

Fredrik Jenssen

Magno: An Application for Detection of Dyslexia

From Native to Web App

Master's thesis in Computer Science

Supervisor: John Krogstie

June 2021

Fredrik Jenssen

Magno: An Application for Detection of Dyslexia

From Native to Web App

Master's thesis in Computer Science
Supervisor: John Krogstie
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science



Norwegian University of
Science and Technology

Abstract

Dyslexia is a reading disorder that affects many. The magnocellular system, responsible for motion detection in visual processing, appears to be deficient in people with dyslexia. A Java application called Magno is used to test an individual for motion detection deficiency. deCODE, a company studying genetics wants to use Magno to discover the genes associated with this deficit. This thesis reimplements Magno as a web application, aiming to increase the number of participants in deCODE's future studies by making it accessible in a browser. The thesis aims to answer which optimizations are necessary when porting a Java app to a web app and how changing from a facilitated to a self-managed test setting affects usability.

Usability testing was conducted on people from 24-81 years old and resulted in a System Usability Scale (SUS) score of 82,7. Previous usability testing resulted in a SUS score of 92,7, suggesting that a self-managed test setting negatively affects usability. Despite this decline, the current score is still considered excellent. Optimizations resulted in stable performance on laptops and tablets, but further adjustments are necessary to ensure high frame rates on weaker hardware. The web app needs to be validated by testing people with high and low reading competence before being employed in deCODE's studies.

Sammendrag

Dysleksi er en leseforstyrrelse som påvirker mange. Det magnocellulære systemet, ansvarlig for visuell bevegelsesoppfatning, synes svekket hos folk med dysleksi. En Java-applikasjon kalt Magno brukes til å teste individer for svekket bevegelsesoppfatning. deCODE, et selskap som forsker på genetikk, ønsker å bruke Magno for å oppdage genene assosiert med denne svekkelsen. Denne masteroppgaven implementerer Magno som en webapplikasjon for å gjøre appen lettere tilgjengelig, og slik øke antall deltakere i deCODE's fremtidige genstudier. Oppgaven forsøker å svare på hvilke optimaliseringer som er nødvendig når man omskriver en Java-app til en webapp, samt hvordan å gå fra en fasilitert til selvstyrt test påvirker brukervennlighet.

Brukervennlighetstesting ble utført på folk mellom 24 og 81 år og ga en System Usability Scale-poengsum på 82,7. Tidligere brukervennlighetstesting resulterte i en poengsum på 92,7 som antyder at et skifte til selvstyrte tester har en negativ effekt på brukervennlighet. En poengsum på 82,7 er likevel ansett som utmerket, til tross for poengsumnedgangen. Optimalisering resulterte i stabil ytelse på laptop og nettbrett, men videre justeringer er nødvendig for å sikre høy bildefrekvens på svakere maskinvare. Webappen må videre valideres ved testing på folk med høy og lav lesekompetanse før den kan tas i bruk av deCODE.

Preface

The work presented in this thesis is part of my master's degree in computer science at the Norwegian University of Science and Technology (NTNU). The project is conducted under the Department of Computer Science under the supervision of Professor J. Krogstie.

I would like to thank Professor John Krogstie for his support and advisory throughout this master's thesis. This thesis would not have been possible without his help. I would also offer my thanks to classmates and friends who have struggled alongside me and provided much-needed coffee breaks over this past year.

A big thanks to my family for their continued support, which has helped me see this master's degree through.

Fredrik Jenssen

June 23, 2021
Trondheim

Table of contents

List of Figures	i
List of Tables	iii
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Outline	2
2 Research Approach	3
2.1 Research Goals	3
2.2 Research Questions	3
2.3 Research Method	4
2.3.1 Evaluation and Data Analysis	4
3 Background	5
3.1 Defining Dyslexia	5
3.2 Consequences and the Importance of Early Detection	6
3.3 What Causes Dyslexia?	7
3.4 Defining Usability	9
4 Related Work	10
4.1 An App for Early Detection of Dyslexia	10
4.1.1 First Iteration: Functionality	10
4.1.2 Second Iteration: Usability	14
4.1.3 Third Iteration: Data Storage	17
4.2 Web RDK	22
4.3 Svipgerð: deCODE's Test Platform	22
4.4 Requirements	23
4.4.1 Out of Scope	27
5 Methods, Tools and Technology	28
5.1 Scrum	28
5.2 Usability Testing	28
5.2.1 Remote Usability Testing	28
5.2.2 The SUS Form	29
5.3 Tools and Technology	30
6 Implementation and Usability Testing	34
6.1 Workflow	34
6.2 Software Architecture	35
6.3 Design Choices	36
6.4 First Iteration	36
6.4.1 Application Overview	36
6.4.2 Performance Optimization	45
6.4.3 Usability Testing	46
6.4.4 Written Feedback and Proposed Changes	48

6.5	Second Iteration	49
6.5.1	Application Changes	49
6.5.2	Usability Testing	51
6.5.3	Written Feedback	53
6.6	Classes and Interfaces	54
7	Evaluation	57
7.1	Usability Testing	57
7.2	Requirements Fulfillment	59
7.2.1	Previous Requirements	59
7.2.2	New Requirements	62
8	Discussion, Conclusion and Further Work	64
8.1	Discussion	64
8.2	Conclusion	65
8.3	Further Work	66
	References	68
	Appendix	73
A	Previous requirements for Magno	73
B	Default Setting Values	75
C	Example Test Results	77
C.1	Motion Test Results Example	77
C.2	Form Test Results Example	78
D	SUS Form	80
E	Usability Test Email	80
F	First Usability Test: Free Text Answers	81
G	Second Usability Test: Free Text Answers	83

List of Figures

1	Overview of the research process. From Oates, 2006 [1].	4
2	Primary and secondary consequences of dyslexia [2].	6
3	Illustration of a random-dot kinematogram. (a) with coherently moving dots (b) only randomly moving dots	9
4	(a), (b): Random-dot kinematograms with coherent motion in (a) and random motion in (b). (c), (d): Form test with random pattern in (c) concentric circles pattern in (d).	10
5	Main menu of Magno’s first version.	11
6	The motion test.	11
7	Form fixed test 100% coherency.	12
8	Form fixed test at 50% coherency.	12
9	Form random test 100% coherency.	13
10	Form random test at 50% coherency.	13
11	Form fixed test in auto and manual mode.	14
12	Home screen, navigation bar to the left.	15
13	Test results.	15
14	First screen of the tutorial.	16
15	Enter age screen.	16
16	Software architecture of the third iteration. Server, database, and research website are hosted in Azure Cloud. MagnoClient refers to the motion and form tests running on the tester’s device.	17
17	Landing page.	18
18	Login page.	18
19	Data view.	19
20	Contact page.	19
21	UML class diagram of the third iteration showing classes and their relationships [3]	21
22	Components in deCODE’s test system.	22
23	Svipgerð with the motion, form fixed and form random tests added as new modules.	23
24	The modified SUS form with preliminary and optional questions added. See the original SUS form in Appendix D.	30
25	Screenshot of the Trello board.	34
26	Example of branching and commits from the project’s Git repository. In this particular example, <i>performance-testing</i> is branched out from <i>master</i> but never merged since it was not intended to be part of the application.	35
27	Introduction screen.	37
28	Tutorial introduction screen.	38
29	Tutorial task screen.	38
30	Tutorial trial screen.	39
31	The various states of the tutorial trial.	40
32	Test screen.	41
33	The various states of the motion test.	42
34	Result screen.	43
35	Loading screen.	44

36	Mobile screen.	44
37	Initial dot placement with straight lined grid and sine wave grid. . . .	46
38	Boxplot of SUS scores.	47
39	Participant distributions across age, gender and device.	47
40	Average score per SUS question.	48
41	Introduction screen.	50
42	Tutorial task screen.	50
43	Boxplot of SUS scores.	52
44	Participant distributions across age, gender and device.	52
45	Average score per SUS question.	53
46	UML class diagram of the final web application.	55
47	Box plot of SUS scores from both iterations' usability tests.	57
48	The average SUS scores per age group.	58
49	Recorded performance from a complete run-through of the motion and form fixed tests on an ASUS TUF gaming laptop using Brave Browser.	60
50	Recorded performance from a complete run-through of the motion test on a MacBook Air, Ipad Pro, and Samsung Tab A.	61
51	Original SUS form.	80

List of Tables

1	Requirements brought over from previous iterations of Magno. See Appendix A for all previous requirements, including those not brought over.	24
2	New requirements for the web application	26
3	Quality attributes and associated non-functional requirements.	26
4	The proposed changes spawned from user feedback.	49
5	Requirements brought over from previous iterations of Magno. See Appendix A for all previous requirements, including those not brought over.	59
6	New requirements for the web application	63
7	Requirements from the first iteration of Magno [4].	73
8	Requirements from the second iteration of Magno [5].	74
9	Requirements from the third iteration of Magno [3].	75
10	All textual feedback received from the first usability test. The motion test was tested.	83
11	All textual feedback received from the second usability test. The form fixed test was tested.	84

1 Introduction

Developmental dyslexia henceforth referred to as dyslexia, is a hereditary reading disorder that affects between 5-10% of the Norwegian population [6]. It comes with several negative consequences for those affected and the society as a whole. A person with dyslexia can experience poor academic results [7], stigmatisation at school and in work, as well as reduced emotional well-being and self-esteem [8] [9].

However, these consequences can be mitigated. A positive diagnosis of dyslexia gives people with dyslexia a label to attribute their differences in reading and learning ability. Without the diagnosis, these people might attribute their differences to a lack of intelligence. Multiple studies thus stress the importance of early detection to reduce negative consequences and aid in the development of a positive self-image [2] [8].

1.1 Motivation

In Norway, specific language difficulties, which may include early signs of dyslexia, can be investigated at the age of 4. However, a dyslexia diagnosis investigation must start in the 2. grade of primary school [6]. The investigation involves tests relying on reading proficiency, putting specific learning measures in place, and then retaking the tests. The schools could benefit from a screening tool to begin investigation sooner and take preventive measures even before reading ability is acquired.

Magno, a screening tool built for this purpose [4], is based on the magnocellular theory of visual processing and shows promising results [10] [11]. deCODE, an Icelandic company studying genetics, eyed an opportunity to use Magno to investigate dyslexia's hereditary causes. They wish to distribute Magno's tests to around 15000 people and cross-analyze test results with their genes to search for the genetic cause of impaired visual processing.

However, Magno is a native application for Android devices and desktop PCs, requiring users to download and install it before the tests can be taken. deCODE's experience with similar tests shows that this process acts as a barrier, resulting in fewer participants [12]. The work presented in this thesis aims to minimize this barrier by rebuilding Magno as a web application to make it accessible directly in the browser. As a web app, Magno can both increase study participants and become a more accessible dyslexia screening tool.

1.2 Thesis Outline

Section 2 will detail my research approach, with specific research questions and the research method applied. Section 3 will outline previous work in the area of dyslexia, discussing consequences, definitions, and causes, as well as defining usability. Section 4 details Magno's development from the project's beginning to its current state and concludes with requirements for this project's web solution, divided into functional and non-functional requirements. Section 5 outlines development- and usability test methods and briefly describes the tools and technology used. Section 6 outlines the workflow, design choices, software architecture, application overviews, and results from usability testing. Section 7 discusses the results from usability testing and evaluates the solution concerning functional and non-functional requirements. Finally, Section 8 discusses test results, concludes by answering the research questions, and proposes further work for the project.

2 Research Approach

This section presents the chosen research approach. In Section 2.1 the research goals are presented. Section 2.2 details the research questions sprung out of the research goals. Section 2.3 outlines the research method applied with regards to strategies, data generation, and data analysis. Section 2.3.1 further delves into how analysis and evaluation of the solution are performed.

2.1 Research Goals

The overarching goal of this project is to allow for easy distribution of Magno's motion and form tests to a multitude of people of varying reading competence, allowing deCODE to progress research on dyslexia's hereditary component and institutions to more easily use it as a screening tool.

With the overarching goal in mind, this thesis's practical goal is to reimplement the motion, form fixed, and form random tests from the Magno application as a web solution. The web solution must use technology that makes it easy for deCODE to install it as a module in their online test platform Svipgerð, or Phenomenon, in English. The new solution should be on par with the Java app's usability and provide similar test results for people with high- and low reading competence.

2.2 Research Questions

A critical difference between the new solution and the old is the test-setting. Whereas the old solution was built around having an instructor facilitate the test, the new solution will be self-managed. The test taker will now access the tests in a web browser on their private PC or tablet device. In order to function well in this self-managed setting, we chose to approach the reimplementation through the following research questions:

RQ-1 How is app performance maintained when porting a Java app to JavaScript?

RQ-2 How can Magno's user interface support a self-managed test setting?

RQ-2.1 What changes to Magno's tutorial instructions are needed to sufficiently prepare the test taker?

RQ-3 How does a self-managed test setting affect the usability of the system?

RQ-4 How can motion test results be validated in a self-managed test setting?

2.3 Research Method

The research will follow Oates' model on the research process [1]. An overview of my research approach is shown in Figure 1.

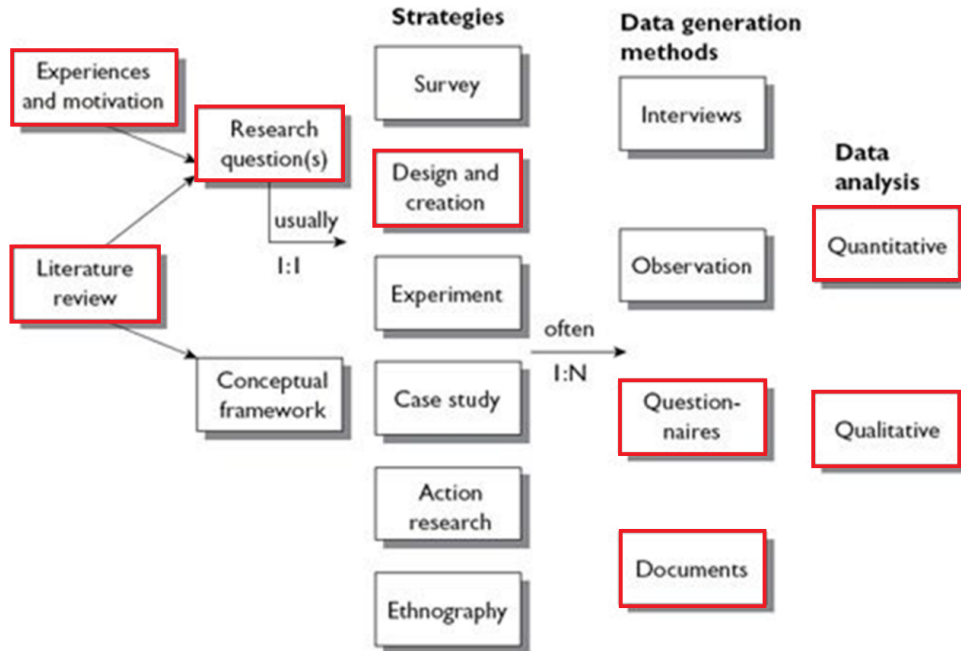


Figure 1: Overview of the research process. From Oates, 2006 [1].

The literature review and implementation plan derived in the specialization project provided the basis for the research goal and research questions mentioned above. The main research strategy will be design and creation, producing new IT artifacts in the form of the new web solution. The data to be analyzed will be twofold; from questionnaires sent to users who participate in usability testing of the web app and the application itself, including code structure, performance data, and design.

2.3.1 Evaluation and Data Analysis

The web app will undergo a qualitative and quantitative analysis during and at the end of development. Usability testing will be conducted on adults of different ages, ranging from 18 to 75. The first usability test will consist of test subjects completing the motion test before being asked to fill out a System Usability Scale (SUS) form [13]. The second test will have the same structure but replaces the motion test for the form fixed test. In addition, the form will contain two additional questions intended to identify specific issues related to the design and usability of the application. Finally, the web app will be evaluated with regards to the functional and non-functional requirements outlined in 4.4.

3 Background

This section defines dyslexia, its consequences on an individual and broader level, and what causes it. The section concludes by defining usability.

3.1 Defining Dyslexia

There are many definitions of dyslexia, and defining it continues to be a topic of discussion. What makes defining it tricky is its several manifestations. A person with dyslexia may experience one or more of several symptoms related to dyslexia. Pedagogues and special pedagogues tend to use practical definitions like those defined by the International Dyslexia Association (IDA), British Dyslexia Association (BDA) or the ROSE-report [14]. Here are the definitions:

IDA defines it as:

Dyslexia is a specific learning disability that is neurobiological in origin. It is characterized by difficulties with accurate and/or fluent word recognition and by poor spelling and decoding abilities. These difficulties typically result from a deficit in the phonological component of language that is often unexpected in relation to other cognitive abilities and the provision of effective classroom instruction. Secondary consequences may include problems in reading comprehension and reduced reading experience that can impede the growth of vocabulary and background knowledge. [15]

The ROSE report's definition:

Dyslexia is a learning difficulty that primarily affects the skills involved in accurate and fluent word reading and spelling. Characteristic features of dyslexia are difficulties in phonological awareness, verbal memory, and verbal processing speed. Dyslexia occurs across the range of intellectual abilities. It is best thought of as a continuum, not a distinct category, and there are no clear cut-off points. Co-occurring difficulties may be seen in aspects of language, motor coordination, mental calculation, concentration, and personal organization, but these are not, by themselves, markers of dyslexia. A good indication of the severity and persistence of dyslexic difficulties can be gained by examining how the individual responds or has responded to well-founded intervention. [16]

BDA adopted ROSE's definition in 2010, however extending it with the following characteristics:

The British Dyslexia Association (BDA) acknowledges the visual and auditory processing difficulties that some individuals with dyslexia can experience and points out that dyslexic readers can show a combination of abilities and difficulties that affect the learning process. Some also have strengths in other areas, such as design, problem-solving, creative skills, interactive skills, and oral skills. [17]

Notably, BDA’s is the only definition to acknowledge the visual and auditory processing difficulties in dyslexics, as will be discussed as possible causes of dyslexia in Section 3.3. A notable difference between ROSE’s and IDA’s definition is ROSE’s emphasis on the importance of seeing dyslexia as a continuum and examining the effect of interventions to assess the degree of severity. Another difference is IDA’s mention of secondary consequences, such as an impediment to vocabulary growth. What the definitions agree on is that it causes difficulties with accurate and fluent reading and spelling.

