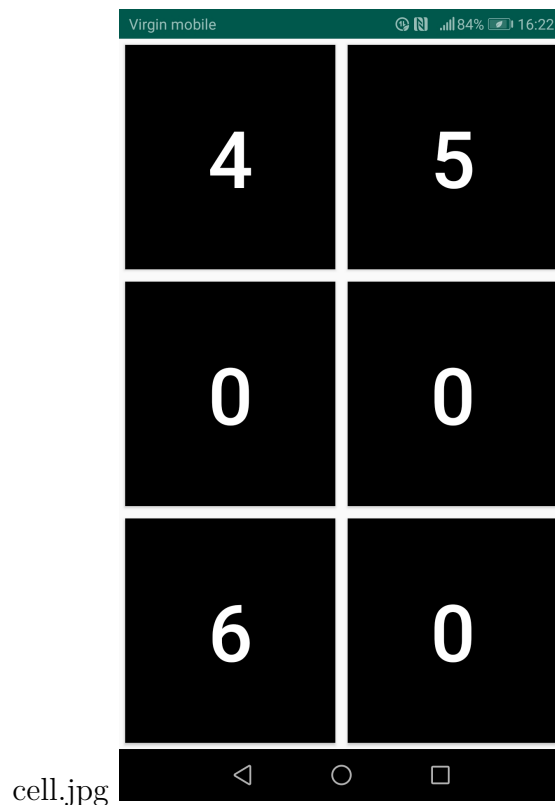


Figure 4.2: Sudoku cell

Chapter 5

Internal specification

5.1 Class architecture

The whole program is based mostly on two classes. The sudoku data processing in class `Sudoku` made of: generating `Sudoku`, validating the `Sudoku`, and all the other methods needed to work with the `Sudoku`. The `MainActivity` class responsible for using the `Sudoku` class and connecting it to the other elements of the game.

Instead of creating additional activity for the view with instructions (that only would handle one action), application uses only the `MainActivity` changing the XML file it uses as the current view.

Figure 5.1 UML of the project.

The application is based mostly on 3 classes:

- `MainActivity` - the activity of this application.
- `SmallSudoku` - matrix with `Sudoku` data, and useful methods.
- `SmallTemplate` - template used to determine which numbers are removed from the `Sudoku` matrix at the beginning of game.

5.2 Resume of important algorithms

Generation of the 6 by 6 `Sudoku` was problematic, due to the fact that a square cannot be made of six square elements (what is required by most algorithms). This

is why this program uses a simpler algorithm:

1. Generate numbers from one to six in random order, and put them into the first row.
2. Go to the next row.

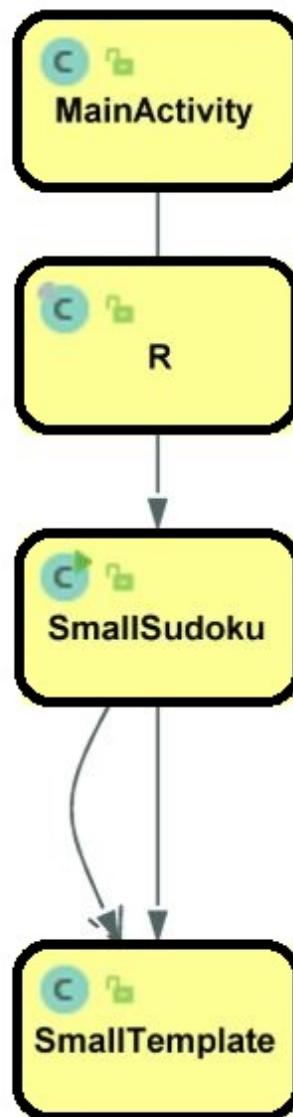


Figure 5.1: UML diagram.

3. Copy the content of the previous row and:
 - (a) If the row number can be divided by 3, shift the numbers of this row by one place.
 - (b) If the row number can not be divided by 3, shift the numbers of this row by two places.
4. If the Sudoku is not full yet, go back to step number two.
5. The Sudoku is ready.

