CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF BACHELOR’S THESIS

Title:	Dragon II - plugins I
Student:	Miroslav Mazel
Supervisor:	Ing. Jiří Chludil
Study Programme:	Informatics
Study Branch:	Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

Dragon II is a touch-based learning application for the Android OS, intended mainly for elementary school students with learning disabilities. The application is loosely based on the desktop learning application Dragon, which was the subject of several Bachelor's theses from previous years.

1) Choose 5 problem areas that children with learning disabilities deal with.
2) Analyze strategies for teaching children with learning disabilities related to the chosen problem areas.
3) Analyze current applications and exercises dealing with these problem areas.
4) Design a module for each problem area.
5) Implement these modules.
6) Implement a complementary exercise creation module for each module, if appropriate.
7) Test each module, including usability testing.

References

Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague February 6, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Bachelor's thesis

Dragon II — Plugins I

Miroslav Mazel

Supervisor: Ing. Jiří Chludil

17th May 2016

Acknowledgements

I would like to thank my supervisor Ing. Jiří Chludil for providing guidance and feedback every week, making sure that the thesis was on track, and working hard to make sure that this project has a future. I would also like to thank my reviewer Ing. Radek Richtr for providing feedback on the thesis. Last but not least, I am very grateful to the entire team behind Dragon II. I am lucky to have worked with such a motivated and talented team of people and I thank them all for providing support and inspiration throughout the course of the project.

Declaration

In Prague on 17th May 2016 . . . .

Citation of this thesis

Mazel, Miroslav. *Dragon II — Plugins I.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

V této práci jsem zanalyzoval, navrhl, implementoval a otestoval soubor cvičení pro děti s poruchami učení a editory těchto cvičení. Každé cvičení a editor bylo součástí modulu pro vzdělávací aplikaci Dráček II. Pomocí Android SDK jsem vytvořil pět modulů pro platformu Android: modul, co dyskalkuliky učí jak číst analogové hodiny, modul umožňující doplnit část slova do mezery, modul pro rozlišování krátkých a dlouhých slabik, modul pro cvičení rozkladu čísla na sčítance a modul, co pomáhá pochopit vztah mezi množstvími ve slovní úloze. Tyto moduly pomohou dětem s poruchami učení na Základní škole Smečno a kdekoliv jinde, kde se bude aplikace Dráček II používat. Zdrojový kód a dokumentace modulů jsou součástí přílohy.

Klíčová slova moduly, vzdělávací aplikace, Dráček II, Dráček, hry, cvičení, děti, poruchy učení, dyskalkulie, dyslexie, Android

Abstract

In this work, I analyzed, designed, implemented, and tested a set of exercises for children with learning disabilities as well as complementary exercise editors. These were developed as modules for Dragon II, a learning application for the Android platform. Using the Android SDK, I created five modules:

a module dedicated to teaching dyscalculics how to read an analog clock, a module allowing children to fill in the blank using parts of words determined by the teacher, a module for determining short and long syllables, a module for practicing number bonds, and a module helping understand the relationship between quantities in word problems. These modules will help children with disabilities at the Smečno elementary school and wherever else the Dragon II application is used. The source code and the documentation of these modules are found in the appendix of this work.

Keywords modules, learning application, Dragon II, Dragon, games, exercises, children, learning disabilities, dyscalculia, dyslexia, Android

Contents

Introduction **1**

1 Analysis **3**
1.1 Background 3
1.2 Terminology 3
1.3 Learning Disabilities 4
1.4 Exercises 7
1.5 Requirements 14
1.6 Use Cases 22

2 Design **25**
2.1 Code Style 25
2.2 Android Components 26
2.3 Custom Libraries 27
2.4 Question Stepper 27
2.5 Editor Recycler 31
2.6 Clock Module 31
2.7 Syllable Length Module 33
2.8 Fill-in-the-blank Module 34
2.9 Number Bonds Module 35
2.10 Word Problem Module 36

3 Realization **39**
3.1 Development Environment 39
3.2 Installation Manual 40
3.3 Communication with the Core 40
3.4 Libraries 41
3.5 Modules 41
3.6 Third-Party Libraries 42

4 Testing **45**
4.1 Unit Testing 45
4.2 Integration Testing 45
4.3 Manual Testing 46
4.4 User Testing 46

Conclusion **49**
Goals 49
Future Work 50

Bibliography **51**

A Acronyms **55**

B Contents of enclosed CD **57**

List of Figures

1.1 Včelka's daily "playlist" of tasks 8
1.2 "Hours and Minutes" exercise in NumberSense Time 9
1.3 Syllable module in Dragon I 10
1.4 Včelka's syllable recognition module 10
1.5 Fill-in-the-blank module in Dragon I 11
1.6 *Český jazyk - pravopis* by PMQ Software 11
1.7 Math Playground Thinking Blocks: Ratio and Proportion 13

2.1 Mockup of the Question Stepper 30
2.2 Mockup of the Clock exercise 32
2.3 Mockup of the Syllables exercise 33
2.4 Mockup of the Fill-in-the-blank exercise 34
2.5 Mockup of the Number Bonds exercise 35
2.6 Mockup of the Word Problem exercise 37

3.1 Communication between the different parts of Dragon II 40

List of Tables

1.1 Core library evaluation 17
1.2 File format evaluation 18
1.3 Programming language evaluation 19

2.1 Naming style 26
2.2 Question Stepper flow evaluation 29

Introduction

The gamification of learning is a hot topic today and a growing movement in education. Children with learning disabilities have the most to gain from gamification, which is why I chose to make a set of simple exercises suited to their needs. My work involves designing, implementing, and testing modules for the Android learning application Dragon II.

For each module, I will start with a rundown of current research on the learning disabilities involved and associated teaching methods. I will follow up with an overview of games dealing with similar topics and discuss their strengths and weaknesses. I will then describe the design and how it builds on those games. I will segue into the implementation and then finally summarize the testing done.

This work builds on the theses of two other team members, one working on the core of the application and one on working on its server side. Another team member is working on another set of modules.

Chapter 1

Analysis

In this chapter, I will cover the problem areas that I am targeting along with the disabilities they are associated with, exercises that deal with these problem areas, the technology choices behind the modules, and the requirements for each module.

1.1 Background

Dragon II is a modular learning application for the Android platform, focused on helping children with learning disabilities. Dragon II takes inspiration from Dragon I, a desktop Java application that was the focus of several past bachelor's theses at CTU (Czech Technical University). While Dragon II shares the goals of Dragon I, it is being designed from the ground up to prioritize touch-based input, to approach the problem with a fresh set of eyes, and to match the requirements of the Android platform, which differ from the requirements of desktop Java applications in some key ways.

1.2 Terminology

In the following text, I will repeatedly refer to a few terms that are worth clarifying.

- The **core** of the application is an Android application used for installing, launching, and gathering data from modules. While it doesn't include any parts of gameplay, it provides an overview of tasks and achievements and serves a single point of entry to all aspects of the various modules.

- A **module** is an application focused on a single way of playing. Its content may vary, though. It communicates with the core but is installed separately. Generally, a module will consist of an exercise and an editor.

- An **exercise** is the portion of a module that provides the gameplay for a student.
- An **assignment** refers to the content provided to the game. For a single exercise, there is often a wide range of possible assignments.
- An **editor** is the portion of a module used to create or edit an assignment.

1.3 Learning Disabilities

This section goes over the learning disabilities that the modules target and then goes more in depth to talk about some problem areas that stem from these disabilities. These areas were chosen as the areas that the individual modules should center on.

1.3.1 Overview

Before I get to the chosen problem areas, I need to cover the disabilities in general.

Dyscalculia

Dyscalculia is primarily characterized by problems with basic arithmetic. At a young age, dyscalculics often struggle with conceptualizing numbers and associating numerals with the amounts they represent [23]. To use an analogy, a dyscalculic might perceive the series "one two three four ..." similarly to how a non-dyscalculic might perceive a non-numeric ordered string of words – e.g. "the quick brown fox jumps over a lazy dog". To understand the difficulty that dyscalculics grapple with, a non-dyscalculic can try performing arithmetic on this series of words, without associating them with the words for the actual numbers. For example, subtracting "fox" from "lazy". This is more complicated with words that appear later in the series, as one generally has to count up from one to find the value of a word. As dyscalculics have trouble understanding the meaning behind words for numbers, they may have problems with counting by one (especially when going over a multiple of ten or when counting down instead of up) or by ten. They may struggle with amounts of objects and need to count and sometimes recount them by one, even if there are relatively few. They may be missing a sense of the scale of numbers and have trouble estimating the difference between two numbers. It follows, then, that they also have issues when it comes to arithmetic and other areas of mathematics where the general population does not, regardless of their IQ.

Additionally, dyscalculics often, though not always, struggle with spacial and temporal orientation, due to the same neurobiological deficiencies that limit their sense of numbers. Maps and clocks may be difficult if not impossible to read and estimating distances or the amount of time it takes to do something may be a futile exercise [22].

Dyslexia

Dyslexia refers to difficulty with recognizing words. It is a disorder characterized mainly by difficulty sounding out words and/or difficulty associating letters with sounds [15]. When attempting to read without going through the proper management exercises, a dyslexic might read too slowly, often reading letter by letter and taking the time to decipher each letter[27]. The opposite can be true as well, though — he or she might read too hastily, guessing at words and not making the effort to interpret them properly. Even with a fluid reading technique, a dyslexic might have problems with distilling meaning from text. Spelling might also turn out to be problematic.

While the causes of dyslexia aren't completely clear, it is known that it is neurobiological in nature[26]. For example, when a non-dyslexic reads, his or her brain's left hemisphere, which is associated with language processing, is active for the most part, whereas when a dyslexic reads, most of the associated brain activity happens in the right hemisphere. It is thought that dyslexia is mostly genetic in nature, but more research needs to be done. Several genes, rather than one, have been associated with dyslexia.

Dyslexia tends to stem from a number of deficiencies[27]. One major deficiency is having trouble with phonology, such as not being able to glean which syllable to stress, which words rhyme, or what the syllables of a word are. Another key deficiency is visual in nature, which has to do both with eye movement and with iconic memory, which tends to be unusually long, resulting in letters being superposed in memory. A memory deficit may be at play as well, especially with regard to short-term memory, which holds information for several seconds, and working memory, where information pertaining to a person's current focus is processed and manipulated. Another deficit often encountered is one of automatization, as dyslexics can be slower to automate processes. There are various other deficiencies that may contribute to a person's dyslexia.

Dysgraphia

Dysgraphics have trouble with writing. The writing of a dysgraphic may be illegible or contain inconsistent letterforms (e.g. mixing lowercase and uppercase letters in the middle of a word)[27]. Even when legible, the effort and time a dysgraphic puts into writing greatly exceeds the same for a non-dysgraphic.

Dysgraphics often have trouble with the fine-motor skills required in writing (some may even suffer from tremors), with visualizing letters, and with remembering the movements associated with drawing letters[24]. As a result, dysgraphics may also have trouble with expressing themselves on paper and with staying focused.

1.3.2 Chosen Areas of Focus

After deeper reflection, I chose five problem areas to base my modules on.

Reading an Analog Clock

Even children without disabilities may face difficulty when learning how to read an analog clock. For dyscalculics of all ages, the clock may be quite daunting, which can lead to all kinds of uncomfortable situations in their lives[22].

One point dyscalculics often struggle with is understanding that there are **two different scales** at play on an analog clock — one going from 0 to 59 for minutes and seconds and another from 1 to 12 for hours[2]. In order to understand the time on an analog clock, one has to grasp both of these scales, which is why it can be helpful to teach the scales separately.

Recognizing Syllable Length

Many dyslexics have trouble recognizing the difference between long and short syllables in speech, which then translates into poor writing[27]. It is important to ingrain in students the difference between short and longs syllables. To do this, some teachers exaggerate the difference between long and short syllables in speech and highlight the meaning of the acute accent (čárka). They can also use audio and visual tools (such as underlining parts of words) to stress the difference[7].

Spelling

Spelling can be hard for everyone, but dyslexics and dysgraphics generally have extra difficulty with spelling. This is partly due to the greater cognitive effort involved in writing for them, making it harder to also focus on spelling, and partly a direct result of the cognitive shortcomings they have. While it has been previously thought that a weak visual memory is the main culprit behind increased spelling difficulty for dyslexics, new research is showing that it plays a rather minor part and that difficulties with learning language are behind it[11]. Highlighting the etymology of words and encouraging children to write words by hand to get a feel for them are among the teaching techniques used to teach spelling. Training sound recognition to help children overcome their cognitive deficiencies has also been shown to help[8].