3.2 Consequences and the Importance of Early Detection

Dyslexia can have serious consequences for the individual, family, and society. A study looking at the effects associated with dyslexia reviewed 97 articles on the subject from 1980 to 2018, categorizing consequences as primary or secondary as shown in Figure 2.

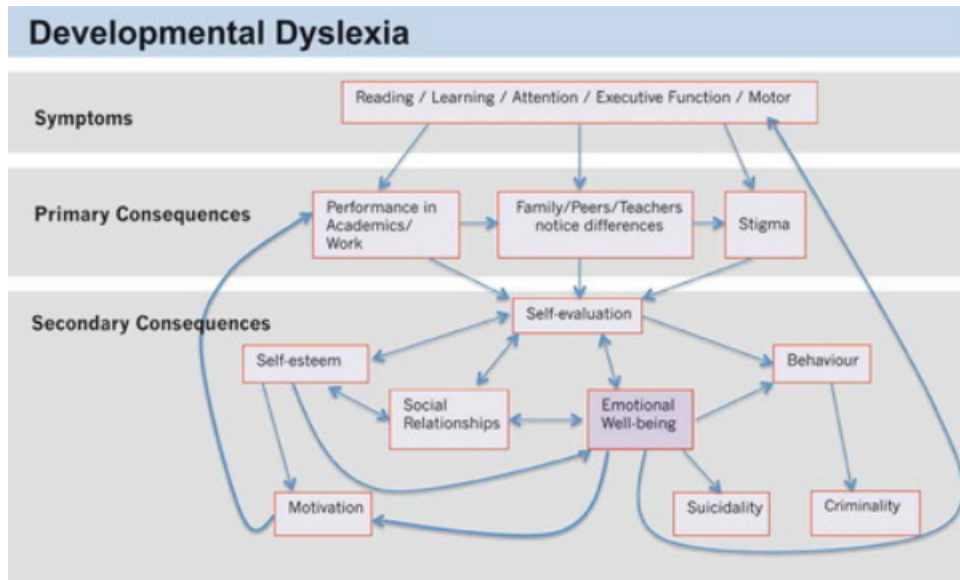


Figure 2: Primary and secondary consequences of dyslexia [2].

The primary consequences can be poor performance in academics and work and differences noticed by peers, teachers, and parents coming with real or perceived stigma. Secondary consequences often start with negative self-evaluation due to primary consequences, which can negatively impact self-esteem, social relationships, motivation, and emotional well-being. The review also stresses that primary and secondary consequences are intertwined. It concludes by highlighting the importance of early, high-quality assessment and diagnosis of dyslexia to minimize the impact of these consequences.

Another study of dyslexia’s consequences interviewed children with dyslexia attending secondary school in the UK who told of dyslexia’s impact on their self-evaluation. Several of the pupils interviewed reported feeling stupid and isolated before diagnosis. After diagnosis, many no longer felt this way and said it had helped to know

others were sharing their problems. They also noted that the dyslexic label helped them understand why they were struggling. The study concludes that an official diagnosis is crucial for developing self-esteem.

A follow-up study of 29 adults with dyslexia in Germany supports the notion that early diagnosis and preventive measures can mitigate consequences. The study interviewed the adults 20 years after graduating from dyslexia boarding schools and found increased spelling skills, high occupational status, and no significant psychiatric issues since attending school [18]. This change can, in part, be attributed to the above-average IQ of the subjects and the high socioeconomic status of their parents. However, the study also notes the importance of remedial work at their schools.

These studies show evidence to support the notion that early detection and preventive measures play a crucial role in dyslexics' development. *Dysleksi Norge*, the advocacy group for dyslexics in Norway, also stresses the importance of early detection. They refer to a study which shows that the effectiveness of preventive measures decreases from 80% in the 1. and 2. grades of elementary school to just 10-15% in the 5. grade [6].

3.3 What Causes Dyslexia?

Reading is a complex function involving many cognitive processes, which are reduced in people with dyslexia. Research has so far identified these processes to be verbal short-term memory, phonological awareness, rapid naming, phonological and orthographic coding, and visual processing [19]. As visual processing is an obvious reading component, I will give a short description of the other cognitive processes.

Verbal short-term memory refers to the ability to process unknown words into their phonemes in order to read them. Known words are often recalled directly from memory. This ability is often examined through a digit span task [20].

Phonological awareness is the ability to perceive, segment, and manipulate the sounds of spoken words. Phonemes are the smallest unit of meaningful sound in a language, which combined form words. Phonological awareness is often tested through a phoneme deletion task, which removes a sound from a word and asks the test taker to pronounce the altered word.

Rapid naming is a measure of speed processing, typically involving the naming of numbers, letters, colors, and objects.

Phonological coding refers to being able to put phonemes together to pronounce unknown words and is usually tested by asking the test taker to pronounce made-up words.

Orthographic coding is the assumed ability to recognize a word by its holistic form and is tested by presenting a word orally, then displaying two phonologically indistinguishable words of which only one is correctly spelled.

The Phonological Theory

The theory garnering the most attention is the phonological theory, which suggests that difficulties with reading stem from a failure to acquire the skill of separating sounds into phonemes that match the letters that represent them [21]. The most utilized test of this skill is to ask participants to read made-up pronounceable words. This test is utilized because it requires letter-to-sound conversion without any help from meaning or context, making it difficult for poor readers to pronounce.

However, this theory is criticized for its applicability to all poor readers, not just dyslexics [21]. John Stein argues that the essence of reading is decoding - translating letters into their corresponding sounds - which the phonological theory reiterates without providing evidence for why people with dyslexia fail to learn it. Furthermore, many dyslexics do not exhibit phonological problems, meaning it is not necessarily a precondition for dyslexia [22]. The argument thus makes the case that poor decoding skills are insufficient to distinguish dyslexics from those who struggle to read due to other causes, such as poor teaching and lack of family support [21].

The Magnocellular Theory

Another theory called the magnocellular (M) theory links dyslexia to how visual information is processed in the brain. M-neurons are part of the visual system and are found in the retina, lateral geniculate nucleus (LGN), visual cortex, and in the parietal and frontal cortices [21]. These neurons specialize in rapid temporal processing, such as a change in luminance, localization of objects, and motion detection. The cells dominate the dorsal pathway from the visual cortex, seemingly responsible for directing the ventral stream's direction to the word it should analyze at a given moment [23]. In other words, the M-dorsal pathway seems to play a vital role in the allocation of attention for reading.

Evidence has accumulated showing an impaired M-system in people with dyslexia. MRI and histological studies of dyslexics' brains have found abnormalities in the M-layers of the LGN. Livingstone et al. found these layers to be 30% smaller and more disorganized than in those of the control group [24]. Giraldo-Chica et al. found the layers to be significantly thinner than in age-matched control groups [25]. Furthermore, Eden et al. used functional magnetic resonance imaging (fMRI) to find that activity in the visual motion area, supplied mainly by the M-system, in response to moving stimulus is much reduced [26].

Further evidence of an impaired M-system in dyslexics has been found by testing visual motion sensitivity through psychophysical measures. Psychophysics is the scientific study of the relationship between stimulus, and sensation [27]. Random dot kinematograms, as shown in Figure 3, is a psychophysical way of measuring motion sensitivity [28]. They consist of numerous dots where a portion moves randomly, and the other portion moves in the same direction. The number of coherently moving dots required to detect the motion determines their sensitivity to motion. Several studies have found that dyslexics need a significantly greater proportion of coherently moving dots to detect the motion [29] [30] [11].

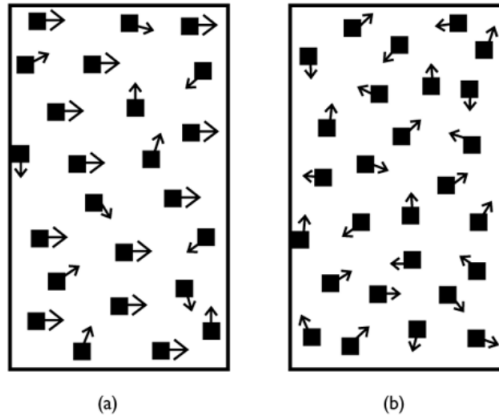


Figure 3: Illustration of a random-dot kinematogram. (a) with coherently moving dots (b) only randomly moving dots

Visual M-weakness is not detected in every person with dyslexia. Furthermore, many children with impaired M-function learn to read well. Thus, an impaired M-system cannot be the sole cause of dyslexia [31]. However, on average, it is shown to be significantly impaired in dyslexics. This provides an opportunity for early detection tools by testing the M-system’s sensitivity, at the benefit of being helpful to those who cannot read.

3.4 Defining Usability

An important aspect of this thesis is the focus on maintaining usability in a self-managed setting. Thus it is valuable to define what usability is. The ISO 9241-11 standardized a definition for usability and its measurement. This standard points out that usability depends on context, where something usable is appropriate to its context. The context consists of the task being done, the experience and background of the user doing it, and the environment in which it is performed. [13] The ISO standard further breaks down usability measurement into three categories:

- Effectiveness - how well the user is able to perform their tasks and achieve their goals
- Efficiency - the amount of resources used to achieve their goals
- Satisfaction - the level of comfort experienced when performing tasks and achieving goals

4 Related Work

This section will cover Magno’s conception, its several iterations since the project’s start, and an alternative random-dot kinematogram web implementation. Further, it will present an overview of deCODE’s test platform, how Magno’s tests will fit as modules in it, and present the requirements specification that serves as a basis for the web solution.

4.1 An App for Early Detection of Dyslexia

4.1.1 First Iteration: Functionality

The Magno project’s initial goal was to re-create an MS-DOS program from the early 2000s called *Form*. This program was used to check for vision problems and showed promise as a screening tool for dyslexia [32]. As a program made for older hardware, it was desirable to bring it to modern-day devices. Modernizing this program would be the basis of master’s student Bjørnar Wold’s thesis. He re-created *Form* as a Java application during the school year of 2015-2016. The tests included a motion detection test based on random-dot kinematograms and two form tests based on static global patterns. An illustration of these tests are seen in Figure 4.

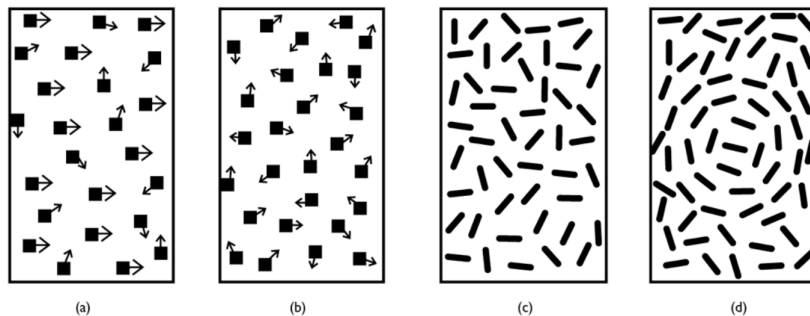


Figure 4: (a), (b): Random-dot kinematograms with coherent motion in (a) and random motion in (b). (c), (d): Form test with random pattern in (c) concentric circles pattern in (d).

The project’s goal was to implement the functionality of these tests as close to the old MS-DOS program as possible by interviewing people that partook in development or utilized the program previously. A snapshot of the program’s main menu is seen in Figure 5. The application was implemented in Java using a game development framework called libGDX [33].

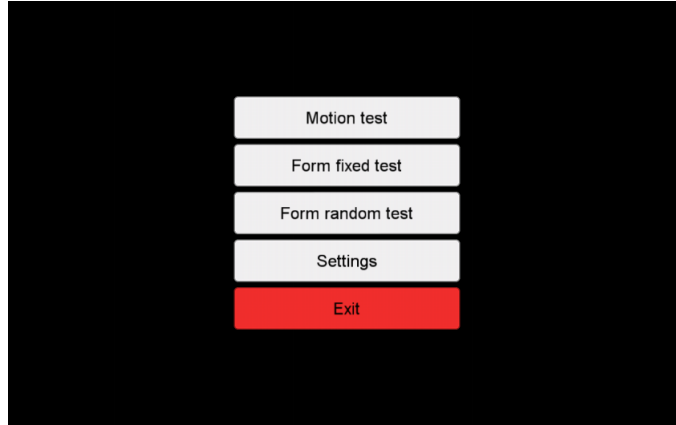
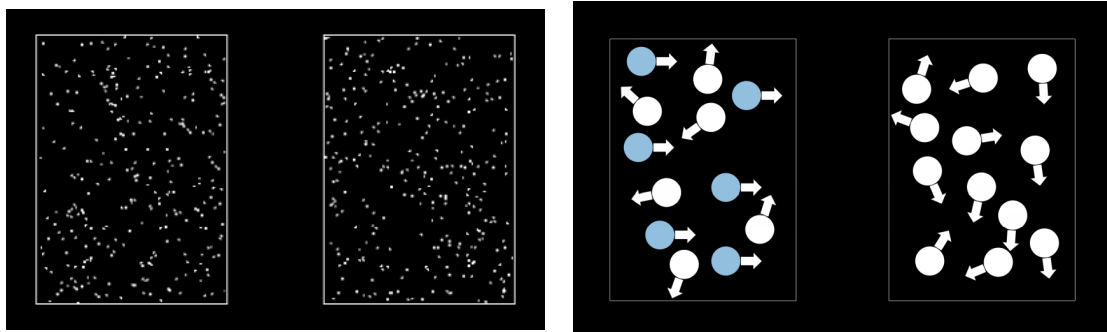


Figure 5: Main menu of Magno's first version.

As illustrated, the application consisted of three tests: a motion, form fixed, and form random test. The app also provided a settings view for tweaking test parameters. Following is a short description of each test's workings with their default settings.



(a) Motion test screen capture.

(b) Motion test illustration at 50% coherency.

Figure 6: The motion test.

The motion test consists of two patches with 300 randomly placed dots with a radius of 1 pixel. At the beginning of each test trial, a patch is chosen randomly to contain the coherent-moving dots while the other contains only random-moving dots. The coherent moving dots will move either left or right, reversing every 0,572 seconds. The randomly moving dots change direction every 0,572 seconds or when colliding with other dots.

The test participant detects the patch with coherent moving dots during the 5 seconds of animation time. After 5 seconds, the dots disappear, and the participant selects the patch he believes to contain the coherent moving dots. On selection, a new trial with increased or decreased level of coherency begins. Dots are given new locations and movement directions. The coherency level decreases if the correct patch is chosen and increases otherwise, making the next trial harder or easier. Figure 6b illustrates the test at a 50% coherency threshold.

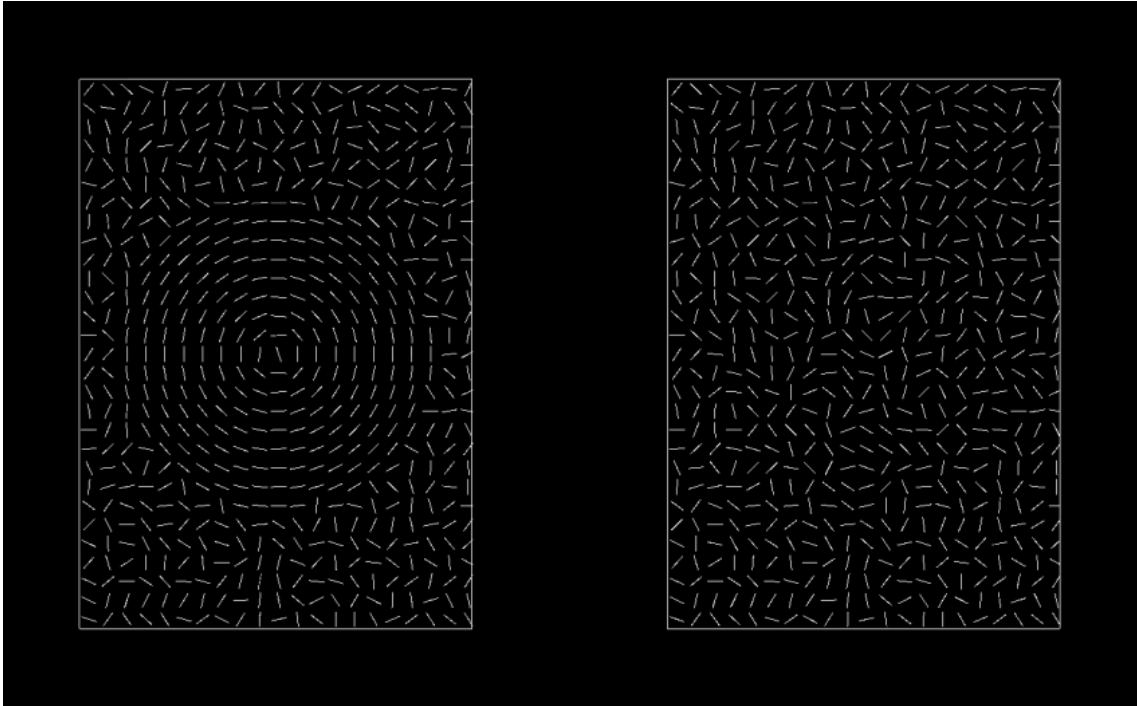


Figure 7: Form fixed test 100% coherency.