Later on the generated data is copied from one array to another during the copying process the correct indexes of the arrays are compared with a ready made `SmallTemplate` object to create holes for the user to fill in. Letters on both the first and the second array can be compared to see if the user solved the second correctly.

5.3 Details of implementation of selected parts

The game is based mostly on only one view in form of XML file, that is one Sudoku cell, with numbers changing after moving between cells, and editing the content inside them. After every swipe the current cell number is adjusted and the function updating the view, with data from Sudoku, is called.

The application behavior with TalkBack should be similar to the one with TalkBack turned off as much as possible. Firstly the reading of buttons labels by TalkBack was turned off, so it would be done by the implementation in the program. It was done by setting buttons `android:importantForAccessibility` to "no" in the XML file, as seen on figure 5.2.

Secondly the reaction to tapping the buttons had to be overruled, so the user would not have to tap twice more times and to override the previously mentioned reading of labels. It was done by using the `setOnHoverListener` on buttons, and implementing there the same behavior as in the `onClick` method, where we test if the TalkBack is on by using the `AccessibilityManager`.

```
1 <Button
2     android:importantForAccessibility="no"
3     android:id="@+id/button"
4     android:tag="0"
5     android:layout_width="0dip"
6     android:layout_height="match_parent"
7     android:background="#000000"
8     android:text="0"
9     android:textSize="64dp"
10    android:textColor="#FFFFFF"
11    android:layout_margin="5dp"
12    android:layout_weight="1" />
```

Figure 5.2: The `button` element from `MainActivity.xml`




Chapter 6

Verification and validation

6.1 Testing paradigm

Due to the fact that the most crucial features could not be tested by means of unit testing nor integration testing (behavior with TalkBack, audio messages), the only performed testing was acceptance testing.

According to the V-model the testing should include unit testing, integration testing, system testing, and user acceptance testing. But (as written above) due to internal architecture of the application it was possible to perform the last type of testing. 

After implementing each functionality it was tested manually on a smartphone.

6.2 Test cases

During the manual testing there were taken into account some crucial features and their behavior. They are described in the table 6.1 cases in testing.

6.3 Detected and fixed bugs


During the development and the actual testing procedure, there have been found and fixed following bugs: 

Table 6.1: «User actions» Result

Action	User actions
[bp!] Edit empty box	After long tapping the empty box the number editing is enabled, and every tap increments the number. After next long tap the content of box is saved.
Edit box with content generated by application	User is informed that the content cannot be changed.
Single tap a box	Application speaks the number that is inside the box.
Single tap a box when TalkBack is on	Application speaks the number that is inside the box.
Swipe vertically and horisontally	The view moves between cells and stops and detects if it can move further.
Fill in the Sudoku correctly	Application informs that the Sudoku is correct.
Fill in the Sudoku incorrectly	Application informs that the Sudoku is incorrect.

- The upper limit for vertical cell movement was too high, that resulted in index out of bounds exception.
- The application was displaying wrong numbers due to a mistake in translation of button number into x and y coordinates of sudoku.
- Vertical swipe was not always working, because the event was not defined properly.
- Application was not able to compile. The first usage of the text to speech was accidentally before its initialization.

Due to the importance of TalkBack compatibility, every feature had to be tested with and without enabled TalkBack.

Chapter 7

Conclusions

7.1 Achieved results

In the chapter one section Objective of the thesis, and chapter three Functional requirements and Non-functional requirements have been stated objectives and expectations towards the thesis. In table 6.1 is shown comparison between them and achieved results.

The final result of the project is fully functional Sudoku mobile game that can be played without looking at the screen. It can work both with and without the TalkBack turned on, and the accessibility feature would not change. The only difference in playing the game with TalkBack turned on is the way player enables the change of number in button. Instead of just holding the button for a longer period of time, it must be double tapped where the second tap is the one to be held longer.