Conceptualizing Numbers

As mentioned above, dyscalculics generally have problems with conceptualizing numbers. It is crucial for dyscalculics to gain a general sense of amount and number placement for at least smaller numbers, as this is the foundation that mathematics is built on. The teaching techniques here are mostly practical and they will be discussed more in depth in the next section.

Word Problems

Word problems are where dyscalculia and dyslexia meet. Dyslexics have trouble parsing the assignment, whereas dyscalculics have trouble working with the values within the problem. It is important, however, that both learn to solve word problems, as they are the foundation of applied mathematics. Teaching strategies generally revolve around trying to model a word problem so that children understand. There is a wide variety of possible models — one may use wooden blocks, plasticine, pictures, diagrams, or anything else that is suitable for the problem at hand.

1.4 Exercises

For each problem area, I will discuss the traditional remedial exercises as well as modern software exercises, which are more direct competitors of the modules being developed, and decide on the most appropriate exercise type for my modules.

1.4.1 Similar Software

Below, I will mention relevant pieces of software for every problem area. As there are pieces of software that also provide a range of modules for dyslexics or dyscalculics, I will provide an overview of them here and then only discuss the relevant modules under each problem area.

Dragon I

As mentioned above, this work is loosely based on the Dragon I Java application, which has a very similar scope. Dragon I is comprised of a core listing the various exercises and a number of modules. Each module has a corresponding editor for making assignments for these exercises. It is up to teachers to make these assignments. Just like Dragon II, Dragon I aims to target learning disabilities in general, so the range of possible modules is quite wide. The modules currently implemented for the application seem to mostly target dyslexia, though there is some overlap with dysgraphia as well.

Včelka

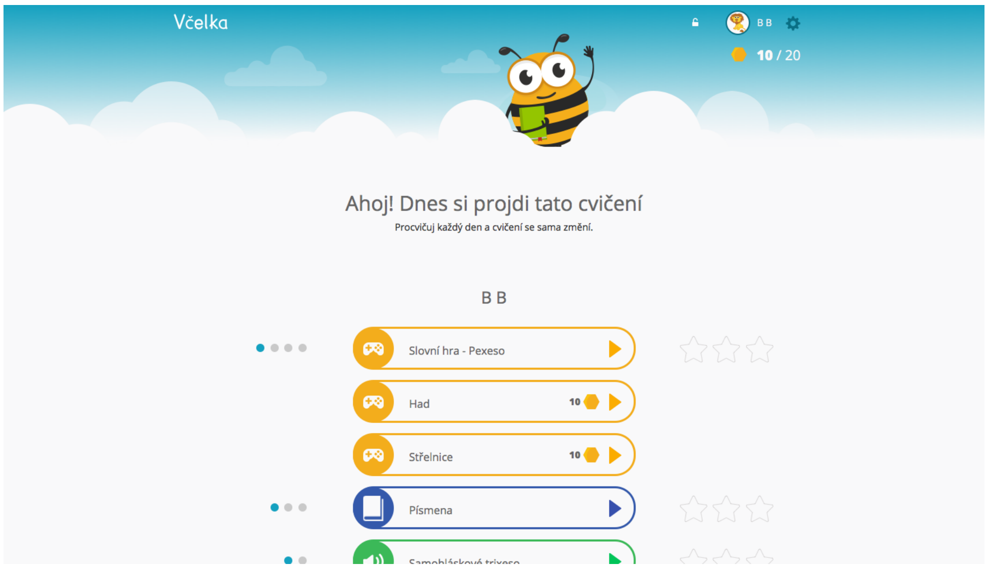

Včelka
B B
10 / 20
Ahoj! Dnes si projdi tato cvičení
Procvičuj každý den a cvičení se sama změní.
B B
Slovní hra - Pexeso
Had
10
Střelnice
10
Písmena

Figure 1.1: Včelka’s daily ”playlist” of tasks[25]

Včelka and its English variant Levebee are both web applications targeted at dyslexic children. They offer a range of exercises, suggesting to the learner which ones and how much to focus on each. Inspired by Duolingo, the app uses game design elements such as laying out the progress made and remaining and rewarding points for answering questions correctly, which the students can then use to ”buy” items in-game. The exercises cover a range of activities, covering pronunciation, listening, writing, reading, and combinations of these, and new types of exercises are being actively developed. Unlike Dragon II, the application is not touch-based and, at the time of this writing, does not have an Android port. Its content is generated by the authors of the application and while there are settings one can tweak for each exercise, there seems to be no way to add custom words.

1.4.2 Reading an Analog Clock

The exercises for reading an analog clock tend to be relatively straightforward, centering around the clock face and hands. Software tools are mostly digital equivalents of tried-and-true classroom tools.

Classroom Tools

Manipulable Clock A clock with manipulable hands is a commonly used tool for both teaching and practicing an analog clock. Letting a person play with the hands helps them get a feel for the hands and for how they move and affect each other. This type of clock allows a teacher to point out the different times on a clock and illustrate the relationships between the time and the positions of the clock hands. To exercise this knowledge, a student is asked to move the clock hands to show a specified time.

Multiple Choice Another method for exercising time is a more generic one: multiple choice. The student can be provided a time and a choice of several images of a clock with hands in different positions and be asked which image matches the time. Another strategy is to provide an image of a clock and give a choice of what time it is showing.

Open Answer Another option is to instead let the student draw clock hands into a picture of a clock. This is basically the same as with a manipulable clock, just suitable for paper use and no direct interaction. The reverse is also possible – showing a picture of a clock and asking for a time.

Software Tools

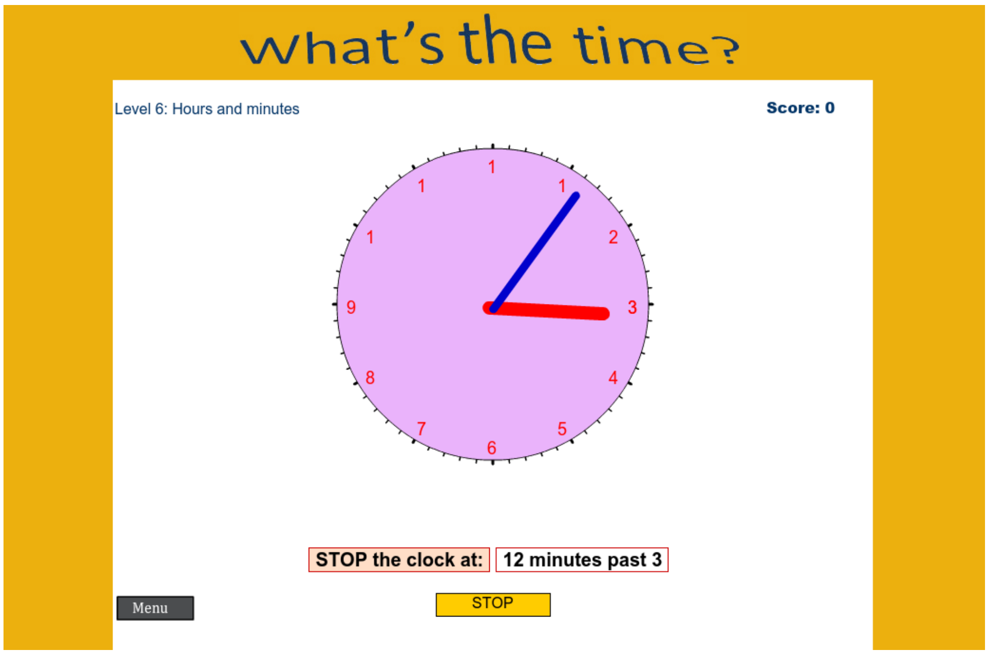


Figure 1.2: "Hours and Minutes" exercise in NumberSense Time[19]

NumberSense Time NumberSense Time is a Flash application designed specifically for dyscalculics. It offers a number of exercises dealing with different aspects of an analog clock — seconds, minutes, hours, and quarter-hour increments are all accounted for. Unfortunately, it contains some bugs, seems to no longer be maintained, Flash prevents it from being used on certain platforms, and it is not touch friendly.

Other Applications There are a range of apps on both Google Play and the iOS AppStore for teaching how to read an analog clock to children. Many of them are split into different sections by topic or by exercise type. A pattern common to many is explaining the two different scales at play. These apps generally stick to the exercises listed above: a clock with manipulable hands, multiple choice or free answer questions.

Evaluation

As multiple choice and free answer questions can be satisfied with more generic modules and as they don't provide direct interaction, thereby being more banal, a **manipulable clock** was chosen as the appropriate content for a module for analog clock exercises.

1.4.3 Recognizing Syllable Length

Syllable length exercises tend to be very similar — they simply ask a child to identify syllable length in one way or another. A rundown of some common exercises follows.

Classroom Tools

Accenting Syllables Children may be asked to accent the syllable lengths in speech. This encourages the child to pay more attention to the sound of the word, recognize syllables, and to notice the differences between their lengths. It is a hands-on approach where feedback is given immediately.

Paper Exercises When interacting directly with the child isn't an option, a child may be given a worksheet asking him or her to highlight, underline, or add symbols to long or short syllables.

Software Tools

Dragon I Module The inspiration for the module comes from a Dragon I module[4], where the student is tasked with dragging words into categories by syllable pattern. Functionally, the exercise portion of the module for Dragon II is the same, while the editor portion differs due to different requirements of the core.

Včelka Včelka offers an exercise where the child first listens to a word and is first asked to indicate how many syllables there were. After they answer correctly, they are asked about the lengths of each syllable. The word is then shown to them.

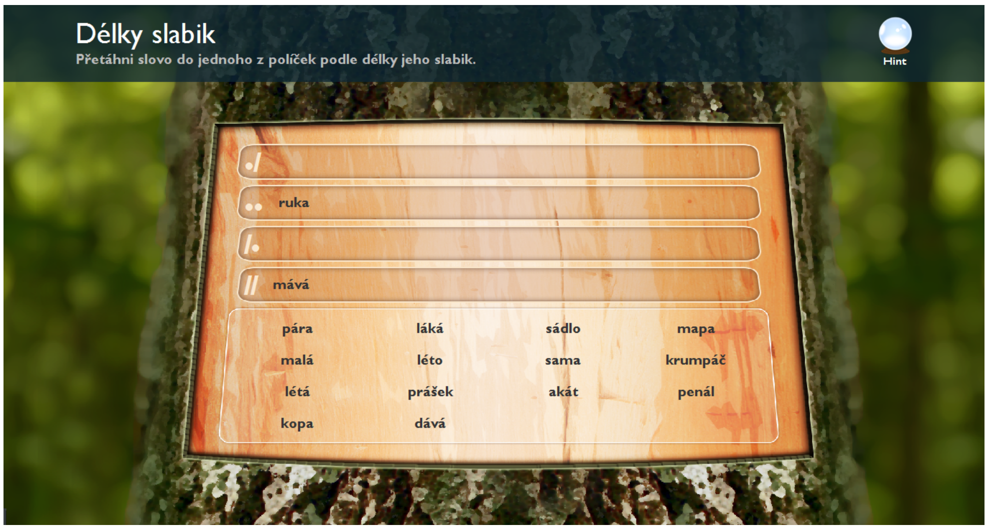


Figure 1.3: Syllable module in Dragon I[6]

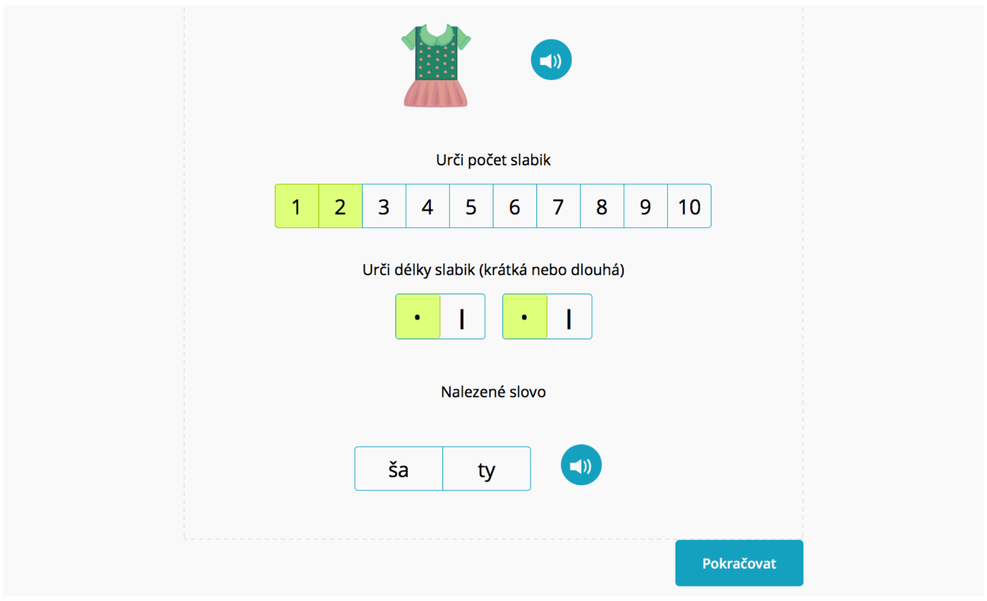


Figure 1.4: Včelka's syllable recognition module[25]

Evaluation

The Dragon I module's approach of grouping words by syllable type was selected, as it's cognitively simpler (there is only one task), shows the relationship between words of a single syllable pattern, and as it is quite suitable for the tablet form factor, where dragging and dropping words can be more entertaining than just tapping buttons.