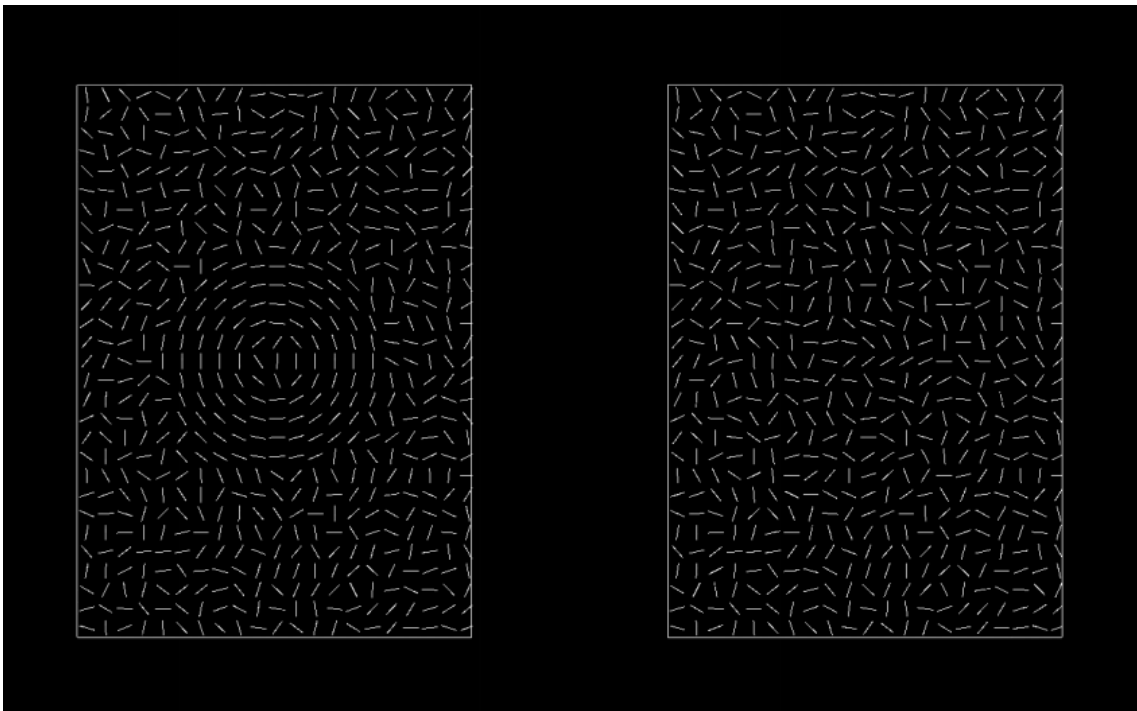


Figure 8: Form fixed test at 50% coherency.

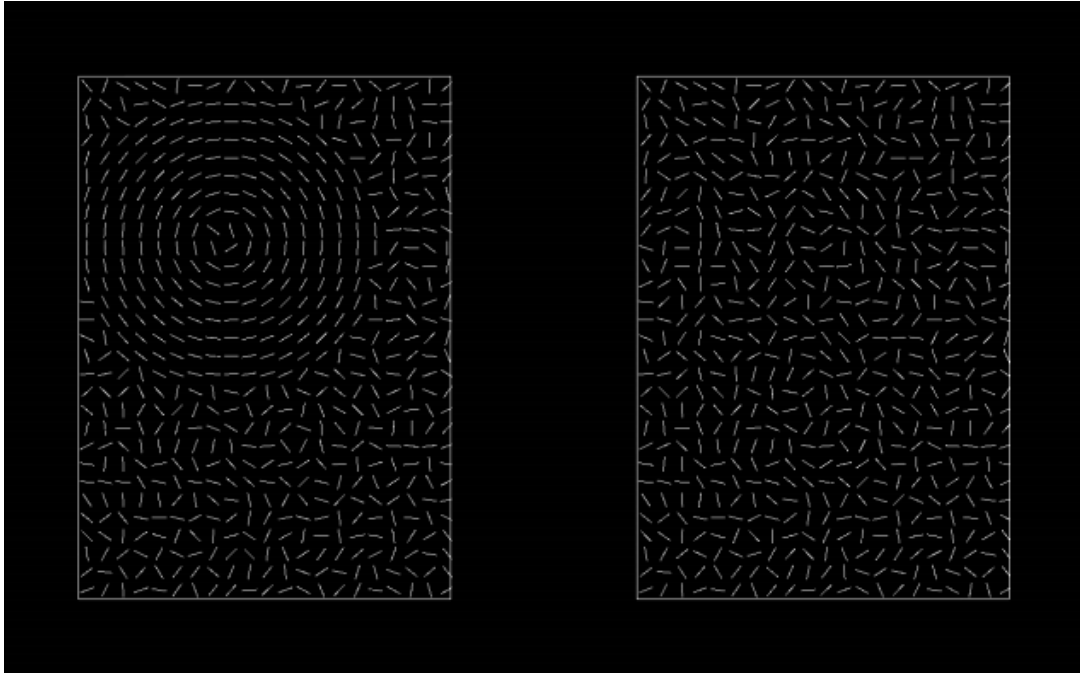


Figure 9: Form random test 100% coherency.

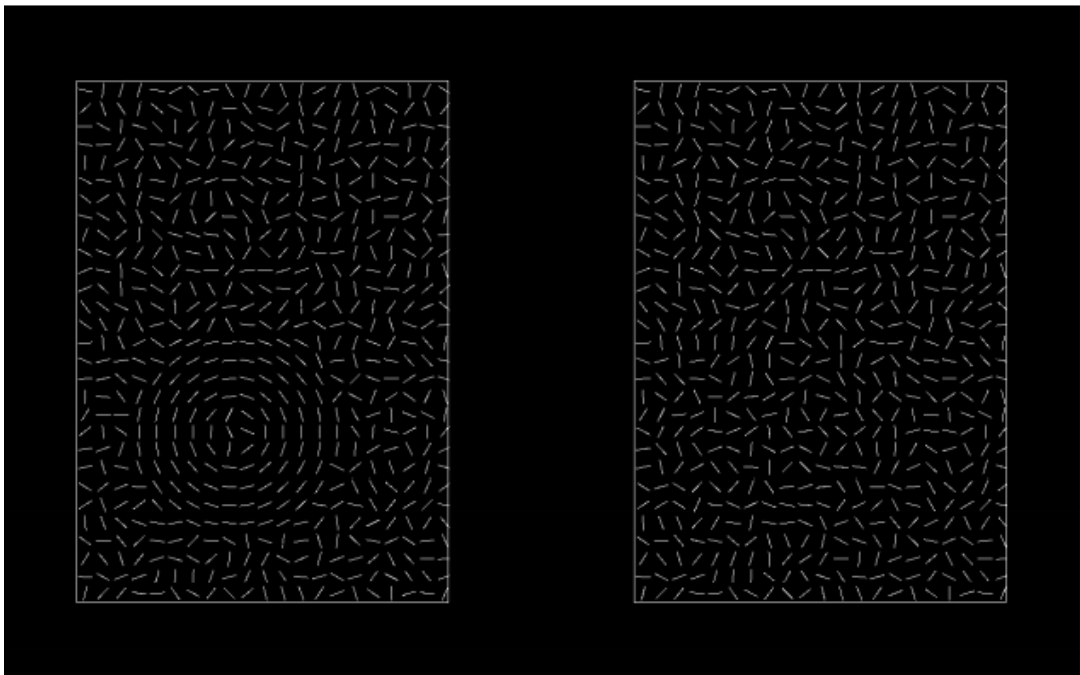


Figure 10: Form random test at 50% coherency.

The form tests, as seen in Figure 7 and 9, consist of 600 line segments with each of the lines having a length of $0,4^\circ$ of the viewing angle and a height of 1 pixel. The lines are distributed evenly. One of the patches is chosen at random to contain a percentage of lines that form concentric circles. For the form fixed test, the center of the concentric circle is always in the middle of the patch. In the form random test, the center moves to a random position on the patch for each trial.

The test participant has 4 seconds to search for the patch containing the concentric rings. The participant then selects the patch they believe contains the rings. The pattern is recalculated with an increased or decreased coherency, similar to the motion test. Figure 8 and Figure 7 show the form fixed test and Figure 10 and Figure 9 show the form random test at 100% and 50% coherency thresholds. The form tests can be run in two modes; auto mode, which distributes line segments evenly as in Figure 11a, and manual mode, which distributes the line segments randomly as in Figure 11b.

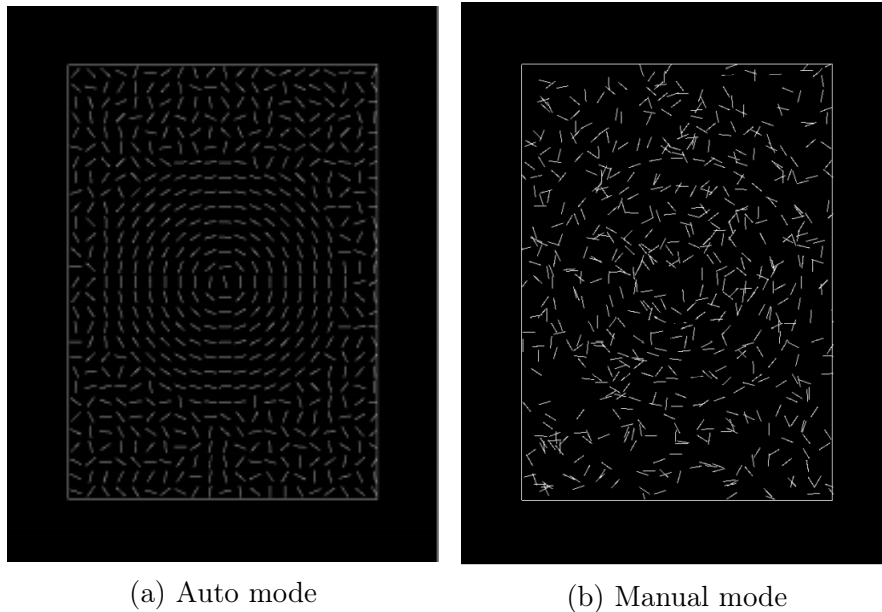


Figure 11: Form fixed test in auto and manual mode.

4.1.2 Second Iteration: Usability

Thea Johansen and Maja Kirkerød made the second iteration of Magno in their master thesis during 2016-2017 [3]. It focused mainly on upgrading the app's usability, with a big overhaul of the user interface. This iteration added a navigation bar for easily accessing the different parts of the application, the ability to enter your age for later statistical analysis, tutorials for enhancing self-management, and a test-results screen showing how well one performed. Screenshots of these can be seen in Figure 12, 13, 14, and 15.



Figure 12: Home screen, navigation bar to the left.

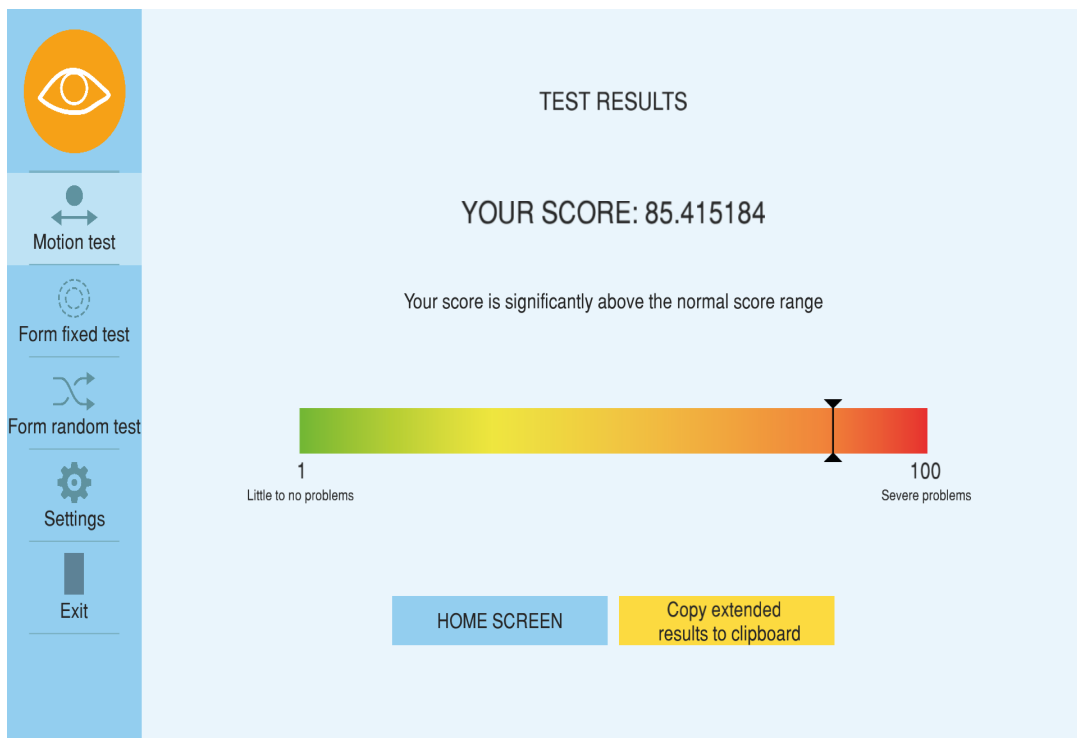


Figure 13: Test results.

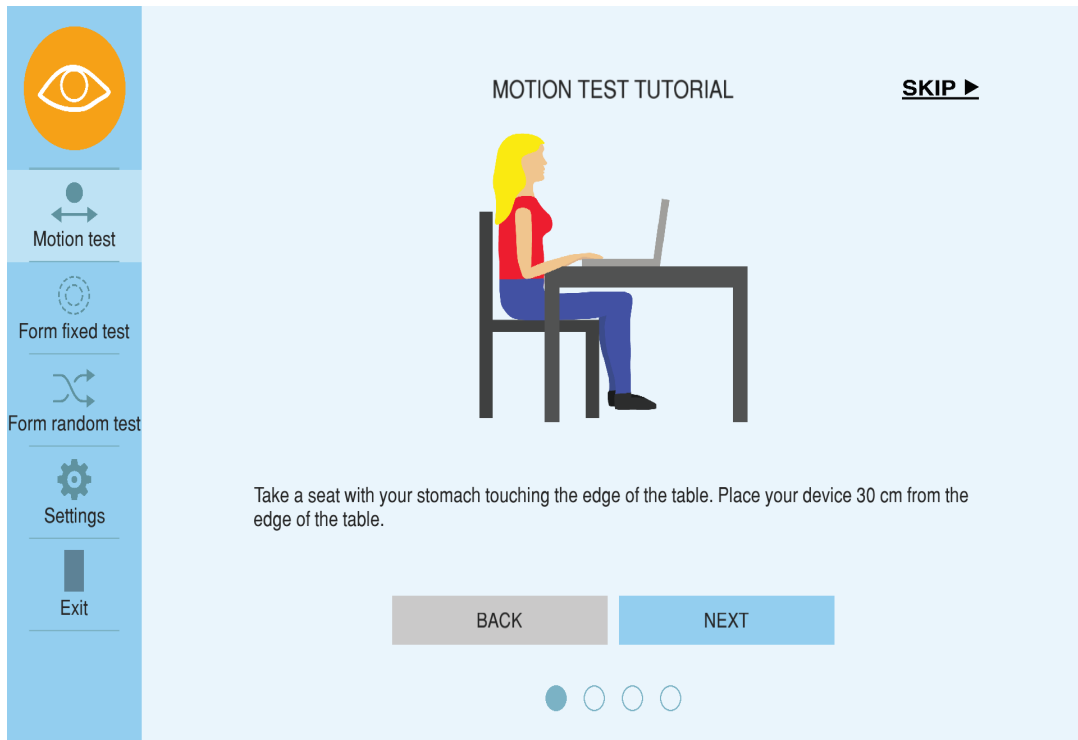


Figure 14: First screen of the tutorial.

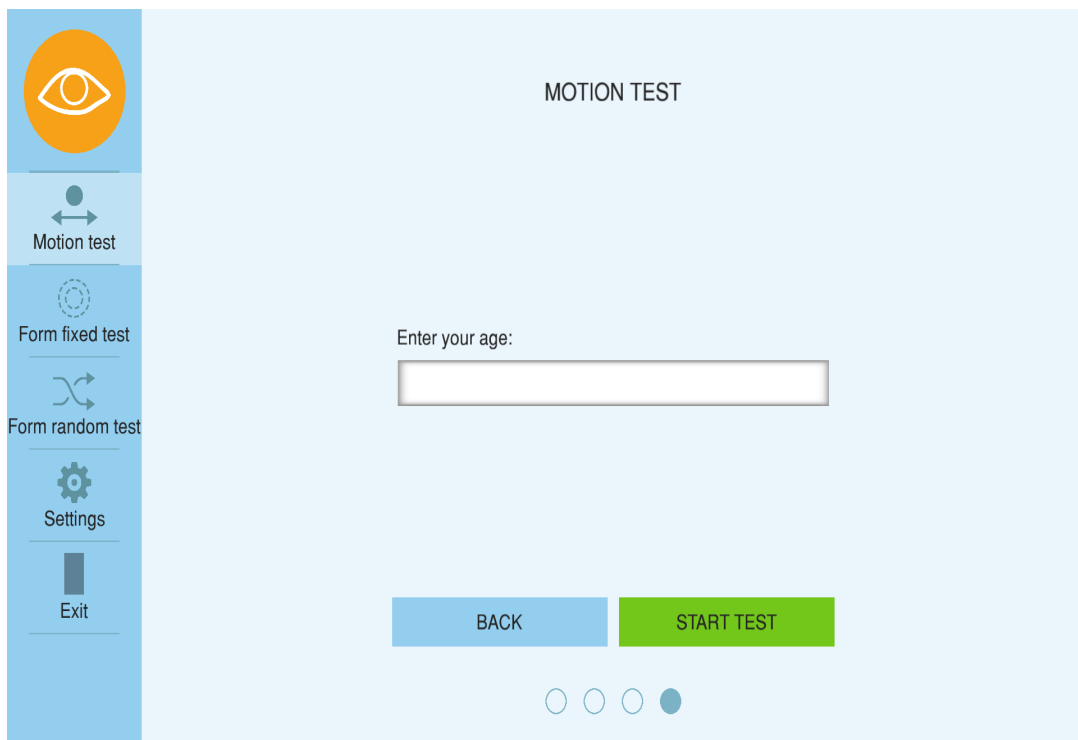


Figure 15: Enter age screen.

Johansen and Kirkerød made several dyslexia-friendly design choices. They chose a blue-themed user interface based on studies showing that reading text on a blue background can help keep letters still on a page. Another design choice made was to

use the Helvetica font, shown to be well suited for readers with dyslexia [34]. Their digital prototype underwent usability testing with 10 students of ages 18-25 from NTNU Gløshaugen. The prototype received a SUS score of 92,7 using a slightly altered version of the SUS form that omitted its first question.

4.1.3 Third Iteration: Data Storage

The following year, Tore Angell Petersen continued work on Magno, focusing on collecting test results and providing an interface for researchers to analyze these results quickly. His project contributions were an SQL database for storing results, a server with an API for receiving and saving results, and a website for researchers to gather and view test results. The system architecture is shown in Figure 16.