7.2 Path of further development

At this time the game can be played only on small (six on six) Sudoku with simple algorithm of generation, for better game play results it could be possible to chose size of Sudoku and difficulty level. The application could be also extended by additional features not necessary for the game mechanics like saving time of the play into a ranking to compere results. Possibility to see the ranking should

Table 7.1: «Is it expected result» [

Action	Is it expected result
bp!] There is generated sudoku puzzle	Yes.
Editing box with content generated by application is blocked, but editing empty box is possible	Yes.
Single tapping a box results in application speaking the number that is inside the box	Yes.
Application works the same with TalkBack on and TalkBack off	When the TalkBack is off editing box is enabled by long tapping the box, and with TalkBack on it is enabled by double long tapping the box.
Only one cell is shown at a time and moving between cells is done by swipping the screen.	Yes.
After filling in the whole Sudoku applications checks if it is correct.	Yes.
Sudoku can be solved without looking at the screen	Yes.
User can chose between 6 on 6 and 9 on 9 sizes of sudoku	No.
Colors are in contrast helping partially blind users.	Yes.
The main objectives of the thesis are meet.	

be made in the menu view, that would be also useful to implement together with going back to menu from the game view.

There are also other features that could make the game more accessible. Possibility of showing the whole Sudoku on the screen, by making the zoom out gesture, and zoom in to go back into the single cell. When the whole Sudoku is visible it is easier to look thru rows and columns, but navigating between 36 elements (or even 81 if the nine on nine version was chosen) could be challenging for a blind person, so both zoom in and zoom out views should be available. Similar result can be achieved by implementing hints for the user: Telling him if there is a certain number in a row or column. How many occurrences of a certain numbers missing in the whole Sudoku. It could also read all numbers in a row or column in the same order as they are placed in the Sudoku.

Leaving the subject of the game itself, there is also place for improvement in the displaying the text. The contrast of white text on black background can help people who still have partial sight, but for large variety of eyesight diseases, it can be possible that for some eyesight disease it would be easier to read with different contrast. This is why it should be possible to change the colors of contrast in the menu settings.

7.3 Encountered difficulties and problems

The most difficult part of implementing this application was making the application behavior, with and without TalkBack, as similar as possible. There were many aspects that had to be taken into account, as many normal behaviors of the phone are overwritten when the TalkBack is on.

The second most difficult part of implementing this application was coming up with easy and intuitive user interface, that would not include many inconveniences for visually disabled users.

Bibliography

- [1] BEN PUTANOi. Most popular and influential programming languages of 2018.
<https://stackify.com/popular-programming-languages-2018/1>. [access date: 2019-01-01].
- [2] Jean-Michel Trivi. An introduction to text-to-speech in android.
<http://android-developers.blogspot.com/2009/09/introduction-to-text-to-speech-in.html>. [access date: 2019-01-02].
- [3] Unknown. igda-gasig.org. <https://igda-gasig.org/about-game-accessibility/guidelines/visual/types-and-definitions/>. [access date: 2019-01-06].



Appendices


List of abbreviations and symbols

DNA deoxyribonucleic acid

MVC model–view–controller

N cardinality of data set

μ membership function of a fuzzy set

\mathbb{E} set of edges of a graph 

\mathcal{L} Laplace transformation

Listings

(Put long listings in the appendix.)



Contents of attached CD

The thesis is accompanied by a CD containing:

- thesis (L^AT_EX source files and final pdf file),
- source code of the application,
- test data.

List of Figures

2.1	Sid Meier's Alpha Centauri - Example of bad combination of colors	6
2.2	Sudoku app1	8
2.3	Sudoku app2	8
2.4	Small Sudoku app	9
3.1	Use case	11
4.1	Instructions	17
4.2	Sudoku cell	17
5.1	UML diagram.	20
5.2	The <code>button</code> element from <code>MainActivity.xml</code>	22

List of Tables

6.1	«User actions» Result	24
7.1	«Is it expected result» [.	26