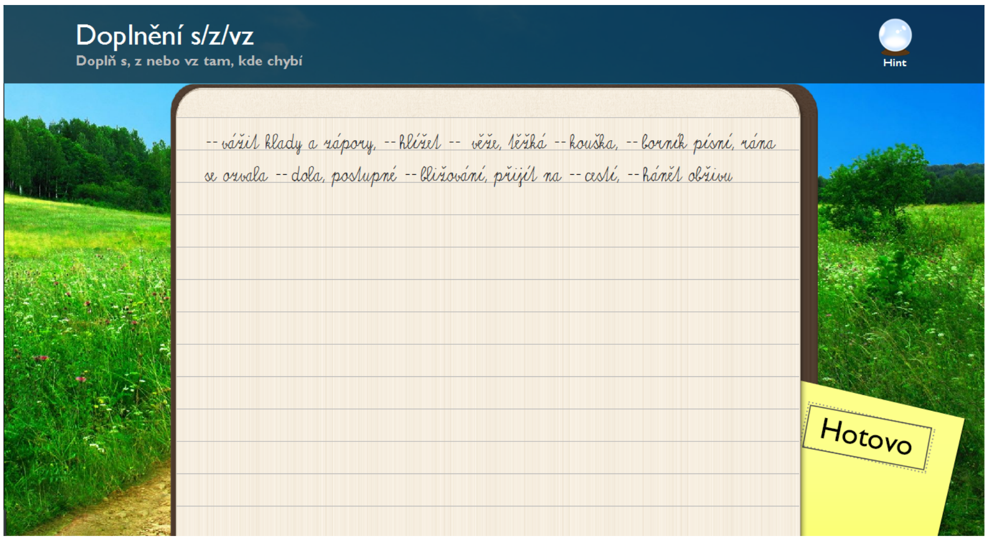
1.4.4 Spelling

Spelling is a problem that requires extra attention for dyslexics, but that children without disabilities need to practice thoroughly as well. The exercises below are relevant for all children, not just dyslexics, even though research was conducted with dyslexia in mind.

1.4.5 Classroom Tools

Writing Words Writing down words is a commonly used strategy for practicing spelling. Writing is a multisensory practice, affecting muscle memory as the student picks up on how to draw the shapes of letters, strengthening the connection between letters and their sounds as the child translates sounds into writing, and committing the appearance of the word into the child's memory[10].

Software Tools



-- vážil klady a zápory, --hlížel -- věže, těžká --kouška, --borník písní, rána se ozvala --dola, postupné --bližování, přijíl na --cestí, --háněl obživu

Figure 1.5: Fill-in-the-blank module in Dragon I[5]

Dragon I Module In this Dragon I module[4], the student is given a longer piece of text with some blanks missing and is tasked with filling in those blanks. Assignments are determined by the teacher, allowing for a broad range of spelling difficulties to be covered.

Český jazyk - pravopis PMQ Software's Android app takes a relatively similar approach, but rather than present all words at once, it presents them one by one for better focus. It also offers a timed variant and lessons related to the practiced topic.

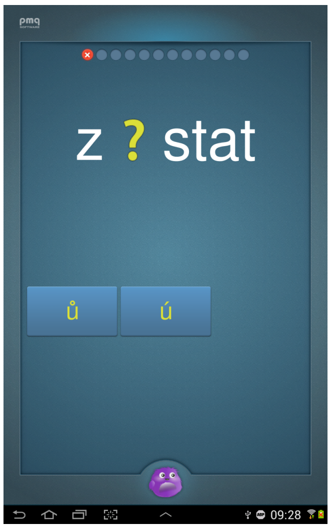


Figure 1.6: *Český jazyk - pravopis* by PMQ Software[21]

Evaluation

As translating spoken word into handwriting would be too complex to use and implement (the teacher would have to record every word as well as write it, and handwriting recognition would be extremely difficult to implement), an exercise based on **filling in the blank** looks to be the most appropriate for a module. For better focus, it's preferable that questions are posed one at a time.

1.4.6 Conceptualizing Numbers

Conceptualizing numbers might require time and dedication, but it is key to dealing with dyscalculia. I go over some ways to instill mental models of numbers in a child below.

Classroom Tools

Bonds A core part of the Singaporean mathematics curriculum, number bonds refer to part-part-whole relationships between three numbers. In a number bond exercise, a student is generally given two numbers (a whole and a part or two parts) and is asked to deduce the third. Number bonds allow students to conceptualize numbers and to practice essential arithmetic skills[3].

Dice Patterns As it is often a problem for dyscalculics to associate meaning with numerals, dice patterns, which they are likely to know from board games and which visually represent a quantity, provide a good alternative representation of numbers[9].

Cuisenaire Rods Cuisenaire rods are blocks that represent numbers up to ten. A small cube represents the number one, a block twice the length represents the number two and so on until ten. These rods, like the dice patterns, are meant as a visual aid for associating values with numbers. They are also very good for learning addition, as they help with splitting the number into tens.

1.4.7 Software Tools

Meister Cody - Talasia

Meister Cody - Talasia is an Android game for young dyscalculics. The game is immersive, putting the player in the middle of a story about a land called Talasia, which the player must save. To do this, the player must solve a range of problems, all of which revolve around number sense in some way. In some portions of the game, boxes of ten cells (laid out in 2 rows by 5 columns) holding dots are used to represent numbers, allowing the child to visually associate an amount with a number.

Evaluation

Number bonds were selected as the exercise, as they have a proven track record of working and translate well to a tablet. The exercise should also provide the option of using dice patterns instead of numerals for better illustration of the quantities they represent.

1.4.8 Word Problems

In the following text, I describe the approaches used for practicing word problems.

Classroom Tools

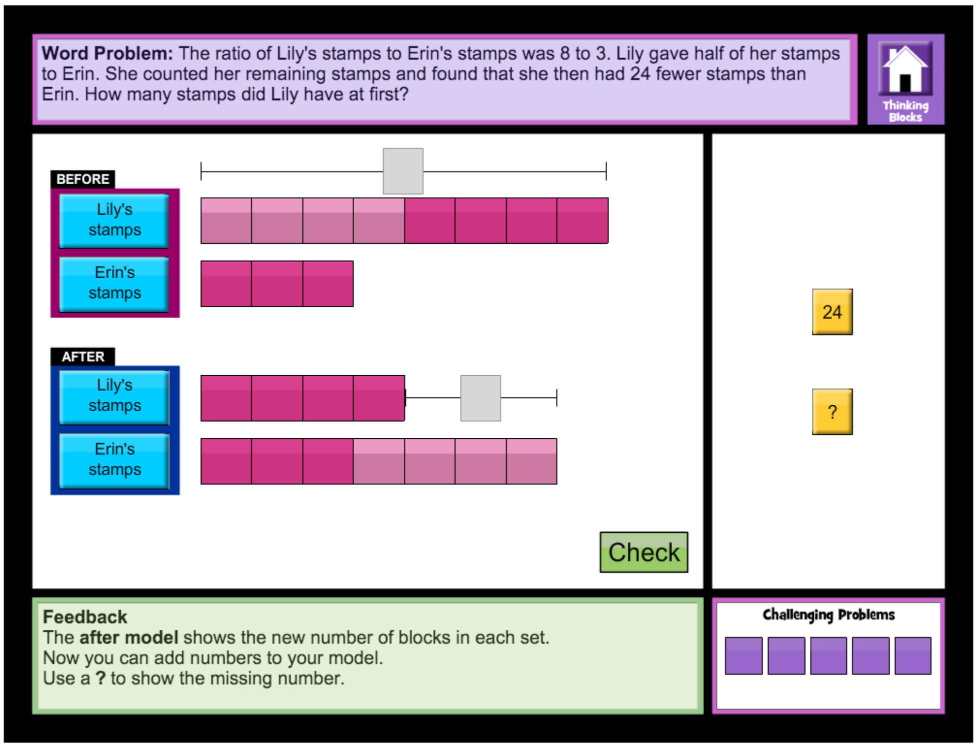
STAR Approach STAR is an acronym that refers to the steps recommended for solving a word problem[16]:

1. **S**earch the problem
2. **T**ranslate the problem into a form you can work with
3. **A**nswer the problem
4. **R**eview the answer

Searching the problem involves reading the problem and extracting facts relevant to the question posed. Translating it refers to using these facts to create a model that one can work with. This can be a set of equations, a diagram, an illustration, or any other model one feels comfortable with. Once this is done, this model is used to find the answer to the given problem. Once the answer is found, one should think about whether it makes sense and, if possible, check it in some way – for example, by plugging it back into equations extracted from the word problem.

Bar Model The bar model is a Singaporean method for visualizing relationships between quantities[18]. The model is generally used for word problems and is suitable in a wide range of situations, from simple comparison problems to more complex problems with variables. Using this model, the student is much more likely to understand and conceptualize the relationship between the quantities in question. For this reason, it is suitable for dyscalculics and children without learning disabilities alike.

Software Tools



Word Problem: The ratio of Lily's stamps to Erin's stamps was 8 to 3. Lily gave half of her stamps to Erin. She counted her remaining stamps and found that she then had 24 fewer stamps than Erin. How many stamps did Lily have at first?

Feedback
The **after model** shows the new number of blocks in each set.
Now you can add numbers to your model.
Use a **?** to show the missing number.

Figure 1.7: Math Playground Thinking Blocks: Ratio and Proportion[17]

Math Playground Thinking Blocks Math Playground offers several Thinking Blocks games in which it uses this model˜citethinkingblocks — a game for addition a subtraction, another for fractions, yet another for multiplication and division, and one for ratios and proportions. It also offers a special version of its addition and subtraction game for smaller children as well as a modeling tool for students of all ages.

A game usually gives the player some blocks and tasks him or her with assembling them into bars and labeling them. Then it asks to assign known and unknown values to the blocks, bars, missing spaces, or groups of bars. Unknown values are represented by a question mark. It then provides a button to resize the blocks to match the input values. Lastly, it asks the user to calculate and fill in the unknown values.

Evaluation

The bar model has a good track record of helping students comprehend a word problem, covers the "translate" part of the STAR approach, and there don't seem to be any superior generic alternatives. Therefore, the final exercise should utilize the **bar model** to help students understand the problem.

1.5 Requirements

For each category of requirements, I will first provide a list of the requirements and then expand on them in more detail below.

1.5.1 Requirements of the Core

CR 1. A module must be an application package.

CR 2. A module must be composed of an exercise and an editor portion.

CR 3. Entry points into the editor and the exercise must be accessible to the core.

CR 4. Results must include score and time spent.

CR 5. Results must adhere to the XML format specified by the core.

Discussion

One of the first things we decided on as a team was that modules would be **separate application packages** (in this case, Android application packages, or APKs) and thus would be downloaded and installed separately. This gives module developers the freedom to choose which technologies to use. The application's manifest file has to make the entry points into the exercise and the editor public so that the core can launch them.

For modules to be able to communicate with the core, communication between them needs to be well-defined. The core needs to be able to **launch an exercise with a certain assignment**, **receive data about how successful the student was**, and **launch a module to edit an assignment**. Assignments will be sent as strings and the core will not need to parse them, so their formats are up to each module to define. Results, on the other hand, must be sent back in a strictly defined XML format. These results must include at least the total time spent in an exercise and the total score.

More information about the reasoning behind these choices can be found in Patrik Pavelec's thesis about the core[20], which, at the time of this writing, is planned to be released by the end of 2016.

1.5.2 Non-Functional Requirements

NR 1. Exercises must be usable by children, editors by teachers.

NR 2. Modules must target Czech, but make localization possible.

NR 3. State should be saved when the application is paused (e.g. when the screen is rotated), but not when it is closed.

NR 4. Modules should be designed for a larger (with at least a 9-inch screen) tablet.

NR 5. The Android SDK should be used to develop the modules.

NR 6. JSON should be used as the format for saving assignments.

NR 7. Kotlin should be used as the programming language of choice, supplemented by Java when desired.

NR 8. Modules need to support API 21 and above.

NR 9. Modules should strive to support the lowest API they can.