MagnoOnline

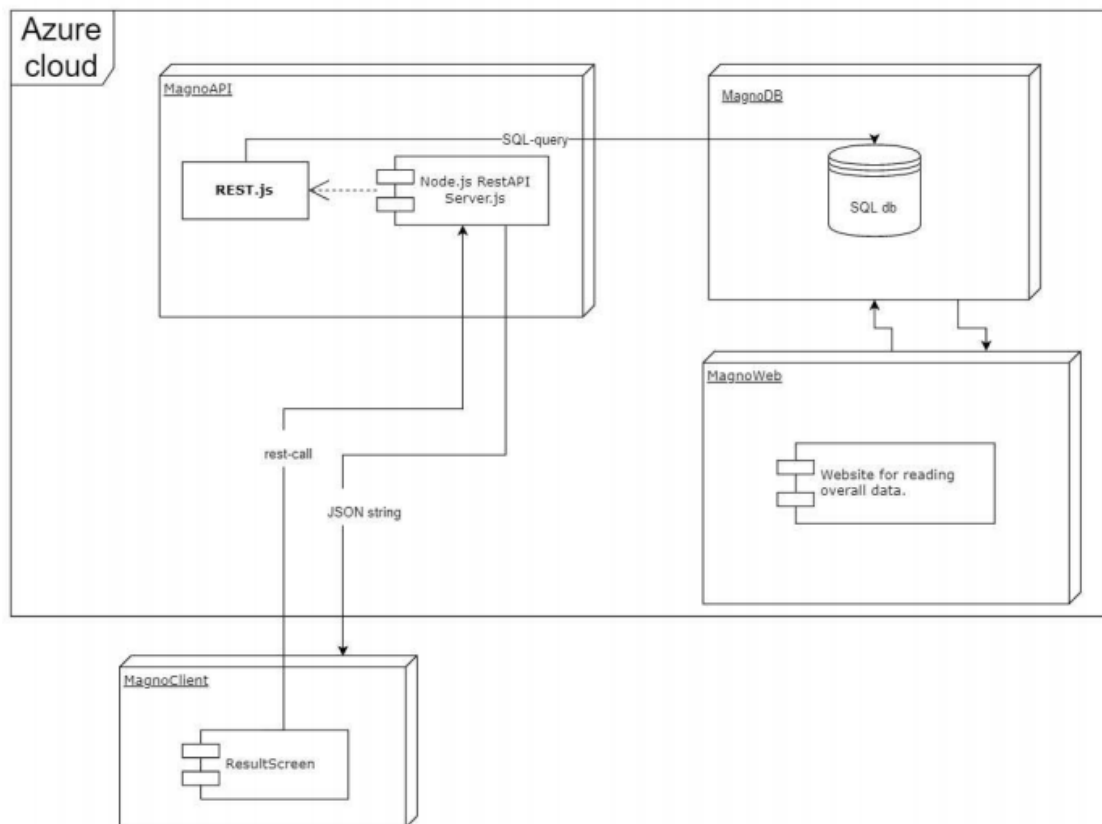


Figure 16: Software architecture of the third iteration. Server, database, and research website are hosted in Azure Cloud. MagnoClient refers to the motion and form tests running on the tester’s device.

The MagnoClient in the figure refers to the Java application with the motion, form fixed, and form random tests. Here Angell added a ServerConnection class that connects to the MagnoAPI backend server. The server is based on REST principles and includes a route for receiving results and sending the mean test score per age group to the client. The server runs in a Node.js environment and is hosted in Microsoft’s Azure Cloud alongside an SQL database and a website for viewing test results. The MagnoWeb website is built with Microsoft’s Asp.Net framework.

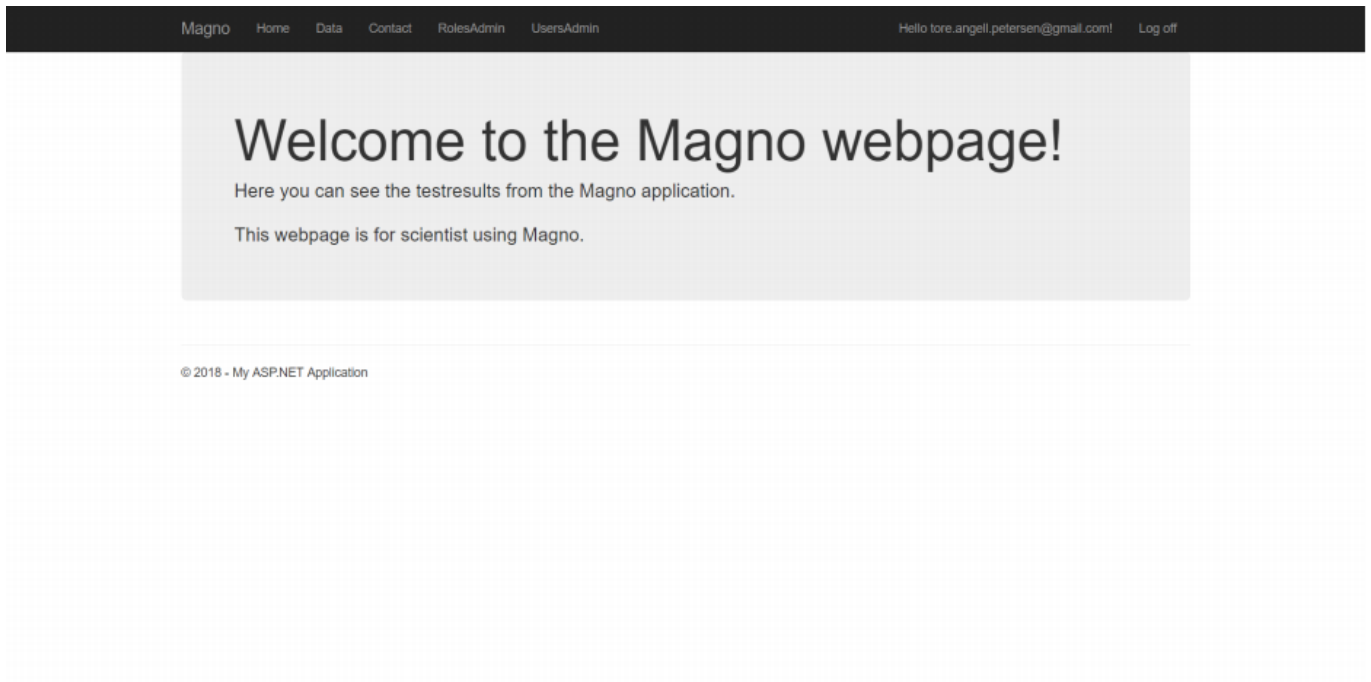


Figure 17: Landing page.

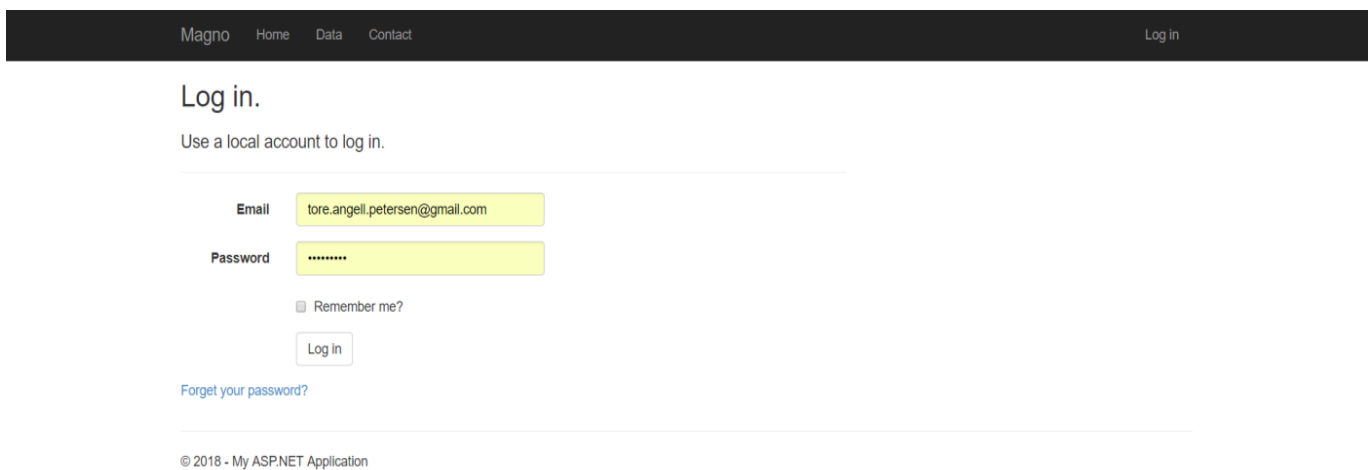


Figure 18: Login page.

Data.

Data information page.



© 2018 - My ASP.NET Application

Figure 19: Data view.

Contact.

Contact page.

One Microsoft Way

Support: [Tore Angell Petersen](#)

© 2018 - My ASP.NET Application

Figure 20: Contact page.

Figures 17-20 show the various screens found on MagnoWeb, including a home page, login page, data view and contact page. The system administrator must authorize a user account before the test results can be accessed. The data view lets you filter results within an age group. The site's contact page is for reporting any errors with the site.

Figure 21 shows the class diagram for the latest iteration Magno. Each square represents a Java class, and a line indicates a dependency between two classes. The yellow boxes represent classes that were altered for this iteration, and the red box is the additional ServerConnection class.

4.2 Web RDK

A research group from the University of California implemented the first open-access random-dot kinematograms for web browsers [35]. Their solution was integrated as a plugin in jsPsych, an open licensed JavaScript library for creating and running behavioral experiments in the browser [36]. It provides much of the same configurations as Magno, with additional options for dot behavior, number of patches, background color, and patch shapes. However, web RDK does not support Brownian motion¹, which is the dot behavior used in Magno.

4.3 Svipgerð: deCODE’s Test Platform

deCODE has built an application platform for distributing tests called *Svipgerð* [37]. Currently, it hosts three modules: *Tóneyra og taktvísi*, a screening test for tone and rhythm blindness, *Persónuleiki*, a personality assessment of the five most-studied personality traits, and *Psoriasis*, a questionnaire with health-related questions for people with psoriasis.

Technology

Svipgerð and the tests residing in it are based on AngularJS v1.7.9 with Bootstrap v4.4.1. deCODE’s backend consists of an Express.js server. The server is responsible for the API that handles HTTP requests from clients and connects to the database to retrieve and insert data. This data typically consists of test results along with user information. deCODE runs an Oracle database for handling persistent storage of these data. A visualization of the system’s components can be seen in Figure 22.

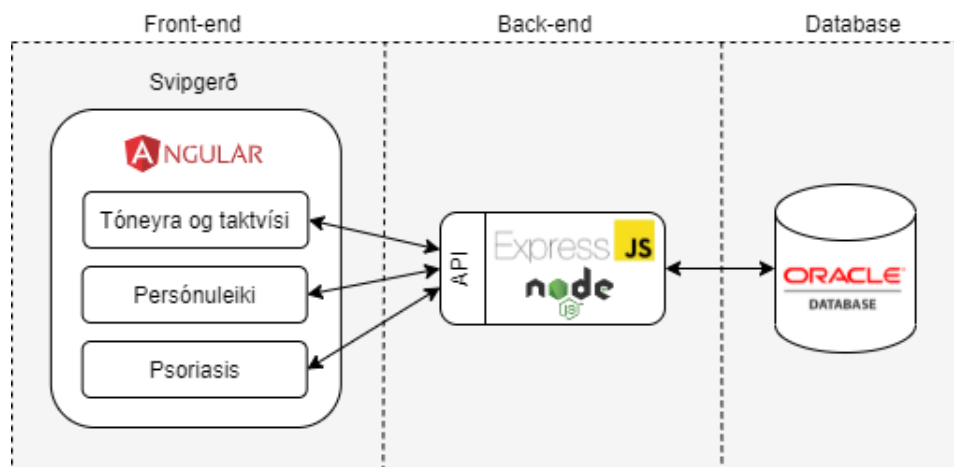


Figure 22: Components in deCODE’s test system.

¹Random movement of particles in a fluid

Backend

The system is based on the client-server architecture, which uses a server to manage and distribute resources requested by the client. The server is built with Express.js, a web framework based on Node.js. This architecture makes it possible to distribute incoming requests to multiple node processes for increased performance and scalability. Additionally, Node.js supports effective load balancing through libraries such as *node-HTTP-proxy*², which manages workload across multiple node servers running on separate machines.

Svipgerð Integration

deCODE has requested that the motion, form fixed, and form random tests be implemented as independent applications so that each can be taken separately [38]. This independence entails each test residing as its own module in Svipgerð. As modules in Svipgerð, the solution must be compatible with AngularJS. Figure 23 illustrates how the tests will fit into Svipgerð.

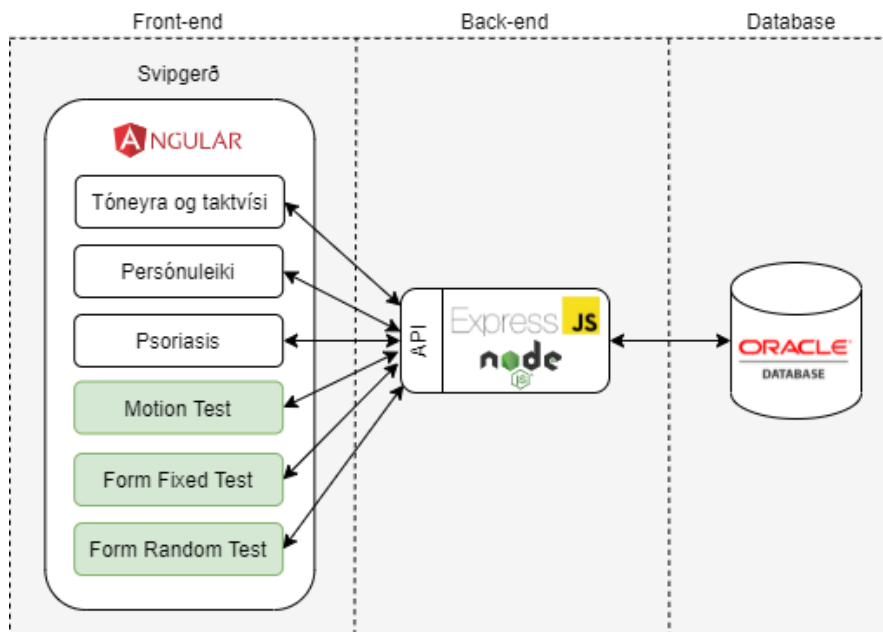


Figure 23: Svipgerð with the motion, form fixed and form random tests added as new modules.

4.4 Requirements

We examined previous iterations of Magno to identify existing functionalities and qualities that are fundamental to the application. Dialogue with deCODE resulted in complementary specifications to better support a self-managed test setting. The requirements brought over from previous iterations are in Table 1, with some given

²<https://github.com/http-party/node-http-proxy>

a slight rewording. The new requirements are in Table 2. The tables use the abbreviations FR for functional requirements and NFR for non-functional requirements.

Id	Description	Priority
FR1	The application should implement the motion test using random-dot kinematograms and the form tests using static global patterns as described in Section 4.1.1.	High
FR2	The application must use screen size, screen resolution, and viewing distance to calculate the size of objects on the screen.	High
FR3	Default settings parameter-values are to be provided. See Appendix B for a comprehensive list of these values.	High
FR4	The application should calculate a threshold score after a successful test run of the motion, form fixed or form random test.	High
FR5	The test subject should be prompted to complete a tutorial before the test.	High
FR6	Threshold score and the settings used during a test should be saved in a test results object.	High
FR7	It should be possible to exit the test at any time.	High
FR8	The application should use screen size to make the design responsive to different sized tablet and computer screens.	High
NFR1	The application should have a minimum SUS score of 80.	High
NFR2	The application should run close to 60 frames per second (FPS) on desktop PCs and tablet devices.	High

Table 1: Requirements brought over from previous iterations of Magno. See Appendix A for all previous requirements, including those not brought over.

Id	Description	Priority
FR9*	When accessed on a computer, the cursor should change form when hovering over clickable content.	High
FR10	If accessed on a mobile device, it should not be possible to take any of Magno's tests.	High
FR11	If accessed on a mobile device, Magno should display a message telling the user it is unavailable for mobile devices and propose using a tablet or desktop PC.	Medium
FR12	The motion, form fixed, and form random tests should be implemented as separate applications so that they can run independently of each other.	High
FR13	Test settings should be kept in a configuration file on the server, one for each test.	High
FR14	A simple HTTP server should be created to test that POST requests containing test results are properly received and acts as documentation for deCODE's team to see which API endpoints are used.	Medium
FR15	The app shall use an HTTP client to send POST requests with test results to an HTTP server.	High
FR16	In addition to the data already included in the motion- and form test results, see Appendix C, they should also include the test type: either motion, form fixed, or form random, as well as specifying auto or manual mode for the form tests. In addition, each coherency threshold step should include: <ul style="list-style-type: none"> • How long the user took to click a patch. • The absolute pixel coordinates (x, y) of where the user clicked on the screen. • The current coherency threshold. • Which key was pressed in case of keyboard use • The patch selected • The target patch 	High
FR17*	The app should facilitate multi-language support.	Medium
NFR3	Settings should be adjustable only by deCODE's system administrators.	High

*Unattained requirements from the previous iterations.

NFR4	The test should be designed with self-management in mind, such that a user is able to understand the application quickly and complete the tests without difficulty.	High
NFR5	The application should be based on JavaScript to easily integrate as an application in deCODE's Svipgerð platform and ease continued development.	High
NFR6	The app should cater to the technical abilities of those in the age range of 18 to 75.	High

Table 2: New requirements for the web application

Examining the NFRs from Table 1 and Table 2 we can use the software quality model of the ISO 25010 standard to identify key quality attributes of the system: usability, performance efficiency, and maintainability. Table 3 shows the link between NFRs and quality attributes. The quality attributes summarize aspects in focus when developing the application.

Quality Attribute	Id	Description
Usability	NFR1	The application should have a minimum SUS score of 80.
	NFR4	The test should be designed with self-management in mind, such that a user can understand the application quickly and complete the tests without difficulty.
	NFR6	The app should cater to the technical abilities of those in the age range of 18 to 75.
Performance Efficiency	NFR2	The application should be able to run close to 60 FPS on desktop PCs and tablet devices.
Maintainability	NFR3	Settings should be adjustable only by deCODE's system administrators.
	NFR5	The application should be based on JavaScript to easily integrate as an application in deCODE's Svipgerð platform and for easing continued development.

Table 3: Quality attributes and associated non-functional requirements.

ISO 25010 defines usability similarly to the definition in Section 3.4. Maintainability and performance efficiency is defined as the following [39]:

- *Performance Efficiency* represents the performance relative to the number of resources used under stated conditions. In this case, resource use must be kept at a level where the device is able to run close to 60 FPS.
- *Maintainability* represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it, or adapt it to changes in the environment and in requirements.

4.4.1 Out of Scope

Equally important to describe what the application will do is describing what it will not. As the web solution will reside in deCODE's Svipgerð platform, some functionality is deemed out of scope for this project. Functionality that is already handled by deCODE, or left for them to address, includes:

- Authorizing users through login capabilities.
- Handling multiple test-runs per user.
- Giving feedback on test scores relative to age group.
- Storing test results in a database.
- Creating and maintaining a server with an API for storing and retrieving results from a database.

5 Methods, Tools and Technology

This section outlines the development method used, how usability is conducted, and which tools and technology are used.

5.1 Scrum

Parts of Scrum will be used as the central development methodology for this project. Scrum is an agile process framework, heavily used within project management and software development [40]. The parts of Scrum we will utilize are the emphasis on stakeholder involvement and iterative development. The framework is based around teams of five to ten members but can also be applied successfully with smaller teams. Scrum breaks work down into manageable pieces, which can be completed in time-constrained iterations, called sprints. Sprints are usually around two weeks to a month-long. This project will be divided into two sprints. The first sprint's goal is to complete an initial version of the application and test its usability. The second sprint aims to improve the application based on feedback from the first usability test and then perform another usability test to see if the changes improved usability.

5.2 Usability Testing

To verify that the solution's usability is on par with the current solution, it is necessary to scrutinize the app through usability testing. However, the ongoing COVID-19 pandemic imposes limitations on how usability testing can be performed. Notably, the situation prohibits the option of being physically present as a supervisor to instruct and observe test takers. With a typical field study requiring physical presence out of the question, usability testing will be performed remotely.

5.2.1 Remote Usability Testing

Remote usability testing consists of two main approaches: remote synchronous usability testing (RSUT) and remote asynchronous usability testing (RAUT). RSUT refers to real-time testing where the tester and evaluator are only separated spatially. In RAUT, the tester and evaluator are separated both in time and space. A study found RSUT to be virtually equivalent to conventional laboratory-based think-aloud tests [41], but at the cost of being time-consuming as the facilitator has to be present throughout the test for every tester. An example of a synchronous setup is to use a live-video chat service such as Zoom or Skype to conduct testing like one would in a lab-based setting.

RAUT has been shown to identify around half of the usability problems of traditional methods but also requires significantly less time to perform [42]. As the facilitator does not have to be present, these tests can enable more participants,

leading to a larger quantity of results for analysis. Teston³ is a popular software providing RAUT services such as screen and voice recordings. However, analyzing video and voice recordings is a time-consuming process, so for this project, we will have participants complete the test and fill out a questionnaire afterward. Besides time constraints, RAUT is also used because it better reflects the intended user setting: testers completing the tests individually.

5.2.2 The SUS Form

A popular questionnaire for usability testing is the System Usability Scale (SUS) form. SUS was first created in the mid-80s by John Brooke as a quick way to test the perceived usability of a computer system. It was made freely available in '86 and has gained much attention since, being cited in over 1,200 publications [13]. Sauro [43] collected data of 5,000 SUS observations and analyzed it extensively, making SUS an excellent tool for comparing usability to other software products. Some key takeaways from his summary of the SUS form are:

- It is reliable. Users respond consistently to the questions, and the form is shown to detect differences at smaller sample sizes than other questionnaires.
- It is valid, measuring what it intends to measure.

SUS is comprised of ten questions concerning usability. The questions are answered with a number from 1 to 5, where 1 corresponds to strong disagreement and 5 to strong agreement. The odd questions are positively worded, while the even questions are negatively worded. The questionnaire is structured this way to avoid response biases, such as marking only the extremes of the scale or agreeing to all items in the questionnaire. The final SUS score is calculated in the following manner:

1. $X = \text{Sum of the points for all odd-numbered questions} - 5$
2. $Y = 25 - \text{Sum of the points for all even-numbered questions}$
3. $\text{SUS Score} = (X + Y) \times 2.5$

Subtracting 1 from each odd question makes the minimum score applied when strongly disagreeing equal to 0 and 4 when strongly agreeing. The same applies to Y , only in reverse. When strongly agreeing to the even-numbered questions, we want to give the minimum score of 0. Likewise, we want to award the highest score of 4 when strongly disagreeing with the same questions. Finally, we multiply by 2,5 to scale the score to be between 0 and 100. Initially, this scaling was a marketing strategy to grab the attention of stakeholders more familiar with a 0 to 100 scale than a 10 to 50 scale. Brooke notes that this scaling has the downside of the score being misinterpreted as a percentage. However, as SUS became increasingly popular, this was the scale that stuck.

The questionnaire I will employ is a modified SUS form. The first question is changed from *"I think I would like to use this system frequently"*, to *"I think I would like to use this system if needed"*. This change is due to Magno not being intended for multiple

³<https://teston.io/no/>

uses. Three questions are added to the start of the form, asking the participant's age, gender, and device to gain insight of the participants' demographics and to analyze those using tablets and PCs separately. Additionally, two optional free text questions are added to enable more detailed and diagnostic feedback, which can be used to identify concrete examples of poor usability to improve upon in the second iteration. The complete form is as follows:

1. Please enter your age.
2. Please select your gender.
3. Which device did you use when taking the test?
4. I think I would like to use this system if needed.
5. I found the system unnecessarily complex.
6. I thought the system was easy to use.
7. I think that I would need the support of a technical person to be able to use this system.
8. I found the various functions in this system were well integrated.
9. I thought there was too much inconsistency in this system.
10. I would imagine that most people would learn to use this system very quickly.
11. I found the system very cumbersome to use.
12. I felt very confident using the system.
13. I needed to learn a lot of things before I could get going with this system.
14. Was there anything in the application that was particularly difficult to understand?
15. Do you have any comments on the design of the application?

Figure 24: The modified SUS form with preliminary and optional questions added. See the original SUS form in Appendix D.

Tullis and Stetson have shown that a sample size of 12-14 is needed to get reliable test results. [44]. We will thus aim to recruit 12 or more participants for each usability test in this thesis.

5.3 Tools and Technology

This section provides an introduction to the relevant tools and technologies used in this project.

TypeScript

TypeScript is a programming language which transpiles to JavaScript. It provides additional features to JavaScript like object-oriented programming concepts such as

inheritance and classes, strong static typing and pre-compilation error detection. As TypeScript transpiles to JavaScript, it is also easily compatible with AngularJS [45] [46] [47].

PixiJS

PixiJS is an open-source, free-to-use JavaScript library with fast 2D rendering using WebGL. It also supports fallback to Canvas for browsers that do not support WebGL. The speed of PixiJS will help keep the frame rate at a steady 60 frames per second when rendering the motion test, the most computationally heavy part of the application. The library supports a variety of sprite sheet formats, with multiple functions for transforming sprites. This makes it possible to reuse the skin of the Java application. Another useful feature is its asset loader, which caches sprites and graphics for quick access if multiple objects use the same sprite [48].

Babel

Babel is a JavaScript compiler that allows writing ES6 syntax without worrying about compatibility with older browsers and environments that do not support the newest JavaScript syntax. ES6 syntax enables features such as classes and arrow functions, making it easier to re-implement Java classes from Magno and keeping the code compact and easy to read [49].

Parcel

Parcel is a Web Application Bundler that offers a multitude of features with no configuration [50]. Its main features are:

- Assets bundling (JavaScript, HTML, CSS, images)
- Zero configuration code splitting (shortening load times)
- Automatic transforms using Babel, PostCSS and PostHTML
- Hot module replacement, updating the content in your browser as you work, allowing you to see changes instantly.
- Caching and parallel processing for faster builds.

Node.js

Node.js is a JavaScript run-time environment for server- and web applications. It executes JavaScript code on Google's V8 engine, allowing JavaScript programs to run on servers. This will be useful to create the server code for handling POST-requests of test result data and keep a consistent data model. It also has a built-in package called node-fetch, which acts as an API to the native fetch API. Fetch will be used to post test results to the server. Node.js also includes the Node Package Manager (NPM), which keeps track of module dependencies and versions. NPM provides a registry with free JavaScript packages for solving a multitude of problems and installs them seamlessly by running a simple NPM install command [51] [52].

i18next

i18next is an internationalization framework for JavaScript [53]. It lets you add multi-language support to your application by handling issues such as detecting user language, and loading and caching translations.

Git

Git is a distributed version control system that enables software collaboration, and code-change tracking [54]. It allows working on different features on separate branches while maintaining a functioning master branch, providing a workflow that makes life as a developer easier. This project will use it to maintain an organized code structure.

GitHub

GitHub is a website offering cloud-based Git repository hosting [55]. This makes collaborating on software projects easy, as every developer has access to the entire codebase and its history through the GitHub repository. Among other services provided by GitHub are access control, feature requests, bug tracking, task management, continuous integration, wikis, and web hosting.

GitHub Pages

GitHub Pages is a service offered by GitHub which provides static web hosting for rapid and easy deployment. It lets you publish your GitHub projects simply by enabling it in your project settings and pushing your latest changes to master. This project will use the NPM package *gh-pages* to deploy by running an NPM script instead of pushing to master [56] [57].

Visual Studio Code

Visual Studio Code is one of the most popular integrated development environments featuring syntax highlighting for hundreds of languages, debugging, code completion, refactoring, Git integration, and community-made extensions for added functionality. Especially relevant to the project is the built-in support for Node.js development. It is also known for its simplicity, speed, and a high degree of customization [58].

Trello

Trello is a cloud-based list-making application [59]. It is ideal for organizing workflow and keeping track of task progression. Trello is free, focused on visual communication, highly customizable, and available directly in the browser. It will be used for project management.

Nettskjema

A solution for quickly creating digital forms and collecting answers was needed to conduct usability testing. *Nettskjema* is a website for creating forms developed by the University of Oslo, available to students and employees in higher educational

institutions around Norway. It provides an intuitive interface for quickly generating universally designed forms, secure data handling, and complies with GDPR. Form responses are easily exported to Excel files for further data analysis [60].

6 Implementation and Usability Testing

This section details the development process and provides an overview of the application screens with changes made from the Java app and the results of both iterations' usability tests.

6.1 Workflow

The Trello Board

Trello was used for task management, and the board was split into four lists. The *Backlog* with proposed tasks, the *Sprint Backlog* with remaining tasks yet to be started for this sprint, *In Progress* for tasks being worked on, and *Complete* for finished tasks. Each task was seen as a feature or a bugfix comprised of several subtasks. Once all subtasks of a feature task were completed, it was moved to the *Complete* list. Figure 25 is a screen capture of the Trello board in its final state.

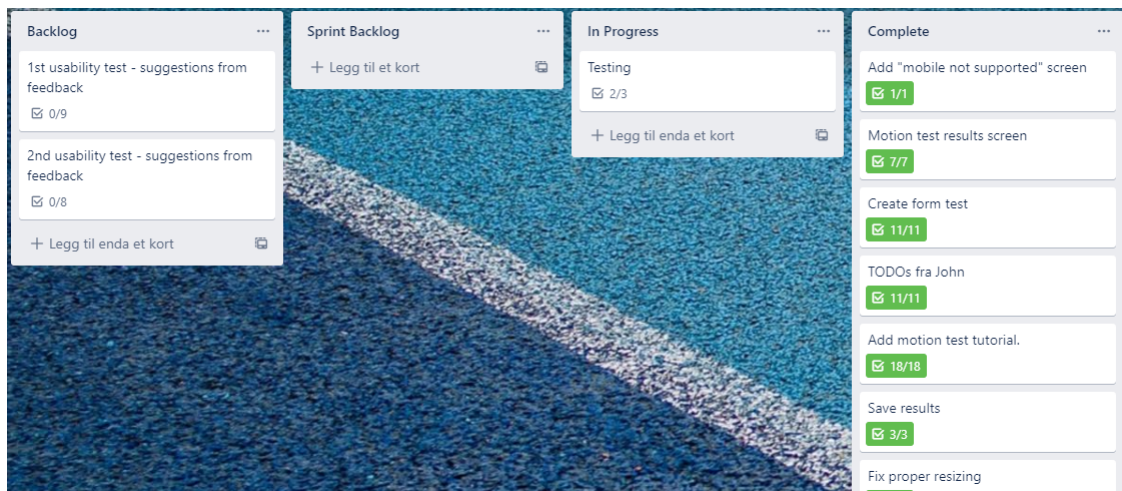


Figure 25: Screenshot of the Trello board.

GitHub Branching and Deployment

Each task was worked on in its own feature branch to separate coding into manageable workloads and generally to avoid creating a mess. The feature branches were named after the convention *feature/{feature_name}*. Using feature branches and committing changes often made for an incremental workflow.


Graph	Description	Date	Author
	<code>master</code> <code>origin</code> <code>origin/HEAD</code> Adjusts spacing of elements in ...	24 May 2021 14:03	Fredrik Jenssen
	<code>performance-testing</code> Move console log to end of function	22 May 2021 15:47	Fredrik Jenssen
	Merge branch 'master' into performance-testing	22 May 2021 14:13	Fredrik Jenssen
	<code>feature/send-results-to-server</code> <code>origin</code> Adds a basic http server and met...	20 May 2021 17:33	Fredrik Jenssen
	Small fix. Adds touch handler for exit button and re-adds handlers on resize	18 May 2021 18:24	Fredrik Jenssen
	Updates readme and removes jest config and sample test	15 May 2021 16:47	Fredrik Jenssen
	minor refactoring	14 May 2021 16:29	Fredrik Jenssen
	removes return statement and comment	14 May 2021 16:03	Fredrik Jenssen
	<code>feature/multi-language-support</code> <code>origin</code> Adds multi-language support a...	13 May 2021 16:15	Fredrik Jenssen
	Fixes typo in script names	10 May 2021 11:38	Fredrik Jenssen
	Changes the text in LandingPageScreen, TutorialTaskScreen and TutorialTri...	5 May 2021 13:03	Fredrik Jenssen
	Removes unused destroyDots() function. Moves getRandomPosition() to u...	5 May 2021 13:01	Fredrik Jenssen
	Adds tutorial settings to the test results object. Changes the lowest score ...	5 May 2021 12:56	Fredrik Jenssen
	Saves fps and screen change to json and prints to console when changing ...	4 May 2021 14:40	Fredrik Jenssen
	Fixes bug for form-random where line segments would disappear when w...	29 Apr 2021 16:56	Fredrik Jenssen

Figure 26: Example of branching and commits from the project’s Git repository. In this particular example, *performance-testing* is branched out from *master* but never merged since it was not intended to be part of the application.

When a feature was finished, it was merged into the master branch and hosted on GitHub Pages to verify that the production build functioned properly for different screen sizes. When the second iteration was completed, the main repository was copied to enable usability testing of both the motion and form fixed tests in parallel.

6.2 Software Architecture

The Model-View-Controller and PixiJS

The Java application employs the model-view-controller (MVC) pattern to clearly separate the test logic and user interface. This provides the developer with a high level of control, but at the cost of increased complexity. The web app’s core is built with the PixiJS library and its components. PixiJS uses containers to store objects such as sprites and graphics. These containers are passed to the renderer, which draws container-content onto a WebGL-enabled canvas. As Magno is a fairly small application and PixiJS containers are extendable classes, it was sensible to combine the model, view, and controller into single classes. This class structure links an object’s logic, graphics, and input handlers to a single class, which reduces complexity by decreasing the number of interacting components the developer must manage. A downside to this design pattern is that classes can get fairly large; however, this was not an issue in this project.

The Game Loop

The game loop is at the core of any application that changes state based on time. This loop typically computes logic and physics before drawing the updated model to the screen [61]. JavaScript is a single-threaded language, which can cause timing issues in the game loop. These issues make the application non-deterministic, meaning it can behave differently on various devices depending on hardware performance. All tests must behave the same regardless of device, so we needed a game loop implementation that behaves deterministically. Deterministic behavior is especially crucial for the motion test and its many collision computations which can easily cause timing issues.

Luckily, a library called `MainLoop.js` provides a robust game loop for handling the timing issues related to JavaScript [62]. With `MainLoop.js`, we can set the simulation time between each frame update, allowing us to cap the maximum perceived FPS to 60. More importantly, it gives us deterministic test behavior independent of hardware limitations.

6.3 Design Choices

The application's overall design remains the same as the Java app, including font selection, colors, button layout, and tutorial structure. The overall design was kept due to the Java app receiving a high SUS score of 92,7 and using dyslexia-friendly fonts and colors. In addition, Johansen and Kirkerød concluded from user feedback that their design was easy to understand and pleasing to look at [3]. The tutorial structure is largely based on textual instructions, which fits the web solution as well, with the intended user demographic being people aged 18 to 75. This application assumes the tester to be literate, which is required to read and comprehend the tutorial instructions. It also assumes basic computer interaction skills to navigate the tutorial and complete the tests by interacting with a mouse, keyboard, or touch device.

6.4 First Iteration

The first iteration's goal was to implement the motion test with an introduction, tutorial and result screen. This section will provide an overview of the first iteration's screens with notes on additions and changes to the graphical interface and changes made to optimize performance. The section further presents SUS scores from the usability test of this first iteration.

6.4.1 Application Overview

This section covers the app screens in chronological order starting with the first screen you meet when visiting the website. Most of the screen captures were taken on a wide monitor and cropped to make their contents readable. Thus, the contents presented do not reflect the actual size computed from the monitor aspect

ratio.



Figure 27: Introduction screen.