Usability

Both the exercise and the editor portions need to be usable by their target users (defined below). The Android Human Interface Guidelines provide a thorough overview of best practices, such as comfortable target sizes and back button behavior, and should be followed unless a good case can be made for an exception.

Language

Modules are primarily designed around the Czech language, but should all be easily translatable into other languages. When choosing topics for modules, language-agnostic ones should be prioritized.

Saving State

Every module should save state when the system pauses it, including when it is rotated. However, state shouldn't be remembered when the application is stopped – for example, due to closing the application or returning to the application after a longer amount of time.

Form Factor

The modules need to be designed with a larger tablet – around 9 or 10 inches – in mind. They do not need to work well on phones. It is important, however, to keep screen rotation in mind, though, as what works in landscape mode might not work in portrait mode.

Core Libraries

While the core technology to build on could be chosen for each module separately, it makes more sense to choose a single foundation for all of the modules for the sake of consistency. If a module requires a different foundation, an exception could be made for it. The three main technologies to consider are game engines, of which libGDX seems the most promising, web technologies, and the Android SDK.

libGDX LibGDX is a 2D and 3D game engine. As a game engine, it shines when it comes to applications with copious amounts of **animation and graphics**. It is written in Java, which makes it a suitable choice for Android, but it also allows porting to other platforms, including iOS, web browsers with WebGL support, and the triad of popular desktop operating systems (Windows, Mac OS X, and Linux). It is released under a permissive open-source license, meaning that it can be used in both proprietary and open-source applications and thus licensing would not be an issue.

Unfortunately, as a cross-platform game engine, its integration with the Android platform is less than stellar and using Android libraries is needlessly complicated. Furthermore, font rendering is an issue.

Web Wrapper Another option would be to make a web application and then simply use a wrapper to make it act as an Android application. The main advantage to this would be having a cross-platform application that could also be hosted online, but the disadvantages would be numerous. The application would not be able to integrate deeply with the underlying platform and it would be less responsive than native applications.

Android SDK Using the Android SDK (Software Development Kit) allows the greatest integration with the Android platform and is the recommended basis for Android applications that are not graphics-intensive. It is also the basis of the core of the Dragon II application, so using it provides some consistency between the core and the modules.

Evaluation Upon deeper reflection, platform integration, familiarity, and code portability were chosen as the deciding factors. Given that the decided-upon modules don't require much animation if any, 2D and 3D animation was not among them. In this table and all subsequent ones, five points is the maximum amount of points that can be awarded for each category.

	LibGDX	Web Wrapper	Android SDK
Platform Integration	3	1	5
Familiarity	2	1	4
Code Portability	4	5	2
Total	9	7	11

Table 1.1: Core library evaluation

Based on this assessment, the **Android SDK** was selected for the project.

File Format

Assignments will be saved as files outside of the module, but will be interpreted and written exclusively by the module. While the structure of an assignment will vary between modules, all modules should ideally use the same markup language, for consistency and ease of implementation.

XML XML is a human-readable format composed of elements. Elements consist of start and end tags, with nested elements in between. Elements may also have attributes. and adaptable for a variety of needs. Classes for working with XML are part of the Android SDK and there are numerous libraries for XML in Android too.

JSON JSON is more compact than XML and, arguably, easier for a human to read. Tools for working with JSON are part of the Android SDK as well, and there are many libraries that work with JSON too. JSON is considerably simpler than XML, allowing for faster parsing and writing and easier translation of objects, but falling short for complex document structures. This isn't a problem, as the assignments won't be very complex.

Custom Format A custom format could be tweaked to suit certain needs. For example, file size, human readability, or speed of interpretation by the module could be prioritized. However, more code would need to be written to interpret the format and people would need to learn about the format first in order to be able to fully understand it.

Evaluation **JSON** was chosen as the file format to use:

	XML	JSON	Custom Format
Platform Support	5	5	1
Familiarity	4	3	4
Ease of Implementation	5	5	1
Simplicity	3	5	5
Human Readability	3	4	5
Total	20	22	16

Table 1.2: File format evaluation

Programming Language

The programming language is partly defined by the core libraries. Both the Android SDK and LibGDX are written in Java and this section will assume the selection of one of those two. Both allow for the use of Java as well as languages interoperable with Java.

Java While Android's Java API does not completely align with official Java versions, it is fair to say that it does implement many of the features in Java 7 SE, but none of the new features in Java 8. Java is one of the most commonly used programming languages, and thus there is a rich ecosystem behind it. It is a statistically-typed language and it uses a garbage collector to manage memory. There is differentiation in how primitive and non-primitive datatypes behave, and thus developers need to approach these two types differently. As a language more than twenty years old, it has been heavily iterated on, but has some deeper flaws that prompted the creation of languages aiming to fix them.

C and C++ Google supports using the C and C++ programming languages to code parts of Android applications[1]. C and C++ are lower-level languages, meaning that there is less abstraction between them and machine code than there is with Java. This leads to higher performance. On the other hand, both languages require the programmer to do more work with regard to memory management, making these languages more prone to errors and the code often longer and harder to get oriented in. Google recommends against using C and C++ except in special cases where performance is a top priority, arguing that, in most cases, the performance gain is negligible and isn't worth the increase in complexity.

Kotlin Kotlin is a recent language introduced by JetBrains, the developers behind the IntelliJ IDEA IDE, which Android Studio is based on. It is interoperable with Java, meaning Java classes and Kotlin classes can be used in the same package, and seeks to address some of Java's shortcomings. One of the key things Kotlin addresses is null safety[14]. In Kotlin, nullable types and non-nullable types are differentiated, helping prevent null pointer exceptions. Kotlin is also more compact than Java, allowing much shorter notation for getters and setters, obviating the need for checking for null objects in most situations, automatically loading views with IDs from Android's XML layouts, and providing a broader range of type inference. Furthermore, primitive types are abstracted from the developer and are handled similarly to objects. JetBrains provides support for Kotlin via a plug-in for their IDEs as well as for Android Studio.

Using a language that hasn't been around very long is generally considered dangerous in terms of code maintenance. However, there are two things to take into account here. Firstly, Kotlin compiles into JVM bytecode, which, given Java's popularity, is likely to continue to be supported. Secondly, each module is relatively small and insular and doesn't serve as a building block of something bigger, meaning the maintenance of a module isn't too important.

Evaluation I am deliberately excluding C from my evaluation, as C++ has basically the same advantages, but it also features object-oriented concepts which are key to writing the code I will be writing.

	Java	C++	Kotlin
Familiarity	4	3	2
Code simplicity	3	1	5
Error prevention	3	1	4
IDE support	5	3	5
Terseness	2	2	4
Online support	5	5	2
Total	22	15	22

Table 1.3: Programming language evaluation

Java and Kotlin were rated equally, but each has different advantages. While Kotlin feels like an improvement over Java, one area where Java outshines it is online support. Because Java and Kotlin files can be part of the same package, I decided to allow **both** — Kotlin where possible, to avoid errors and have more compact code, and Java if Kotlin falls short.

License

To have the maximum impact with this project, I am planning to release it under an open-source license after I present this thesis (in order to avoid plagiarism concerns). When looking at different open-source licenses, I found three significant license categories — strong copyleft, weak copyleft, and permissive. I have narrowed down my license options to three representatives of these licenses that I find most suitable: GPLv3, MPL 2.0, and Apache 2.0.

GPLv3 Version 3 of the GNU General Public License, or GPLv3 for short, is a strong copyleft license. Derivative works must be released under the same license and version and most not restrict the rights provided by the license any further. The Free Software Foundation, which authored this license, claims that both static and dynamic linking of GPL-licensed libraries makes a work derivative, though there is no consensus on this point. GPL-licensed works must make it easy to find the source code for those that have access to the executable and both copies and derivative works must include the text of the license. One may also specify the license as ”GPLv3 or later”, making it possible to combine software released under this license with software released under newer versions of the license.

MPL 2.0 The Mozilla Public License is a bit more lenient, allowing proprietary software to both statically and dynamically link libraries released under this license. However, derivative works must still be released under this license and must at least make it clear where to find a copy of the full text of the license. MPL 2.0 is compatible with GPLv3, meaning that software under the GPL license can make use of MPL-licensed code.

Apache 2.0 The Apache License is a permissive license, meaning that derivative works do not have to use the same license or be open-source at all. However, it does require that all license notices be kept intact.

Evaluation The Apache License was found to be unsuitable, as I deeply do not want anyone to be able to profit off of taking the code and modifying it without contributing the improvements back. As for the remaining two licenses, I decided to use both: the **MPL for libraries** and the **GPL for the rest** of the application code. This way, even proprietary software can use a MPL-licensed library, but it has to contribute the changes to this library back, making it stronger. The modules themselves, however, cannot be used as part of larger proprietary software.

API version

The core requires API version 21 and above, making it available only on a subset of Android devices. To work wherever the core works, modules need to support API version 21, but could support lower API versions as well. This is useful both in case the core gains support for older Android versions and in case the code is reused elsewhere. As each module has different requirements, each module might be able to support a different minimum API. Generally, **modules should support the lowest API they possibly can**, unless it would be highly detrimental to them or be much harder to implement.

1.5.3 Functional Requirements

As the functional requirements are quite straightforward, they do not warrant lengthy explanations and therefore will simply be listed. Reasoning behind the decisions related to the exercises can be found in the *Exercises* section above.

Functional Requirements of an Exercise

An exercise should allow the learner to:

FR 1. Launch an assignment

FR 2. Learn about his or her errors when they occur

FR 3. Correct his or her errors

FR 4. Give up

Functional Requirements of an Editor

An editor should let the teacher:

FR 5. Create a new assignment

FR 6. Edit an existing assignment

FR 7. Customize the settings

FR 8. Exit without saving changes

1.5.4 Clock Module

FRC 1. Provide **at least three different configurations** (e.g. minutes, hours, and a combination of the two), so that the student can learn the two scales separately.

FRC 2. Allow the player to **freely move and play with the clock hands**, to allow them to get a sense of both hands.

FRC 3. Present time in words rather than in numbers when possible, as words often seem friendlier to dyscalculics than numbers.

1.5.5 Syllable Length Module

FRS 1. Ask the user to categorize words by syllable length.

FRS 2. Indicate when a word is miscategorized.

FRS 3. Allow teachers to add their own words and categories.

1.5.6 Fill-in-the-blank Module

FRF 1. Offer a limited set of choices with which to fill in the blank (as there may otherwise be multiple correct answers).

FRF 2. Show the filled-in portion in context of the word.

FRF 3. Allow teachers to add their own words and choices of parts to fill in.

1.5.7 Number Bonds Module

FRN 1. Allow using dot patterns from dice instead of numerals.

FRN 2. Use color coding to indicate the composition of a larger number if possible.

FRN 3. Allow teachers to tweak the settings for an assignment.

1.5.8 Word Problem Module

FRB 1. Let students find missing quantities

FRB 2. Allow teachers to set the questions and customize the bars to suit them.

1.6 Use Cases

While modules may cater to students with different disabilities, use cases for all modules can be defined in a general way. Specific modules always offer a single task to complete at a given time, so their usage should be straightforward.

1.6.1 Student

- The student can launch the exercise to complete an assignment of his choice.
- The student can quit the exercise prematurely, losing all progress.

1.6.2 Teacher

- The teacher can create a new assignment from scratch.
- The teacher can edit an existing exercise.

CHAPTER 2

Design

In this chapter, I will cover the code style as well as go over the design of the libraries and modules that I will be implementing. This includes the visual and interaction design as well as the assignment structure of a module.

2.1 Code Style

For better code readability as well as simply for the sake of consistency, it is useful to establish a code style guideline before starting work on a development project. It is important to consider both the naming of elements as well as the format of inline documentation.

2.1.1 Naming

Kotlin's style guide recommends using **camel case for names** [12], with methods and properties starting with a lowercase character and types starting with an uppercase character, much like is recommended for Java code. Kotlin's design prioritizes using direct property access over setters and getters, so using Hungarian notation would not be appropriate. Given Android Studio's color coding of member and static variables, Hungarian notation would provide little value anyway.

In Kotlin, there are two types of variables: read-only, created using the keyword `val`, and read-and-write, created using the keyword `var`. As both of these are equally efficient to use, the use of the read-only `val` is much greater than Java's `const`. While all-caps constants may often be seen in Java, they are not appropriate for Kotlin's `val` type. However, Kotlin also allows compile-time constants, denoted by the `const` keyword, for which all-caps is appropriate. Enum names (including names of integers that serve as enums) should also be all-caps.

An area where naming standards aren't very well-defined are project resources. Resource files and most resource names generated by Android Studio tend to be lowercase with underscores used as separators, except colors and styles use camel case. Color names begin with a lowercase letter and style names begin with an uppercase letter. There doesn't seem to be a naming convention for IDs. However, since Kotlin Android Extensions uses view IDs as names for variables that refer to those views, it is best to use the same style for IDs as for Kotlin variables.