Figure 27 shows the introduction screen which has the Magno logo [3] at the top and the subheading "*MOTION TEST*". This subheading reflects the name of the test, which is either "*MOTION TEST*", "*FORM FIXED TEST*" or "*FORM RANDOM TEST*". Following the heading are two paragraphs: one explaining what the test does and its purpose, and the second describing the application's course of events. The text buttons are revised, now sporting rounded edges with a darker shade of blue and a drop shadow to visually stand out. The mouse becomes a pointer when buttons are hovered and clicked. Notably, the navigation menu on the left-hand side has been removed since the tests will live separately. The settings screens are also removed since we no longer want to tweak the app on the client-side.

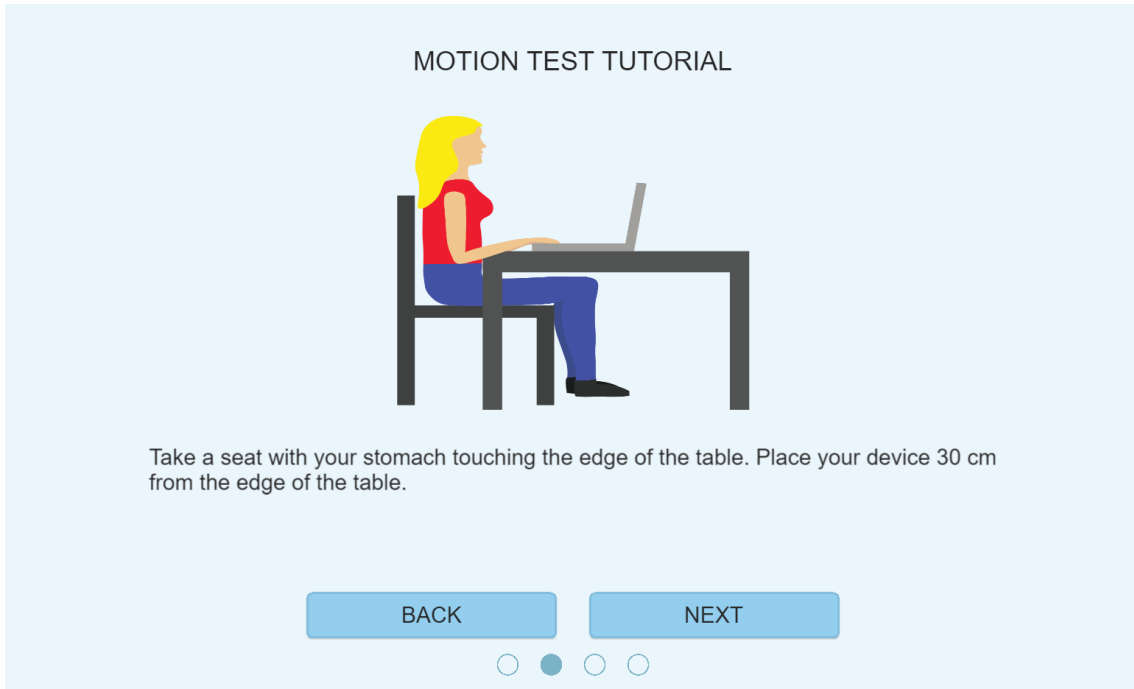


Figure 28: Tutorial introduction screen.

Figure 28 instructs the tester to sit close to the table with the device at a certain length. A notable change is the removal of the "SKIP" button. This button allowed the tester to go straight to the test without completing the tutorial. The button was no longer wanted due to the increased risk of error in a self-managed setting, where skipping the tutorial could cause invalid test results. Having the tester go through the tutorial screens seeks to prepare the tester thoroughly. This button is also removed from the other tutorial screens.

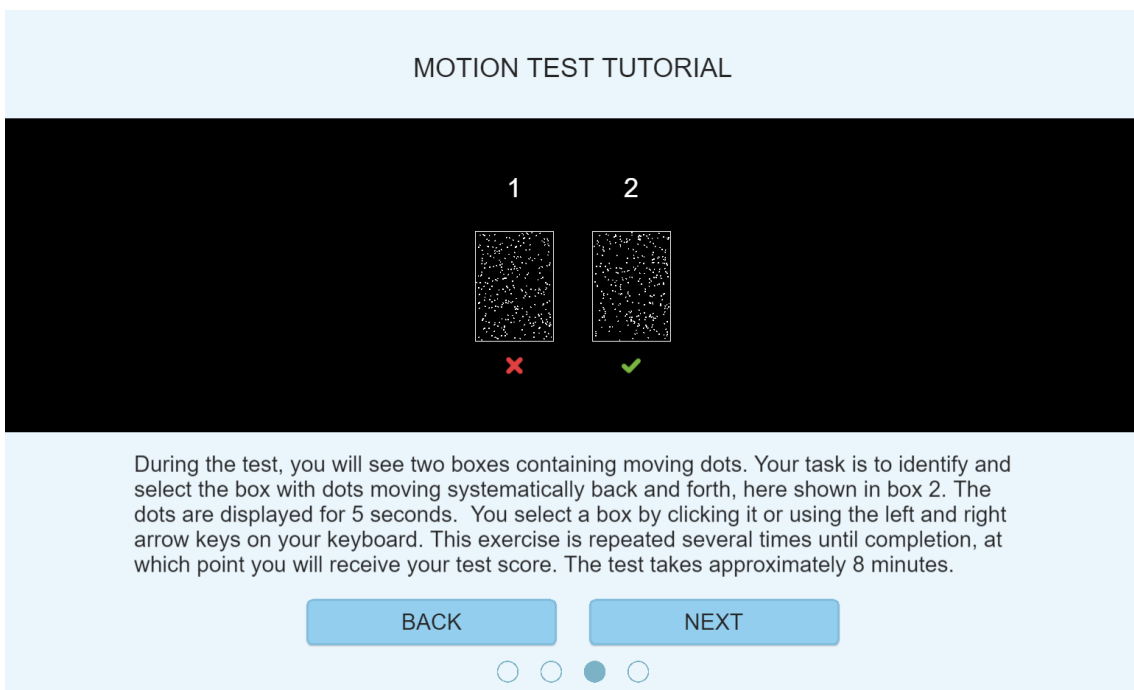


Figure 29: Tutorial task screen.

The tutorial task screen in Figure 29 details the test task. The text description is far more detailed than before to accommodate the missing facilitator who previously instructed the tester. It adds explanations of how long the dots are displayed, how to select a box, and that the task is repeated multiple times. The word *patch* is changed to *box* in the user interface to avoid confusion. A green checkmark replaces the arrow pointing to the coherent patch, and a red cross is added to the random patch. These symbols and their colors clearly distinguish correct from wrong. Patch labels 1 and 2 are added to keep a consistent layout in the app.

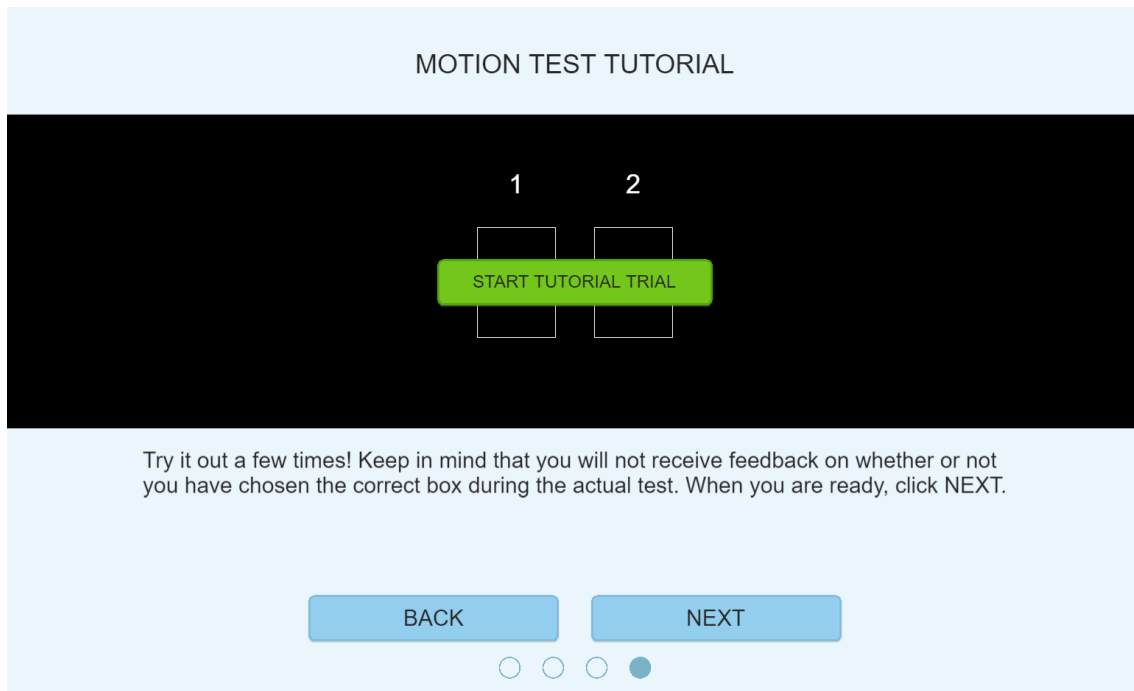
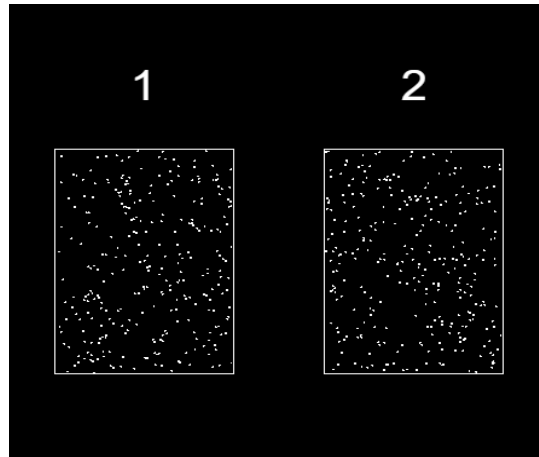
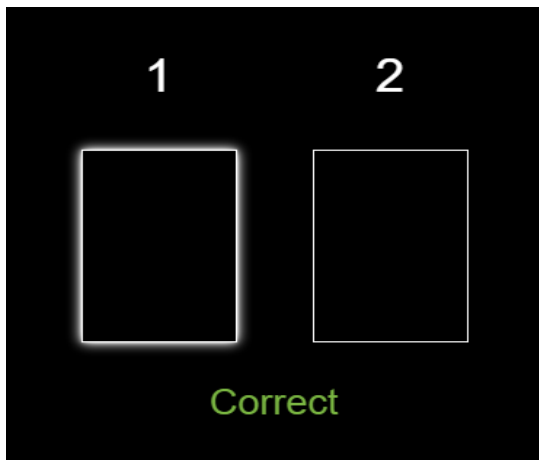


Figure 30: Tutorial trial screen.

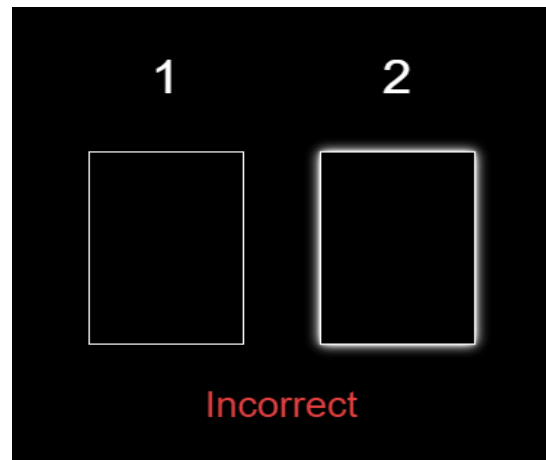
Figure 30 shows the tutorial trial screen which has only a few changes. The start button's text is modified from "*START TRIAL*" to "*START TUTORIAL TRIAL*" to indicate that this test is part of the tutorial and not the actual test. The description is extended with a sentence that indicates the tester may proceed before completing all trials.



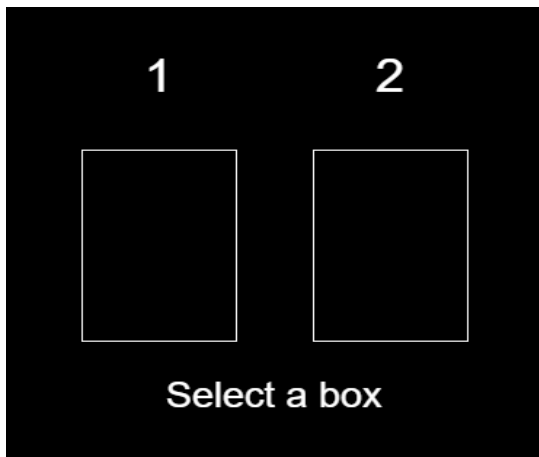
(a) Running.



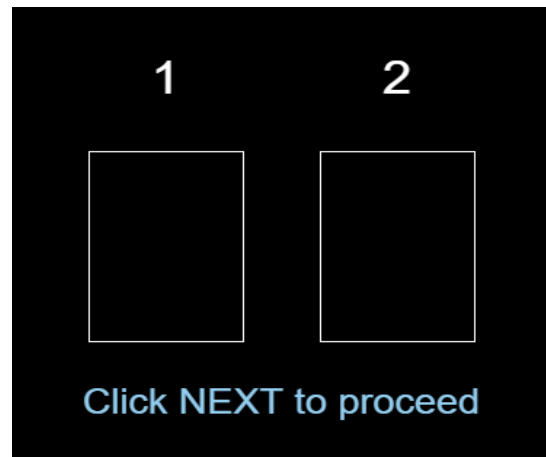
(b) Correct selection.



(c) Incorrect selection.



(d) Paused.



(e) Finished.

Figure 31: The various states of the tutorial trial.

The different states of a trial is shown in Figure 31. 31a shows a trial in the running state, where some dots are moving coherently in one patch and randomly in the other. 31b and 31c shows part of the screen when a patch is selected correctly and incorrectly, respectively. The feedback text is moved below the patches to not obstruct the view. It is also shortened to a single word, namely *"Correct"* and

"Incorrect". A white highlight surrounds the selected box, followed by a slight pause before the next trial begins. The highlight is an addition that gives the tester clear feedback on which patch they have selected. 31d shows the trial pause state. New to this is the text displaying *"Select a box"* which was added to guide the tester in case of confusion when dots are no longer displayed. The final state in 31e shows the finished state, which occurs when all three trials are completed. The feedback text is moved below the patches, as with the texts in the other states.

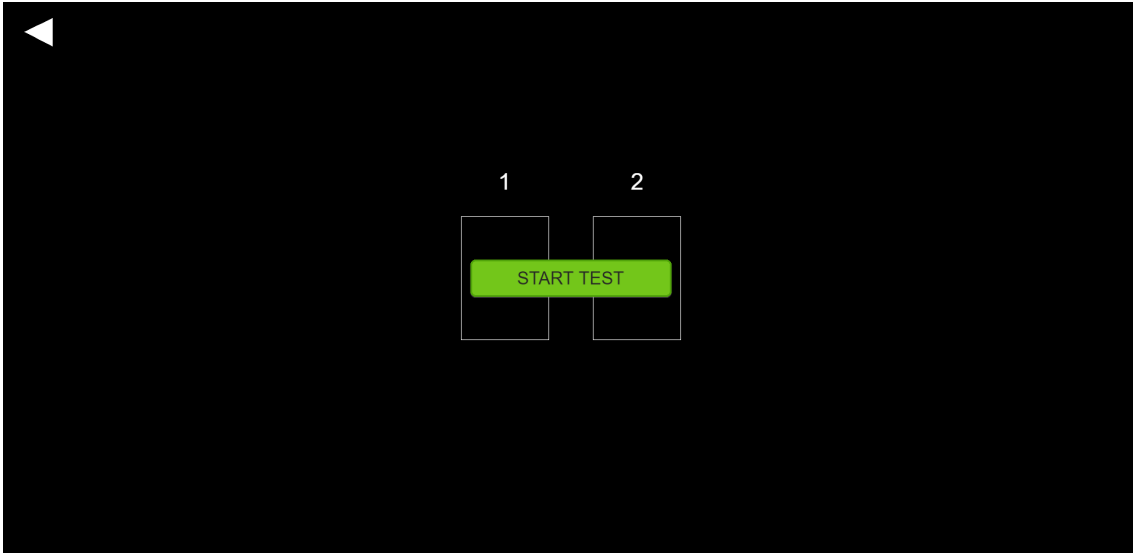
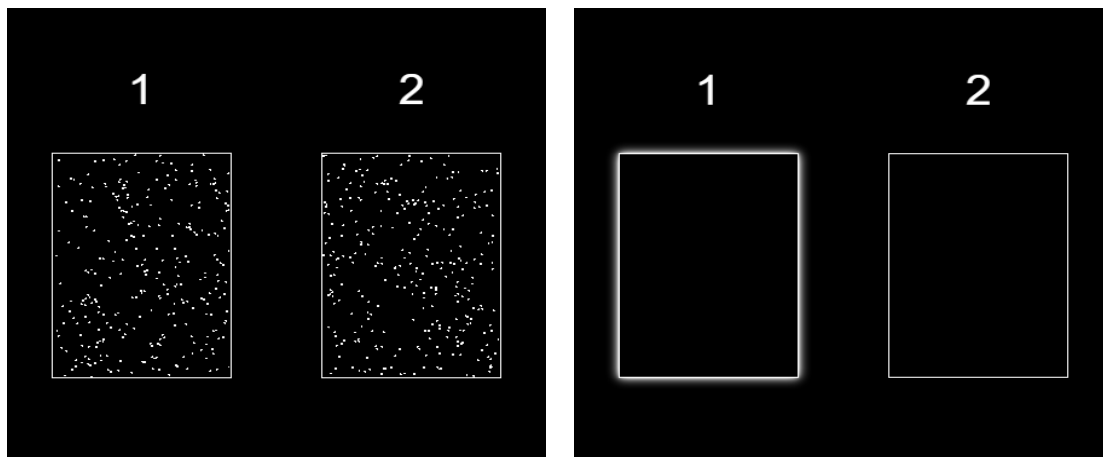


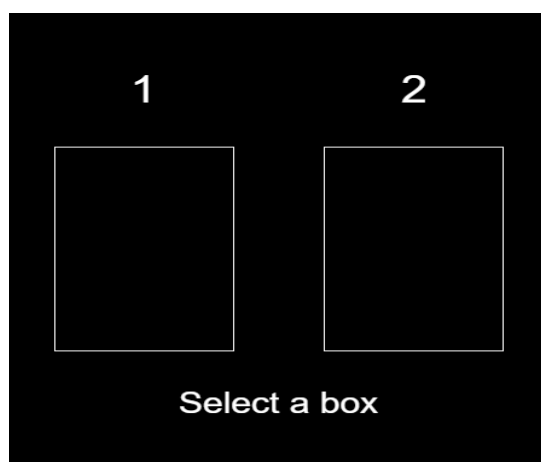
Figure 32: Test screen.

The test screen in 32 is modified with an added start button reading *"START TEST"*. The previous implementation showed empty patches before the test start and instead required a click anywhere on the screen to start the first trial. The start button provides a clear mechanism to start the test. It shares similar states to the tutorial trial as seen in Figure 33, including the running state, selected state, and paused state. In opposition to the tutorial trial, there is no indication of right or wrong selection in the test. In the Java app, the next trial continues immediately after selection. This sometimes made it difficult to see that a new trial had started, which is why a slight pause was added between selection and the next trial.



(a) Running.

(b) Patch selected.



(c) Paused.

Figure 33: The various states of the motion test.

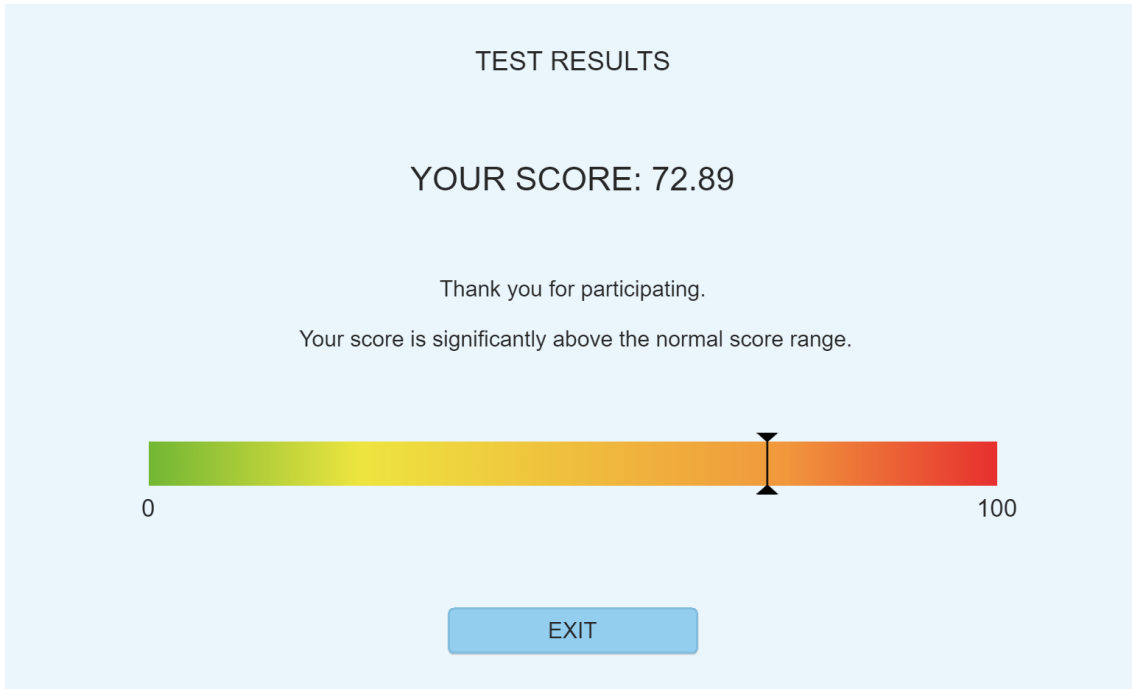


Figure 34: Result screen.

The result screen received only minor changes. Replacing the button that takes you to the introduction screen in Figure 27 and the button that copies test results to the clipboard, there is now simply an exit button that closes the browser window. The result bar's extremes still have number labels, but their text labels are removed. These labels stated "*Little to no problems*" on the left extreme and "*Severe problems*" on the right extreme. Neither the motion nor form tests are diagnostic, so these wordings were deemed too conclusive and thus removed. A line has been added to thank the tester for participation.



Figure 35: Loading screen.

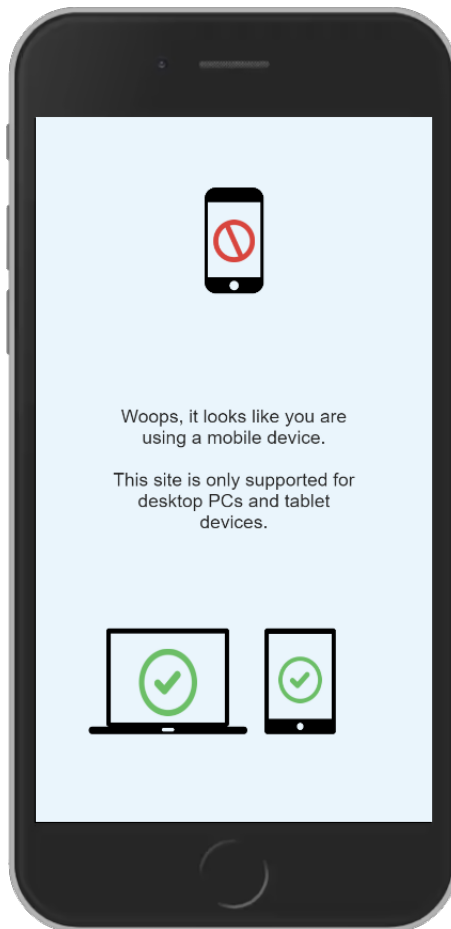


Figure 36: Mobile screen.

A loading screen with the Magno logo and an orange spinning wheel in the center of the screen, see Figure 35, was added. This screen is shown while the app loads all screens and their components when first starting the app. The loading screen communicates that the app is getting ready instead of showing a blank screen or a stuttering app that might act as a nuisance.

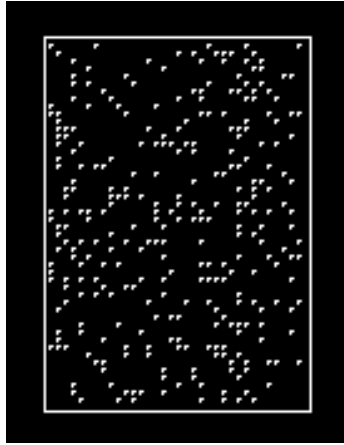
Figure 36 shows the screen displayed when the tester tries to access the app on a device whose screen width is less than 760 pixels. It displays a text explaining which devices are supported and which are not.

6.4.2 Performance Optimization

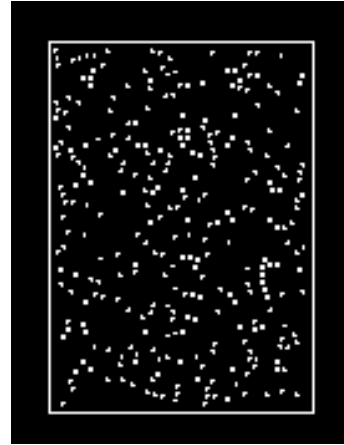
The most computationally heavy part of the application is the motion test. It contains hundreds of dots per patch that can collide with each other and the patch's boundaries. Additionally, they change direction approximately every half second. The first attempt at implementing the motion test copied the Java app's implementation. This implementation caused a significant performance hit, to the point where the app struggled to maintain a steady frame rate even on high-end gaming computers. A few changes were made to boost the motion test's performance.

PixiJS has a built-in component called *PIXI.ParticleContainer* which can be used to render many sprites at fast speeds. Switching from a regular *PIXI.Container* to a *PIXI.ParticleContainer* improved the performance slightly. Another performance booster is the quadtrees from the Java implementation, which are brought over to prune dot collision checks. However, the main performance bottleneck discovered was the function placing dots at their initial positions. In the motion test, dots are initially placed randomly across the patch with a minimum distance to its neighbors. The Java implementation randomly generates a location and checks that there are no neighbors within the vicinity of the minimum distance. If a spot is occupied or neighbors are too close, the function generates a new location until a free spot is found. As the patch is increasingly occupied, the number of locations generated and distance checks increase rapidly.

This problem was solved by reducing the location space of the patches. Instead of using all possible patch locations, a grid of lines spaced at least by the minimum dot distance was created. The intersecting lines make up the new location space. These points are shuffled for randomness and stored in a list, and dots get their location by pulling the first item off this list. The precomputed list significantly improves performance while ensuring that dots are properly spaced. The straight grid lines caused initial placements to not look random, so they were replaced by sine waves to increase perceived randomness. Figure 37 shows the difference in perceived randomness between straight grid lines and sine waves.



(a) Straight lined grid.

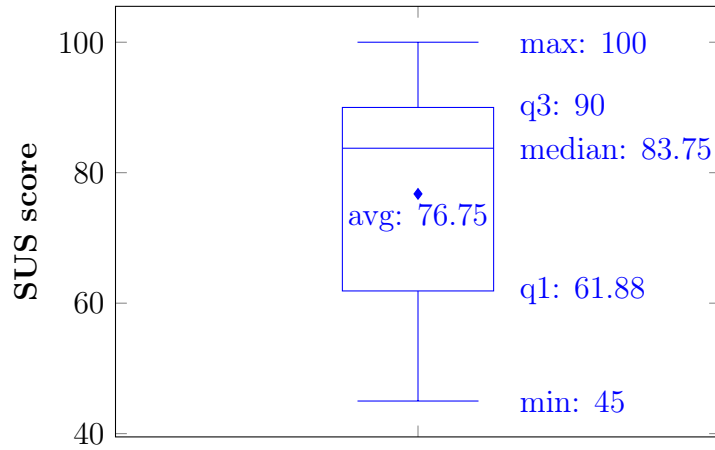


(b) Sine wave grid.

Figure 37: Initial dot placement with straight lined grid and sine wave grid.

6.4.3 Usability Testing

The usability test of the first iteration was carried out by sending an email to employees at the department of computer science (IDI) with a link to the website hosting the motion test and a link to the questionnaire hosted on Nettskjema. The email presents the web app and instructs the tester to first complete the motion test before filling out the form, and noting that no sensitive data is recorded. There were 20 participants aged 24-81 of which 5 were female, 14 were male and 1 who preferred not to disclose their gender. 2 participants took the test on tablet devices, 18 on PCs. The age, gender and device distributions are in Figure 39. The average SUS score was 76,8, as seen in Figure 38. Figure 40 shows the average score per question.



1st iteration, motion test. (n=20)

Figure 38: Boxplot of SUS scores.

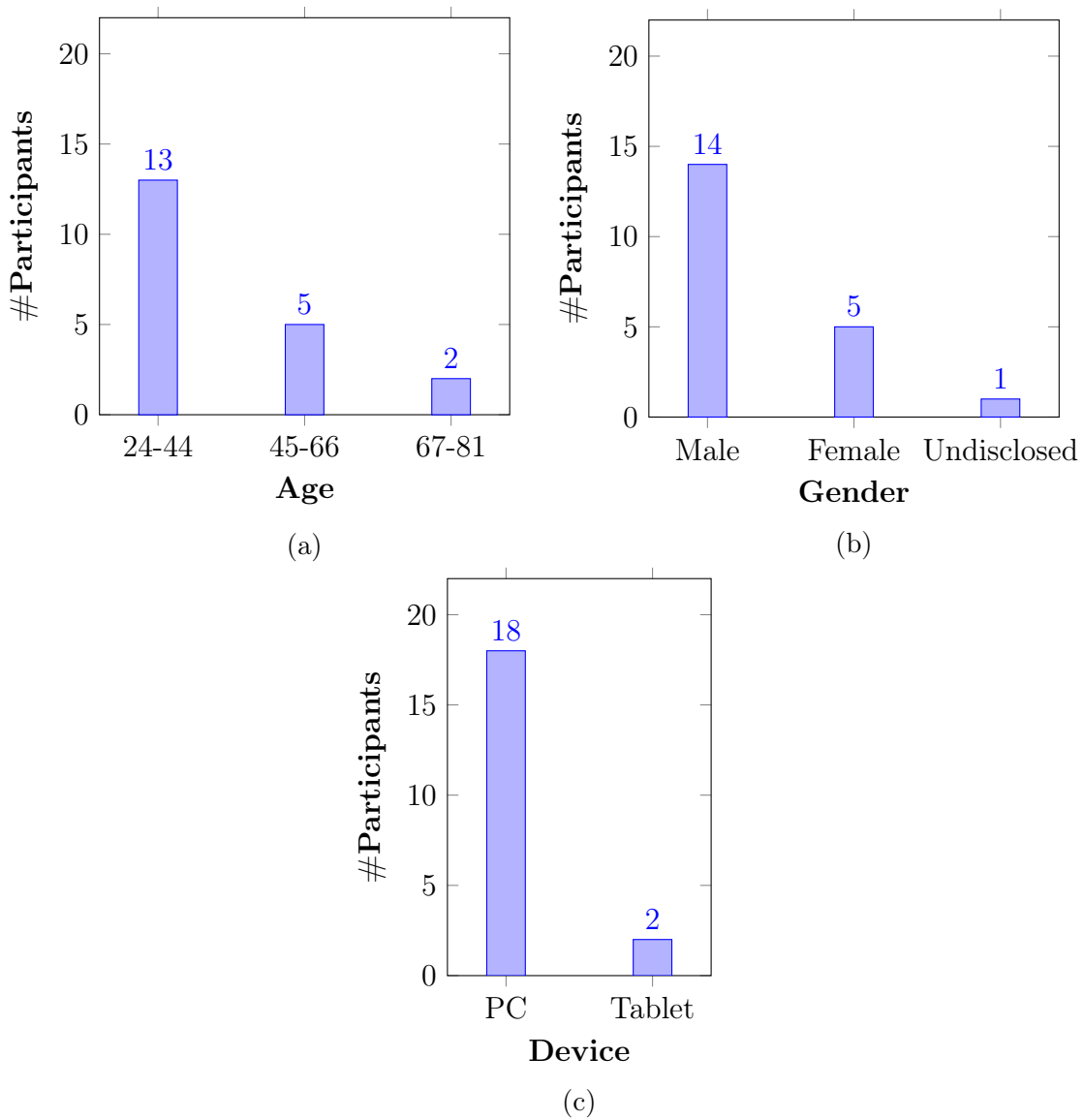


Figure 39: Participant distributions across age, gender and device.

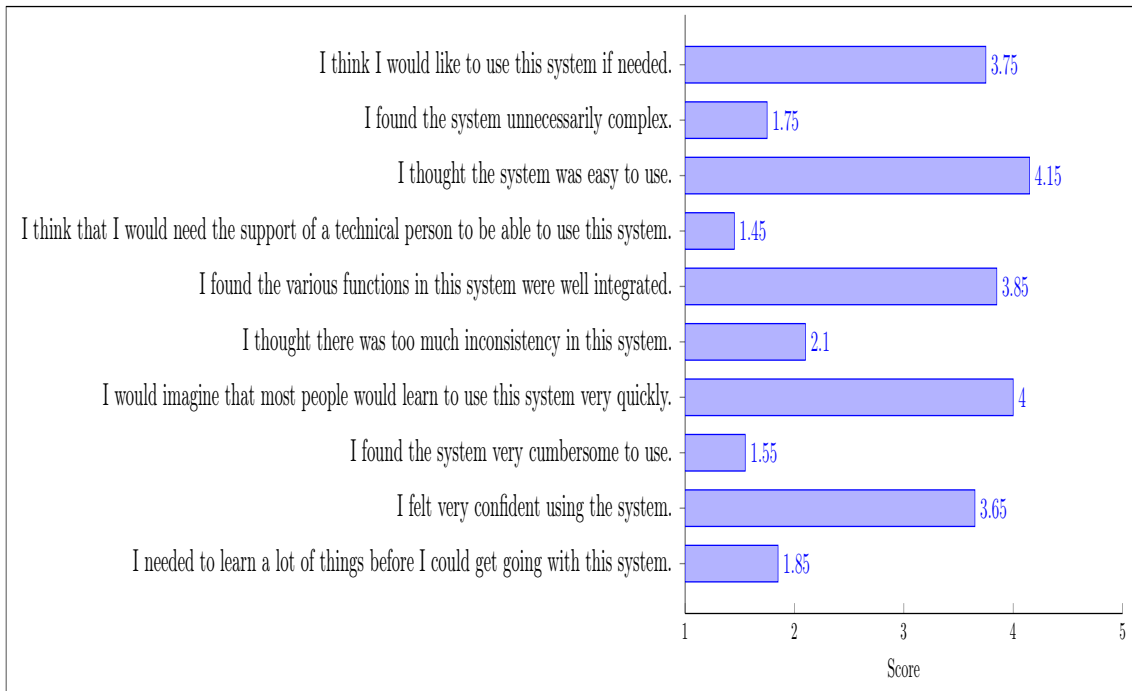


Figure 40: Average score per SUS question.

6.4.4 Written Feedback and Proposed Changes

In addition to the SUS scores, testers provided useful feedback in their free-text answers to question 14 and 15 in Figure 51. These answers were reviewed to discover major or minor flaws that could be improved upon. Some of the feedback did not make sense to act upon, such as a tester stating being confused when the difficulty was great. This increase in difficulty is part of the test mechanic, so reducing it would make it hard to validate the web app against the Java app. Many suggestions were actionable, but time constraints forced prioritizing what to improve upon in the second iteration. The prioritized changes are in Table 4. See Appendix F for all written feedback provided by testers.

Id	Proposed change description
C1	Clarify that the difficulty increases or decreases depending on the answer given.
C2	Make sure both boxes are to be selected in the tutorial trial, as not to prime the test subject to either patch.
C3	Clarify that precision is valued over speed.
C4	Provide more tutorial trial examples.
C5	Increase the rate of change in difficulty in the tutorial trial so the user experiences a difficult trial at least once before taking the test.
C6	Change the wording of the free text questions from "application" to "tutorial or test".