Naming Style	**Items**
Camel case, lowercase first	Methods, properties, variables, color resources, and resource IDs
Camel case, uppercase first	Types and style resources
All lowercase, separated by underscores	Names of other resources and of resource files
All caps, separated by underscores	Compile-time constants and enums

Table 2.1: Naming style

2.1.2 Documentation

Kotlin's language for documentation is called KDoc and its documentation generator is Dokka[13]. The KDoc syntax is very similar to the syntax of JavaDoc, except tailored to Kotlin and using Markdown for inline markup instead of HTML, making documentation in source files shorter and more readable.

2.2 Android Components

In the following text, I will sometimes refer to some Android components that I should clarify first:

- A **Floating Action Button** is a round button that floats above the user interface. It holds an important action and, according to Google's design guidelines, there should be no more than one per screen.

- A **Pager** is an element that allows flipping between different pages while the rest of the screen remains unchanged.

- A **Recycler** is an element that shows a list of items. Its name refers to its implementation, which I will not discuss here.

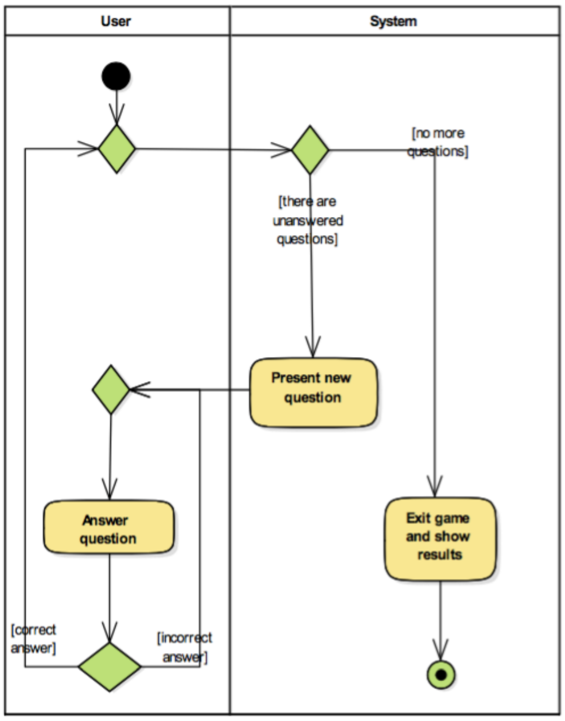
2.3 Custom Libraries

To avoid code duplication, some parts of the code should be split up into libraries that several modules can then use. In particular, each module will need a part that stands in for the core, to be able to test that launching the exercise and the editor using intents works as expected, without having to rely on the core to be complete. It also needs to know certain constants defined by the team to be able to communicate with the core. This functionality will go under the **Commons Library**, a library holding elements useful to Dragon II modules in general. Another aspect that a number of modules share is a **Question Stepper** – an element that transitions between different questions. Lastly, some of the editors share a common list structure, so a component for it will be handy. This component will be provided through the **Editor Recycler** library.

2.4 Question Stepper

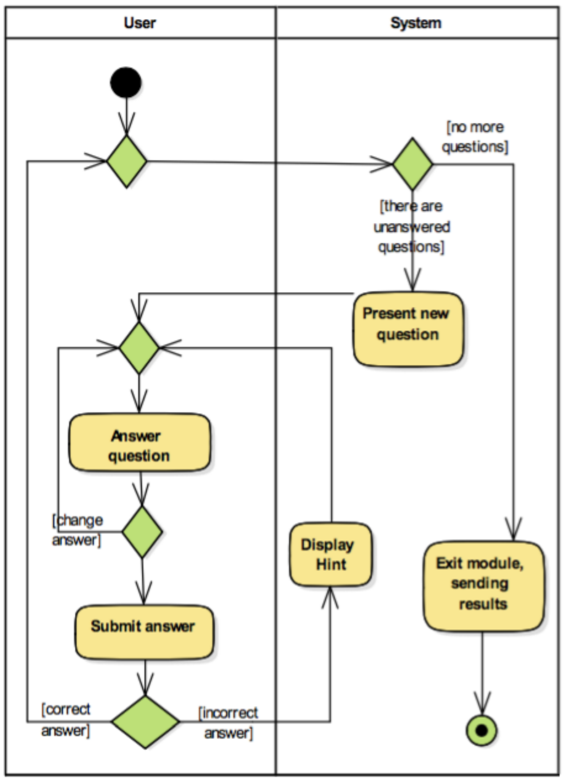
2.4.1 Flow

As several modules are built around a Question Stepper, an element that presents questions one at a time, it is important to consider how this pager should work. These are the main variants that I considered:



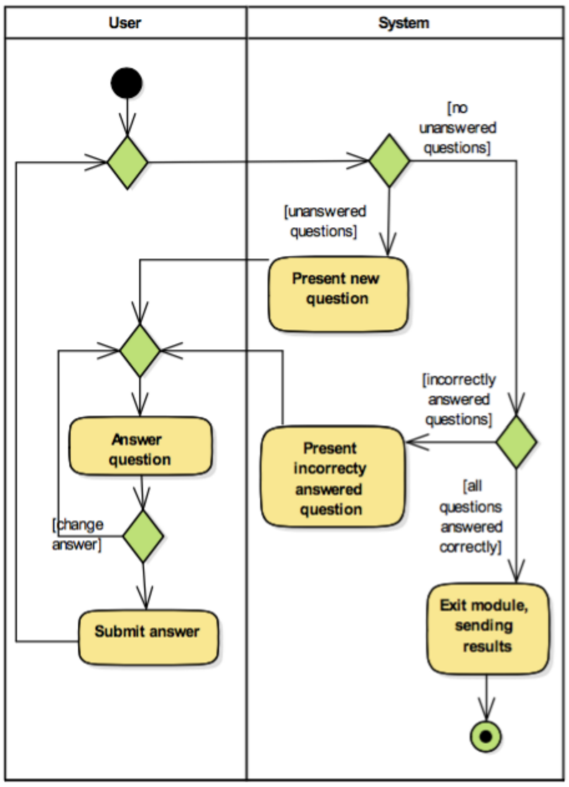
Variant 1

In this variant, the user is made to answer a question until they get it right. It may motivate the user to consider why their original answer is wrong, but it can prove to be frustrating if they are unable to get it right. The answer is submitted right away, which can also prove frustrating when one accidentally taps something. It also narrows down the range of possible exercises, as exercises with answers requiring multiple steps don't work in this scenario.



Variant 2

In this variant, the user can change his or her mind before submitting. This allows for more complex ways to answer a question as an answer may now consist of multiple steps, but makes submitting an answer take longer than necessary. It also gives more control to the user, as it doesn't register accidental taps as wrong answers. Additionally, a helpful message is shown to the user when she or he gets a question wrong, thereby lessening the frustration of getting it wrong and increasing the chances of getting it right.


User
System
[no unanswered questions]
[unanswered questions]
Present new question
[incorrectly answered questions]
Answer question
Present incorrecty answered question
[change answer]
[all questions answered correctly]
Submit answer
Exit module, sending results

Variant 3

In this case, users can still amend their answers, but rather than being asked to answer the same question over and over until they get it right, they are shown the reasoning behind the correct answer, which they can learn from, and given a new question right away. The incorrectly answered questions are presented again after all questions have been answered, testing whether the user learned from his or her mistakes. This goes on until all questions have been answered correctly. The main disadvantage to this is that it allows the user to run through the exercise, answering incorrectly, get the answers without thinking about them, and then mindlessly put in those answers without learning anything.

Evaluation

	Variant 1	Variant 2	Variant 3
Speed	4	3	3
Prevention of Accidental Errors	1	3	3
Prevention of Cheating	4	4	1
Prevention of User Frustration	1	3	5
Total	10	13	12

Table 2.2: Question Stepper flow evaluation

Variant 2 was deemed as most suitable. It encourages learning and prevents frustration by providing a helpful message when the user answers incorrectly, yet it also avoids the potential cheating problems presented by variant 3.

2.4.2 Score Calculation

The score will be calculated as the number of correct attempts over the sum of correct and incorrect attempts. (Attempts where no answer was selected are not counted.) If the exercise isn't completed, the final score will be 0.

$$score = \frac{wrongAttempts}{correctAttempts + wrongAttempts} \cdot 100\% \tag{2.1}$$

2.4.3 Answer Types

The Question Stepper will allow for three types of answers:

- A **correct answer**, which will allow a user to proceed further
- An **incorrect answer**, which will show a hint specific to the exercise and subtract from the score on submission
- A **non-answer**, used when no answer is selected or when it is incomplete (e.g. a text entry field is empty). Submitting a non-answer will have no impact on the score and will show a hint indicating that the answer is incomplete.

2.4.4 Visual Design

The Question Stepper will feature an indicator of progress up at the top and a continue button at the bottom right. There will also be an empty area at the bottom, where messages may appear.

The progress indicator will show how many questions have been answered correctly, what the current question is, and how many questions remain. Answered questions will use green text, fill, and stroke, but these colors will be reserved enough as to not draw much attention. The current question will draw a bit more attention, shown in bright yellow. The remaining questions will be drawn without fill, with text and stroke having a faint gray color. The contrast between the three will be high enough so that the colorblind will be able to differentiate without issue.

The message area will initially be empty. A message will appear inside a floating rectangle in the center of the area, with a close button inside the rectangle on the right.

The continue button will be a standard Floating Action Button at the bottom right, using a checkmark as its icon.

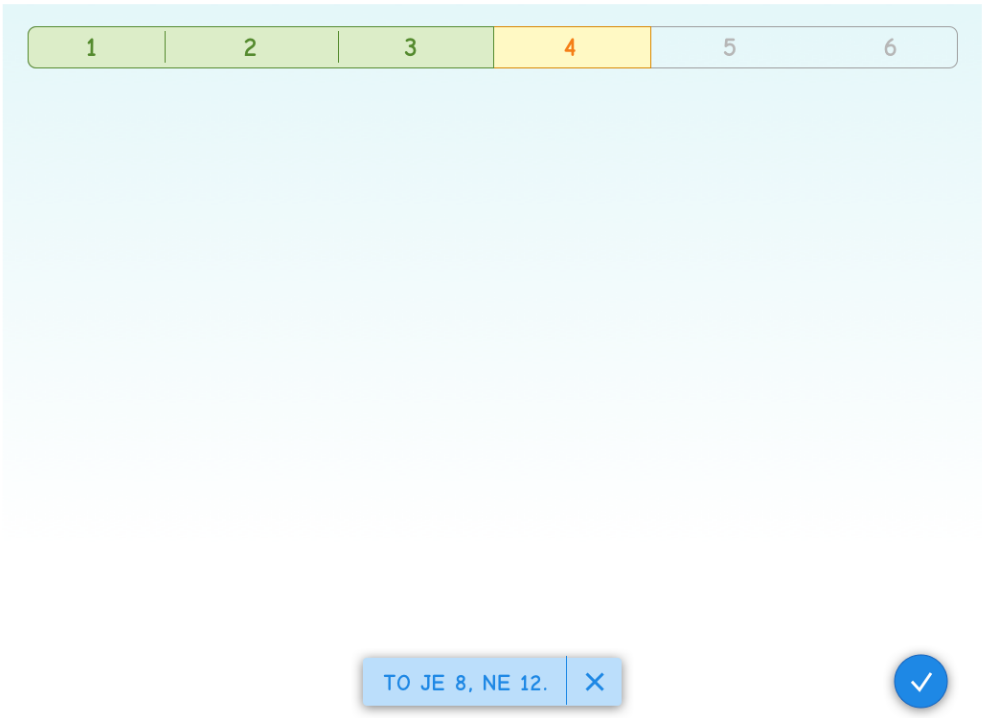


Figure 2.1: Mockup of the Question Stepper

2.4.5 Behavior

- Tapping the **close button** on a message hides the message.
- Tapping the **continue button** submits the answer.
  - If the answer is incorrect, a message is shown.
  - If the answer is correct and it is the last question, the exercise will close, returning the user to the core.
  - Otherwise, if the answer is correct, the next question will be shown.

2.5 Editor Recycler

A typical Editor Recycler activity will consist of a list of cards, defined separately in each module. Each card will have two modes: an edit mode and a view mode. By default, all cards start in view mode. Tapping a card puts the card into edit mode, letting the user modify the card's contents. Only one card can be in edit mode at a time. The content of cards is saved automatically, but changes are made to the underlying assignment only once the user chooses to save the assignment.

Behavior

- Tapping a card that isn't being edited will put it into edit mode as well as put any card in edit mode into view mode.
- Tapping a card in edit mode puts it into view mode.
- Changes made to a card are saved automatically.
- Tapping the "Add" Floating Action Button will add a card to the editor. (The initial content of the card is up to the module.)

2.6 Clock Module

2.6.1 Exercise Types

The module will have the following exercise types:

- **Hour**, for learning how to read the hour hand
- **Minute**, for learning how to read the minute hand
- **Hour and minute**, for learning how to combine the hour and minute hands and read the time

2.6.2 Assignment Structure

Each assignment will consist of one or several assignment components. Each component will consist of the following items:

- **Exercise type**
- **Hour labels**, specifying if hours should have numeric labels and at which frequency (every hour or every quarter hour)
- **Minute labels**, specifying if minutes should have numeric labels and at which frequency (every 5 minutes or every 15 minutes)
- **Question count**, specifying how many questions of this type should appear in the assignment

2.6.3 Exercise

Visual Design

The exercise will use the Question Stepper as a basis. The main content of this pager will consist of textual directions up at the top and a large analog clock. The clock will have either an hour hand, a minute hand, or both, depending on the exercise type. The clock may also have numeric labels. Hour labels will

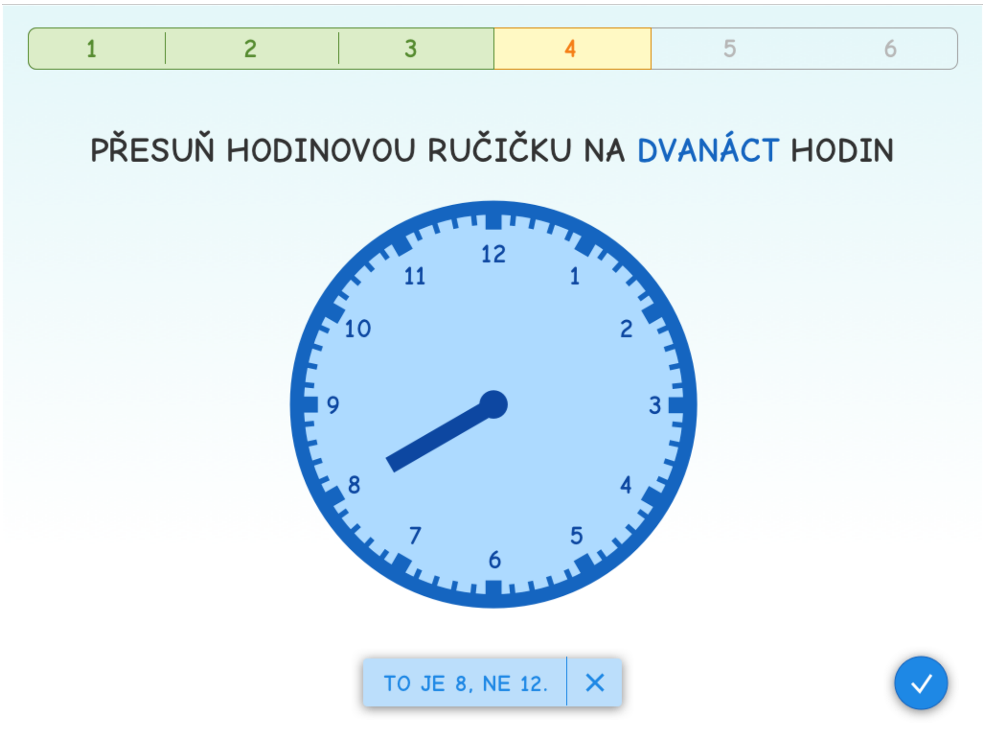


Figure 2.2: Mockup of the Clock exercise

appear on the inside of the clock face, as is standard practice. Minute labels will appear outside of the clock face, to allow both hour labels and minute labels to be shown at the same time and to make it clear that minute labels generally don't appear on analog clocks.

Behavior

- Dragging a **clock hand** around the clock will rotate it to the nearest allowed position. Only positions where the hand could realistically be on a clock are allowed; in-between positions are not.
- Dragging the **minute hand** will also rotate the hour hand, if visible, to the corresponding location between the past hour and the upcoming hour, just like on an actual analog clock.

Hint Format

The hint will only mention the time that the clock is currently showing. The user can then compare that to the time in the directions and see what mistake he or she is making.

2.6.4 Editor

The editor for this exercise will be built around the Editor Recycler. Each card in the Recycler will represent a component of the assignment described in the *Assignment Structure* section above, allowing the teacher to set the question count, hour labels, minute labels, and the exercise type.

2.7 Syllable Length Module

2.7.1 Assignment Structure

Each assignment will consist of a **list of patterns** and a **list of words** associated with each pattern.

2.7.2 Score Calculation

The user will be penalized for every incorrect categorization of a word. The final score will be the number of correctly categorized words divided by the number of categorizations overall.

$$score = \frac{correctAttempts}{totalAttempts} \cdot 100\% \tag{2.2}$$

2.7.3 Exercise

Visual Design

The graphical user interface of this exercise will be split up into two main sections: a section with words to categorize on the left and a section with syllable pattern categories on the right. Syllable pattern categories will show the syllable pattern on the left and have a box, initially empty, for dragging words in on the right. Syllable patterns will use the same symbols as in the Dragon I implementation: a slanted line for long syllables and a dot for short syllables, as there is no standard representation. (While a dot is commonly used to represent short syllables, representations of long syllables vary between horizontal, vertical, and slanted lines.)

Behavior

- Dragging and dropping a **word** onto the syllable pattern category it belongs to will move it there and disable the word's further dragging.
- Dragging and dropping a **word** onto the wrong syllable pattern category will move the word to this category and highlight the word as incorrect. The word will remain draggable.
- Only when all words are in the correct syllable pattern categories will the exercise close and return results to the core.

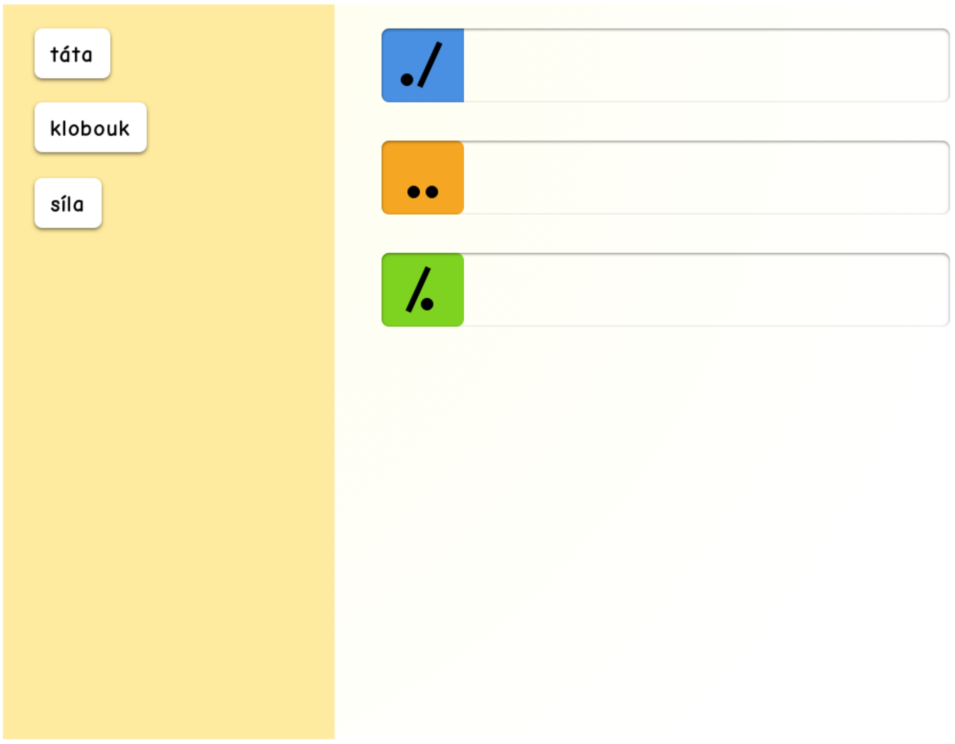


Figure 2.3: Mockup of the Syllables exercise

2.7.4 Editor

The editor will use the Editor Recycler as a basis. Each card in the Recycler will feature a syllable pattern on the left and a text field for entering words belonging to that syllable pattern on the right. The words will be separated by spaces, as the syllable patterns will always pertain to single words only. Adding a new element will launch a dialog asking the user to enter a pattern.

2.8 Fill-in-the-blank Module

2.8.1 Exercise

Visual Design

This exercise module will use a Question Stepper as its basis. Each question will show a word or phrase at the top, with an underscore representing the part of the phrase to be filled in. This underscore will be colored, to show that this part is changeable. The options to fill in will be offered below, using standard Android radio buttons.

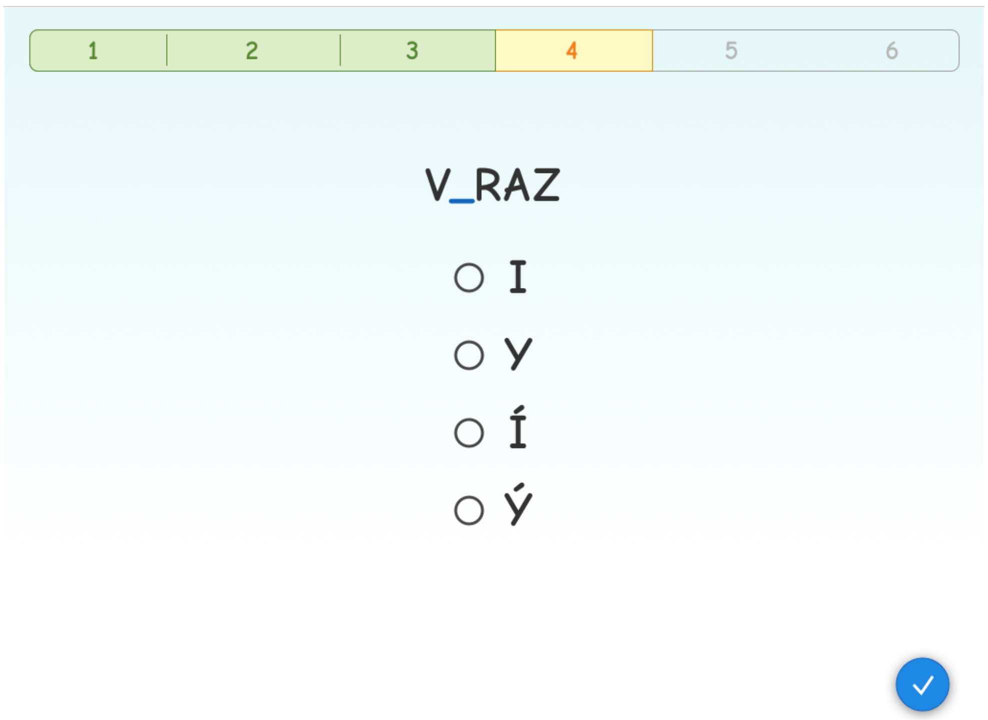


Figure 2.4: Mockup of the Fill-in-the-blank exercise

Behavior

- Selecting an option will place the part of the phrase associated with this option into the phrase above. If a different phrase is already there, it will be replaced. This part of the phrase will remain colored.

2.8.2 Editor

Just like the other editors, this editor is based on the Editor Recycler. A card in this Recycler is composed of a phrase on the left and a text field on the right for entering words. The words filled in here will be the ones which are completed with the phrase on the left. A period will be used to indicate where the phrase should be filled in. The phrases in the cards will then be used as choices in the assignment. New cards will be added with an example, to get the point across.

2.9 Number Bonds Module

2.9.1 Assignment Structure

Just as with the Clock module, this module's assignment is split up into components. A single component consists of these elements:

- **Sum range**, as specified by minumum and maximum sum, defining the range in which the sum should be generated
- Flag determining whether to **use dot patterns** instead of numerals
- Maximum **difference between choices**, specifying how much the different choices should differ
- **Question count**, saying how many questions of this kind to generate

2.9.2 Exercise

Visual Design

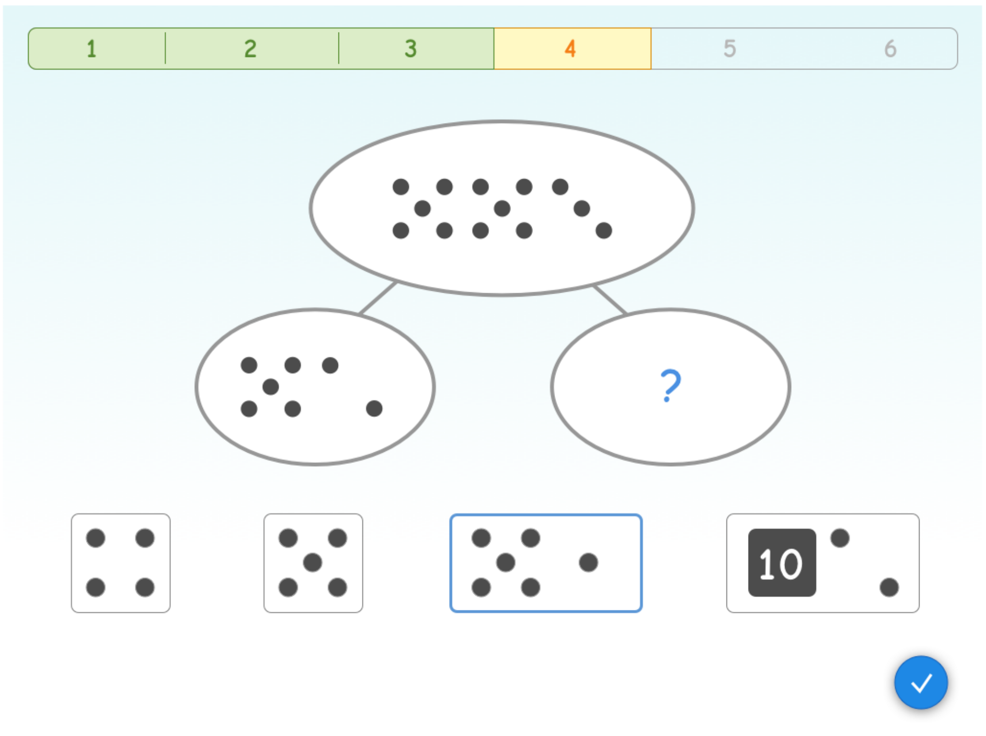


Figure 2.5: Mockup of the Number Bonds exercise

The Question Stepper will serve as the basis for this module. Each question will show a number bond diagram, where the sum is presented in an oval at the top and the summands below, inside ovals connected to the sum oval. The rightmost summand oval will be empty. Below the diagram will be a set of four choices for numbers that should fill this oval.

Behavior

- Selecting a choice will put that choice into the rightmost summand oval, replacing any contents that oval may have.

Hint Format

The hint will allude to the child's current choices. If no choice is selected, the hint will tell the user that nothing is selected. If the wrong choice is selected, the hint will tell the user that "[Summand A] and [Summand B] don't make up [Sum]".

2.9.3 Editor

The cards in the editor, centered around the Editor Recycler just like the others, adopt the same structure as the components specified in the *Assignment Structure* section above: a text field for the question count, two text fields for the maximum and the minimum of a sum range, a checkbox for using dot patterns, and a text field for the maximum difference between choices.

2.10 Word Problem Module

2.10.1 Assignment Structure

In this module, the assignment structure is quite complex and layered. The assignment itself contains the story and the bar group, which serves as the basis for both the bar model and the questions that pertain to it.

The **bar group** contains the following elements:

- The **bars**, described in more detail below
- An optional **question** about the value of the whole bar group
- A flag specifying whether to **display the total value** of the group (the sum of the values of bars) in the bar model

A **bar** consists of:

- A **label**, showing what the bar stands for
- **Blocks**, which I will describe below

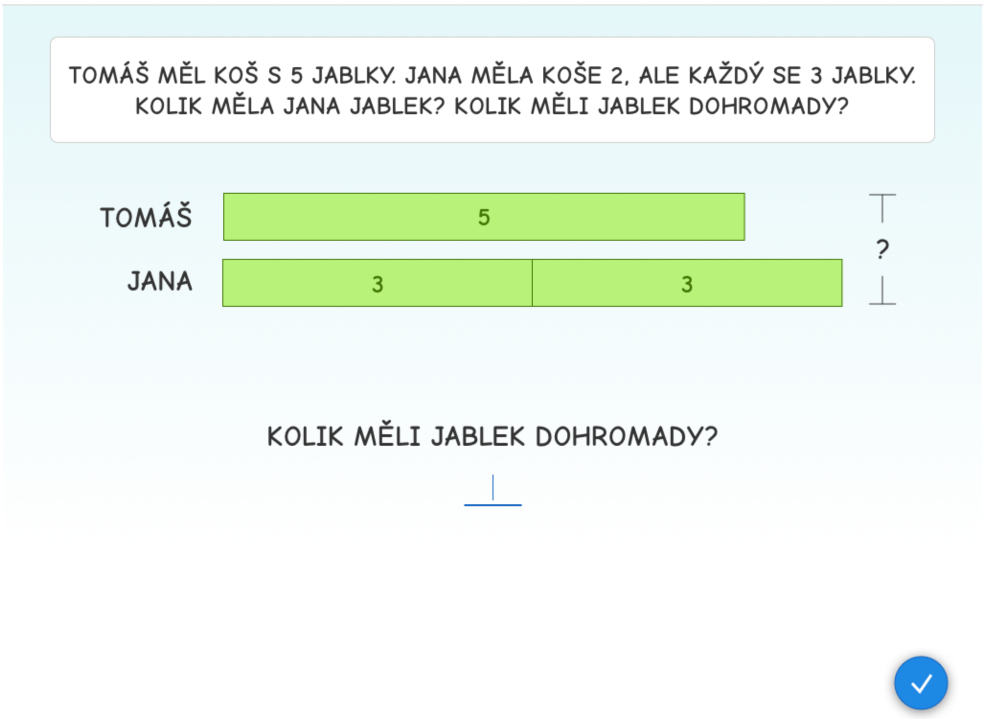
- An optional **question** about the value of the bar (the value of a bar is the sum of the values of its blocks)
- A flag specifying whether to **show the value** of the bar
- A flag specifying whether to **show the value above** or below the bar

A **block** is made up of these parts:

- The numerical **value** of the block
- The **color** that the block should have
- An optional **question** with the answer being the value of the block
- A flag specifying whether to **show the value** of the block

2.10.2 Exercise

Visual Design



TOMÁŠ MĚL KOŠ S 5 JABLKY. JANA MĚLA KOŠE 2, ALE KAŽDÝ SE 3 JABLKY. KOLIK MĚLA JANA JABLEK? KOLIK MĚLI JABLEK DOHROMADY?

Figure 2.6: Mockup of the Word Problem exercise

This module will use a modified version of the Question Stepper without its progress indicator. The stepper will cycle through the questions extracted from the bar group, letting the user input the value associated with a certain part of the bar model.

2.10.3 Editor

Given the complexity of the assignment structure, this module requires an editor with two screens. The first screen will allow the user to input the description of the word problem, tweak the bar group's properties, and will also provide a list of the individual bars. Tapping on a bar will take the user to a screen with settings for that specific bar. Along with simple toggles and inputs, this screen will also contain an Editor Recycler with the blocks in this bar. The Recycler will allow inline customization of the block.

Chapter 3

Realization

3.1 Development Environment

To set up the development environment, a developer needs to install three things:

- **Android Studio**, provided by Google and available from the Android Developers website. The Studio should come bundled with the **Android SDK**.
- **Java SDK**, provided by Oracle
- The **Kotlin** extension for Android Studio, provided by JetBrains and available from the official Android Studio extension repositories (which it shares with IntelliJ IDEA)

As all of these tools are being actively maintained, the latest stable versions are likely to deliver the best results.

To run an application from Android Studio, one needs to either have access to an Android device or to set up an emulator.

3.1.1 Device

Using a device with Android Studio is relatively straightforward. On the Android device, one needs to enable developer options. To do so, one needs to open up Settings and find the section with device information. On stock Android, this section is called "About phone". In this section, one needs to tap on the build number seven times. A message saying that developer options have been enabled should appear. After this is done, there should be a new "Developer Options" section on the Settings homepage. In this section, one needs to enable "USB Debugging". Android Studio should then recognize the device when it is plugged into the computer via USB and offer it as a choice when one runs an application.

3.1.2 Emulator

If one lacks an Android device, one can install a virtual device and run applications in an emulator. To do this, one has to open up the AVD Manager – it is a menu item in the Android submenu under the Tools menu. Once there, clicking on "Create Virtual Device" launches a wizard that guides a person through setting up the virtual device. After the virtual device is created, it should show up as a choice in the "Run..." dialog.

3.2 Installation Manual

Once the core is developed and both the core and the modules are published on the Google Play Store, installation will be quite straightforward and in line with what most Android users are accustomed to — standard installation from the Google Play Store. The core may also include links to the Play Store module pages, for easier discovery.

Currently, though, the only way to install these modules is using standalone executable APK files. These files will be released along with the code under an open-source license on GitHub after this thesis is published. To install the modules, one has to have the installation of third-party applications enabled on the Android device. To enable it, one has to visit the "Security" section in the Settings app and check the "Unknown Sources" box. Once this is done, one can simply download and run the APK files and Android should offer to install them.

3.3 Communication with the Core

The core can open an exercise or an editor by sending an `Intent` to the application along with a `String` that contains the assignment. In the case of the editor, the assignment can be blank or null.

In both cases, the core expects to receive a result back. In the case of the exercise, the result sent back is a `String` in the XML format, providing information about the score, the time it took to complete the assignment, and optionally exercise-specific information. In the case of the editor, the result sent back is a `String` containing the edited assignment.

3.4 Libraries

3.4.1 Question Stepper

The Question Stepper is implemented as a layout that contains a custom pager for cycling through the different questions called the `QuestionPager`. It also contains a custom component with the name `ProgressView` showing the progress made and a Floating Action Button serving as a button for submitting.

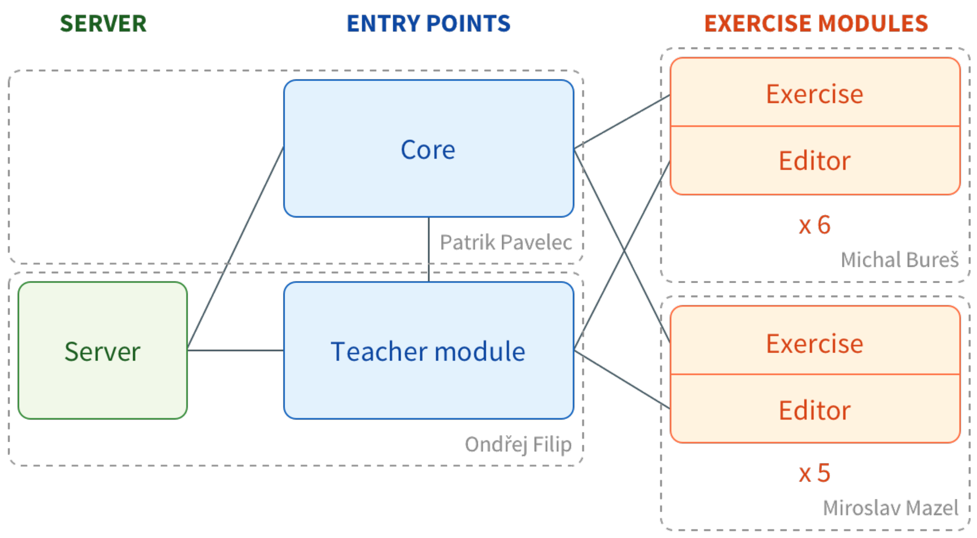


Figure 3.1: Communication between the different parts of Dragon II

Individual questions need to extend the abstract `QuestionFragment` class, which keeps track of the time spent on an individual quesion and of the wrong answers given. This is done using the MVP architecture – there are corresponding `QuestionFragmentModel` and `QuestionFragmentPresenter` classes. The view and corresponding presenter portions must be implemented separately for each module.

Communication with a Module

As the `QuestionFragment` holds information about a question, which is then used to calculate the total score and the total time spent, it needs to be able to communicate with `ExerciseActivity`. It does this through the `QuestionStepper` view, which, just like other Android Views, has methods to get this data — a `getTime()` function for getting the total number of milliseconds and a `getPercentage()` function for getting the final percentage. The `QuestionStepper` gets this information by talking to its `QuestionPager`'s adapter, which is the class that takes care of populating its individual pages.

3.4.2 Editor Recycler

The Editor Recycler provides a Recycler with read/write modes built in. The library takes care of switching the modes and saving content at the right time. Editors using the recycler will need to extend the abstract `EditorHolder` to describe the view behind the edited items and how the view's contents are saved. The `EditorAdapter` needs to be extended to generate this holder.

3.5 Modules

3.5.1 General Module Structure

A module tends to be split into three packages:

- common
- exercise
- editor

The **common** package contains classes useful to both the exercise and the activity. A class representing the structure of an assignment as well as a class that can read and write an assignment are usually parts of this package.

The **exercise** package contains the `ExerciseActivity`, the main entry point into the exercise part of the module, as well as all other classes used exclusively within an exercise. This often includes classes for working with individual questions in an exercise, split into classes following the Model-View-Presenter (MVP) architecture.

The **editor** package contains only classes used within the editor. The point of entry of the editor is the `EditorActivity`. Some of the heavy lifting behind the editor is done by the Editor Recycler library, so the package tends to contain implementations of abstract classes from this library.

Outside of these packages is the `MockCore` activity. For the time being, this activity is the initial screen of the application. Once the Dragon II core is completed, this activity will be relegated to testing builds only and the release version of the application will not include it.

3.5.2 Module-Specific Libraries

Some modules have their own libraries, for easier reuse by other projects.

- The Clock module has the **Clock View** library, which hosts a custom `View` of a clock with manipulable hands.
- The Number Bonds module has the **Dot Pattern** library, which allows generating a `Bitmap` or a `Drawable` with a certain dot pattern.
- The Word Problem module has the **Bar Model** library, which allows generating custom bar models.

3.6 Third-Party Libraries

Here I would like to give a short rundown of the third-party libraries that I am using.

Kotlin's Standard Library As I am using Kotlin, I also need to use the standard library that it is built on. After installing the Kotlin plug-in in Android Studio, the IDE automatically offers to set up the project for Kotlin. This adds the Kotlin dependencies to the project's Gradle build files.

Android Support Libraries As the goal is to support as many versions of Android as possible, I am using Android's support libraries. These libraries are designed to bring new features in Android to older Android versions and allow developers to use the same code for newer and older Android versions. The activities and fragments that I use come from these libraries, as well as the RecyclerView that the Editor Recycler is based on and the Floating Action Buttons that I use in many of my layouts.

Jackson Jackson allows me to translate objects in my application into JSON files and vice versa, generally with very little code. More importantly, the Jackson library includes a module for integration with Kotlin, allowing me to use the language for the whole project. I use Jackson for both reading and writing an assignment object.

Petrov Kristiyan's Color Picker As Android doesn't offer a color picker component, I looked for an open-source library that would offer one. I chose Petrov Kristiyan's MIT-licensed library, as it offered the greatest retro-compatibility out of all the color pickers I looked at. The color picker is used only in the editor of the Word Problem module.

Chapter **4**

Testing

4.1 Unit Testing

Unit tests allow the developer to test one class at a time, without being influenced by the implementation of classes that this class interacts with. To isolate an object from the objects it depends on, a developer replaces these objects with mock objects. A **mock object** is an object with the same methods as the object it is replacing, but with no implementation behind these methods. Developers can specify the values that methods of mock objects return.

For my tests, I am using the testing libraries recommended by Google:

- **JUnit**, a unit testing framework for Java
- **Mockito**, a testing framework for JUnit that allows mocking objects

I am using unit tests mainly to test the communication between object based on the MVP model – that is, to ensure that the presenter is calling the view at the right times and vice versa.

4.2 Integration Testing

Integration tests allow testing certain parts of an application or the entire application as a whole. In these types of tests, the developer investigates how several classes work together as a whole.

I am using the library **Espresso** for my integration tests, using JUnit's AndroidJUnitRunner to run the tests. Espresso is an integration testing framework made specifically for Android and simulates user interaction either on device or in an emulator.

The integration tests that I am using click through the exercise and through the editor portions of the application, making sure components work as intended. This form of testing can be a bit limited or complicated to write, though, so manual testing is needed when it falls short.

4.3 Manual Testing

Where automated testing is too limited or too labor-intensive to write, manual testing comes in handy. The manual testing I did for this application ranged from logging certain variables and outputs to going through the application to make sure certain parts were working as expected.

4.4 User Testing

User tests help where automated tests cannot. They are used to make sure that the software is usable, that it lets the user accomplish the goals that it was designed to accomplish, and that the experience of using the software is pleasant.

4.4.1 Preparation

Given that we expected user tests for all modules to look very similar, Michal Bureš and I split up the preparation work thus: I prepared the materials for the testing of exercises by children and he prepared the materials for the testing of editors by teachers. Both of us produced the same types of materials, described below. These materials can be found in the addendum to this work.

In order to prepare for the testing, we made **introductory questionnaires** to understand the users' backgrounds. The intention was to have the test subjects fill out these questionnaires in advance and bring them to the test. In the case of children, these questionnaires would be filled out by parents, asking for the input of children when appropriate. Furthermore, scenarios were made to describe the testing process to the moderators, especially the tasks that the users were to complete. Finally, **post-test questionnaires** were written, meant to guide the post-test interviews that follow every test of a module.

4.4.2 Execution

Unfortunately, the user testing that was planned for this application with children and teachers from Smečno was postponed to a time after the publication of this thesis, yet before its presentation. The tests will be carried out, their results investigated, and changes will be made to ensure a better experience.

I have, however, conducted a user test with my 9-year old brother. He was able to finish every exercise quickly and successfully, except he didn't comprehend the point of the syllable length exercise, dragging words into boxes randomly until he finished the exercise.

4.4.3 Bug Fixes

Given that longer bodies of text can be off-putting to dyslexics, I was intent on using textual instructions only as a last resort. Instead, I tried using sound to indicate the syllable pattern. On testing this module again, with my brother still oblivious as to the point of the exercise, it was almost immediately obvious to my brother after hearing the sound cues that a syllable pattern indicated the long and short syllables in a word. He was then able to finish the exercise with a score of 100%.

Conclusion

As part of this work, I produced a set of modules for the learning application Dragon II targeted at children with learning disabilities. I wrote about our current knowledge of learning disabilities, zeroing in on the topics that the modules deal with. I went over ways of teaching tailored to them, discussed current exercises dealing with the same topics, and went over the design, implementation, and testing of the modules. I worked alongside a skilled team of three other members, one of whom worked on the application's core, one on the network side of things, and one on a different set of modules.

Goals

In this section, I will detail how my work has met the original goals set for this thesis.

Choose 5 problem areas that children with learning disabilities deal with. I chose the five areas described in the *Learning Disabilities* section under the *Analysis* chapter: reading an analog clock, recognizing syllable length, spelling, conceptualizing numbers, and understanding word problems.

Analyze strategies for teaching children with learning disabilities related to the chosen problem areas In describing the learning disabilities and problem areas in the *Analysis* chapter, I went over the pertinent strategies.

Analyze current applications and exercises dealing with these problem areas. Under the *Exercises* section of the *Analysis* chapter, I described the exercises and applications that target the chosen problem areas.

Design a module for each problem area. The *Design* chapter holds a thorough description of each module design.

Implement these modules. The modules were implemented based on this design. The source for these modules is attached to this thesis and will be made available on GitHub later. The *Implementation* section describes some aspects of the implementation.

Implement a complementary exercise creation module for each module, if appropriate. Every module was made with both an editor (for exercise creation) and an exercise. Details about the editor are present in both the *Design* and the *Implementation* section as well as in the attachments to this thesis.

Test each module, including usability testing. Unit, integration, and usability tests were all carried out, as described in the *Testing* section.

Future Work

The Dragon II application in general will continue to be developed. The core has yet to be implemented and more modules are being actively worked on.

As for the modules I worked on, they will be released under an open-source license and I will tweak and improve them based on the response they receive. My hope is that members of the community will work to translate the modules into more languages, use the libraries from the modules in their own projects and contribute to them, and that the modules can provide enough of a solid basis that some members of the community make new modules.

One potential area for improvement in all modules is graphics. More thought should be given to the graphical elements of each module, modules should be made livelier, and opportunities for animation should be investigated.

Another area that could be investigated further is data collection and analytics. Currently, my modules only record the overall time and score, but there are many other measurements that could be made.

I am convinced that Dragon II has a bright future. However, if it doesn't succeed in gaining a large audience, at least the open-sourced software may prove to be helpful to other developers.

Bibliography

[1] Android Developers: Android NDK. Available from: `https://developer.android.com/tools/sdk/ndk/index.html`, [Accessed: 2016-03-22].

[2] Bird, R.: The Secret to Reading an Analogue Clock. Available from: `https://www.youtube.com/watch?v=hZffJ-Dfs1I`.

[3] Blažková, R.: *Matematická cvičení pro dyskalkuliky : soubor ověřených pracovních listů pro práci se žáky s dyskalkulií na I. stupni ZS*. Stařec: Infra, 2013, ISBN 978-80-86666-44-0.

[4] Bláha, M.: *Výuková aplikace Dráček – Vývoj vybraných aplikačních zásuvných modulů*. Bachelor's thesis, Czech Technical University.

[5] Bláha, M.: Dragon I: Fill-in-the-blank Module. 2015.

[6] Bláha, M.: Dragon I: Syllable Module. 2015.

[7] British Council: Word stress. Available from: `https://www.teachingenglish.org.uk/article/word-stress`, Mar 2005.

[8] Children's Hospital Boston: "Sound Training Rewires Dyslexic Children's Brains For Reading". Available from: `https://www.sciencedaily.com/releases/2007/10/071030114055.htm`, Nov 2007.

[9] Emerson, J.; Babtie, P.: *The Dyscalculia Solution: Teaching number sense*. Bloomsbury Publishing, 2014, ISBN 9781472920997.

[10] Godsland, S.: Learing to Write and Spell - Spelling. Available from: `http://www.dyslexics.org.uk/spelling.htm`.

[11] International Dyslexia Association: Available from: `http://www.readingrockets.org/article/spelling-and-dyslexia`, 2009.

[12] JetBrains: Coding Conventions – Kotlin Programming Language. Available from: `https://kotlinlang.org/docs/reference/coding-conventions.htmll`, [Accessed: 2016-03-22].

[13] JetBrains: Documenting Kotlin Code – Kotlin Programming Language. Available from: `https://kotlinlang.org/docs/reference/kotlin-doc.html`, [Accessed: 2016-03-22].

[14] JetBrains: Kotlin: Reference. Available from: `http://kotlinlang.org/docs/reference/`.

[15] LD OnLine: Learning Disabilities: An Overview. Available from: `http://www.ldonline.org/article/5613`, [Accessed: 2016-03-09].

[16] Maccini, P.; Gagnon, J.: Mathematics Strategy Instruction (SI) for Middle School Students with Learning Disabilities. Available from: `http://www.ldonline.org/article/mathematics_strategy_instruction_(si)_for_middle_school_students_with_learning_disabilities`, 2006, [Accessed: 2016-03-22].

[17] Math Playground LLC: Thinking Blocks - Ratio and Proportion. Available from: `http://www.mathplayground.com/tb_ratios/thinking_blocks_ratios.html`, [Accessed: 2016-03-23].

[18] Minnesota STEM Research Center: Modeling Word Problems — SciMathMN. Available from: `http://scimathmn.org/stemtc/resources/mathematics-best-practices/modeling-word-problems`, [Accessed: 2016-03-23].

[19] NumberSense: time24. Available from: `http://number-sense.co.uk/time/`, [Accessed: 2016-03-23].

[20] Pavelec, P.: *Výuková aplikace Dráček II – Vývoj jádra aplikace pro žákovskou část.* Bachelor's thesis, Czech Technical University, 2016.

[21] PMQ Software: Český jazyk - pravopis. Available from: `https://play.google.com/store/apps/details?id=com.pmqsoftware.languagerules&hl=cs`.

[22] Rothmann, L.: My world without numbers. Available from: `http://tedxtalks.ted.com/video/My-world-without-numbers-Line-R`.

[23] Simon, H.: *Dyskalkulie : jak pomáhat dětem, které mají potíže s početními úlohami.* Praha: Portál, 2006, ISBN 80-7367-104-2.

[24] The International Dyslexia Association: Dysgraphia. Available from: `http://dyslexiasd.org/factsheets/dysgraphia.pdf`, 2000.

[25] Včelka: Včelka — výuka čtení a pomoc při dyslexii. Available from: `http://www.vcelka.cz/cs/`, [Accessed: 2016-03-09].

[26] Wood, T.: *Overcoming Dyslexia For Dummies*. Wiley, 2011, ISBN 9781118068519.

[27] Zelinková, O.: *Poruchy učení : dyslexie, dysgrafie, dysortografie, dyskalkulie, dyspraxie, ADHD*. Praha: Portál, 2003, ISBN 80-7178-800-7.

Appendix **A**

Acronyms

APK	Android application package
SDK	Software Development Kit
XML	Extensible markup language
JSON	JavaScript Object Notation
API	Application programming interface
IDE	Integrated development environment
CTU	Czech Technical University

Appendix B

Contents of enclosed CD

```
readme.txt .................. the file describing the contents of the CD
exe .................................... the directory with executables
src ..................................the directory of source codes
  impl ......................................implementation sources
  thesis .............the directory of LaTeX source codes of the thesis
text ........................................the thesis text directory
  BP_Mazel_Miroslav_2016.pdf ......... the thesis text in PDF format
  BP_Mazel_Miroslav_2016.ps ............ the thesis text in PS format
```