Table 4: The proposed changes spawned from user feedback.

6.5 Second Iteration

The second development cycle sought to implement the two form tests, form fixed and form random, with the auto and manual mode options as described in Section 4.1.1. This section will provide an overview of the modifications made to the user interface and other aspects of the app based on user feedback from the first usability test. Furthermore, it will describe additional functionality that did not make it into the first iteration before presenting this iteration's usability test results.

6.5.1 Application Changes

Since the layout for the motion and form tests are the same, this section will cover the implemented changes from Table 4.



Figure 41: Introduction screen.

The introduction screen’s text was reworded slightly by removing the second sentence of the first paragraph to keep the introduction short and to the point.

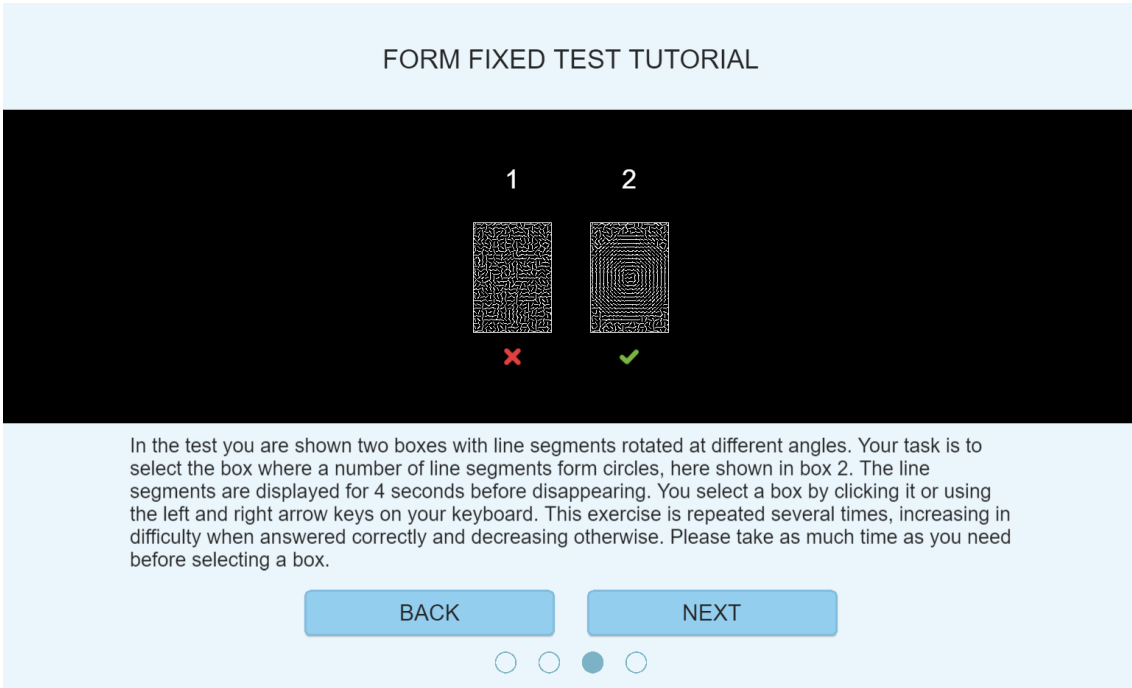


Figure 42: Tutorial task screen.

In the tutorial task screen (Figure 42), proposed changes C1 and C3 from Table 4 were implemented. C1 was implemented by changing the second to last sentence to:

This exercise is repeated several times, increasing in difficulty when answered correctly and decreasing otherwise.

This way, the change in difficulty is clear to the tester. C3 was implemented by changing the last sentence to:

Please take as much time as you need before selecting a box.

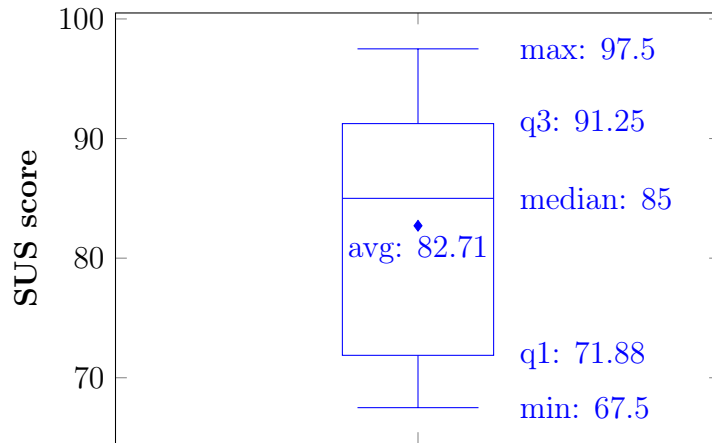
This sentence aims to clarify the precision versus speed ambiguity, aiming to lead the user to take their time rather than rushing through the test.

Further, the tutorial trial screen was changed to reflect the proposed changes C2, C4, and C5. C2 was implemented by setting the coherent patch in the second trial as the opposite of the coherent patch in the first trial. This ensures that both patches are selected at least once during the tutorial trials, giving a smaller chance of priming the test subject to either patch. C4 and C5 were implemented by increasing the number of trials from 3 to 7 and increasing the change in difficulty from one trial to the next.

Other additions to this iteration were a simple HTTP server to receive test results and a corresponding HTTP POST method with the fetch API for sending test results. Multi-language support was added by storing all text in translation files loaded by the *i18next* library. Resizing was changed from simple canvas scaling to redrawing objects at the appropriate size with regard to the window aspect ratio.

6.5.2 Usability Testing

For the second usability test, C6 was enacted to change the wording of the last two questions from "*application*" to "*tutorial or test*" in order to avoid confusing terms. This test was sent to other employees at IDI with a link to the website hosting the form fixed test and a link to the questionnaire hosted on Nettskjema. The email sent out shared the same format as the first one. This usability test had 12 participants aged 24-59, of which 3 were female, and 9 were male. All participants conducted the test on PCs. The age, gender and device distributions are in Figure 44. As seen in Figure 43s boxplot, the average SUS score was 82,7, a significant increase. Figure 45 shows the average score per question.



2nd iteration, form fixed test. (n=12)

Figure 43: Boxplot of SUS scores.

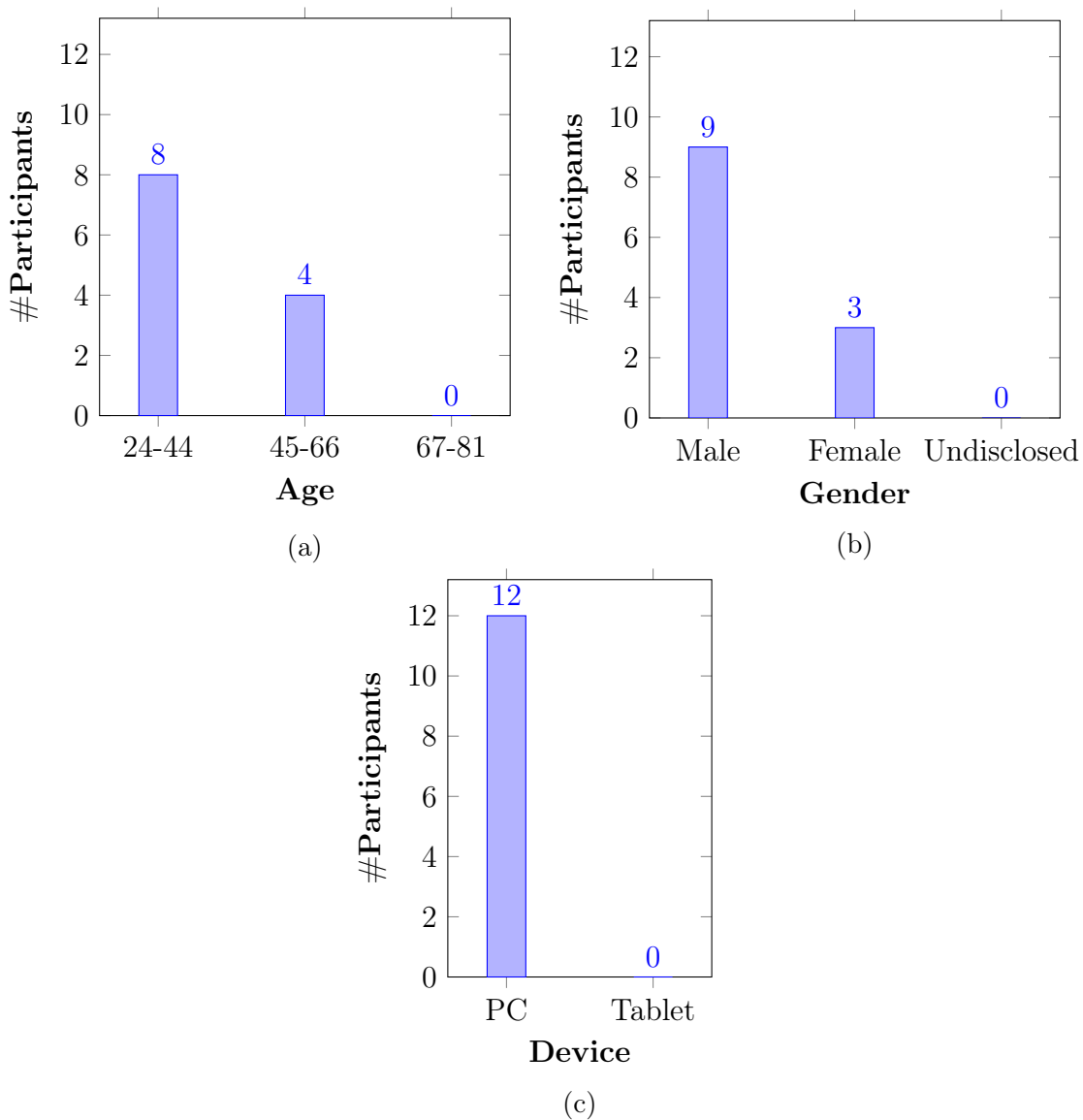


Figure 44: Participant distributions across age, gender and device.

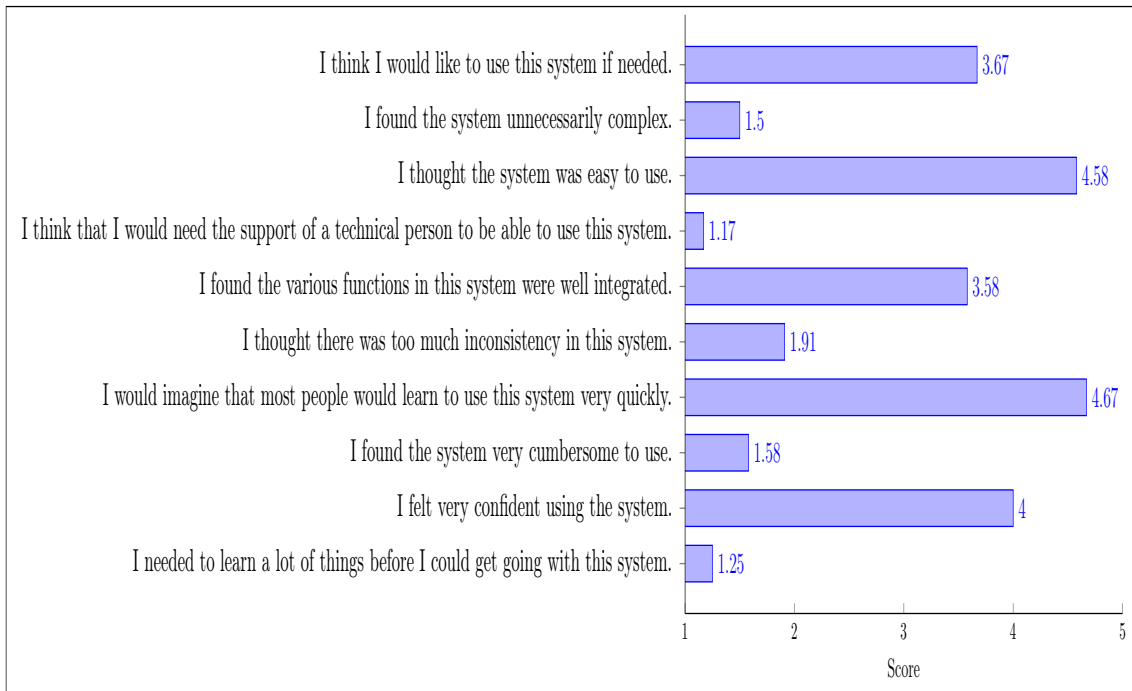


Figure 45: Average score per SUS question.

6.5.3 Written Feedback

Again, the testers provided insightful feedback on the design and use of the application in their written answers. Development seized after the second iteration, so this feedback is discussed to address limitations of the project's current state and suggest future solutions. Only the feedback deemed most interesting is discussed here. See Appendix G to read all answers given.

Three testers commented on the hard trials where the patches are almost indistinguishable, suggesting there is still ambiguity around what to in such cases:

After a few iterations I could really NOT see any circular patterns, so I was selecting randomly. I would expect instructions on what to do in such a case to exist in the tutorial.

When I swear there was no circle I just randomly picked and I sort of wish there was a 'no circle' option maybe by pressing down.

The hard tasks feel very random, they don't make sense, which doesn't feel very good. I understand that the difficulty is supposed to be increasing, but I could imagine a less experienced person being unsure if everything was working correctly when none of the boxes have circular patterns.

This could be improved by explicitly instructing the tester to guess in such cases. Alternatively, one could follow the second suggestion and include a "no circle" button, which could be integrated into the existing format by counting a click on this button as a wrong answer. Further, two testers noted they would like to get information on the state of progress:

Translated from Norwegian: progress bar showing how far (in %) one has progressed / has left.

Provide feedback on the state of the process.

One of the testers in the first usability test also wanted feedback on the state of progress. However, progress is non-linear and is instead dependent on the number of reversal points, where the tester goes from having selected wrong to right, or right to wrong. This would perhaps require a more sophisticated solution than showing a percentage. For example, showing 50% when having reached half of the reversal points does not necessarily imply that the remaining half takes as long as the first half. Another issue is that it could distract from the task at hand, which is to answer to ones best ability, and instead focus on progressing to the end of the test. So instead of attempting to answer correctly, they may be busy figuring out how to progress.

6.6 Classes and Interfaces

This section presents an overview of the classes and interfaces that make up the final application and their relations. Figure 46 shows the UML class diagram of the final app. Green boxes represent new classes and interfaces while white represents those with the same name and purpose as in the Java app.⁴

⁴Additionally, there is an abstract Settings-class storing app settings in static variables, which most classes use. This class has been excluded from the diagram to avoid cluttering.

The class structure is similar in design to the Java app, with a few notable changes. There are fewer classes in total due to some components being removed. The settings screens were no longer needed as it did not make sense to have adjustable settings on the client-side for deCODE's use case. Input handler classes are also removed as they are merged with each screen that has user interaction. The render classes are also gone due to PixiJS handling the entire rendering process. The classes and interfaces in green are now briefly described.

LandingPageScreen is the first page you visit which shows the logo and the introductory text. It has a next button that takes one to the tutorial.

Patch is a class for drawing the white rectangles that contain dots or line segments.

TextButton is used to create all buttons with text labels, such as the "NEXT", "BACK" and "START TEST" buttons.

SpriteButton is for creating sprite-based buttons without text labels. It is used to create the back arrow button in the *TestScreen*.

GameApp is responsible for handling the game loop, renderer, loading assets, and managing screens. It uses the *Screens* interface to create the object for storing all screens.

TestScreen is responsible for creating the motion and form tests and keeping track of test progress. It uses the *Trial* interface to create objects that store data for each test trial and generates test results when a test is finished. The results are passed to *GameApp* which further instantiates *ResultsScreen* with the test results.

ResultsBar is an object used to create the bar which displays the test score.

The *ScreenSettings*, *MotionSettings*, *FormSettings*, *StaircaseSettings* and *TutorialSettings* interfaces are used to clearly define the settings included in the test results.

MobileScreen is the new screen shown when trying to access the app on a mobile device.

LoadingScreen is used to display the logo and a spinning wheel to indicate the app not being ready for user interaction.

7 Evaluation

This section will discuss the results of the usability tests and evaluate the application with regards to the predetermined requirements in Section 4.4.

7.1 Usability Testing

With the two usability tests conducted, this section attempts to answer the following questions:

1. How significant was the increase in SUS score from the first to the second usability test?
2. What is the perceived usability of PC and tablet users?
3. What is the perceived usability across age groups?

To answer question 1, we will compare the results from the two usability tests. Figure 47 shows a box plot of the SUS scores of both usability tests next to each other.

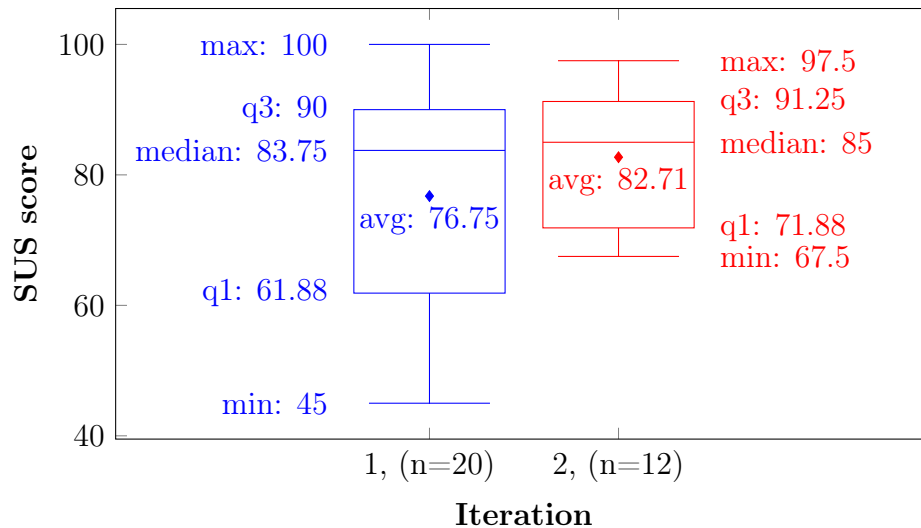


Figure 47: Box plot of SUS scores from both iterations' usability tests.

Both usability tests have roughly the same median, the first being 83,75 and the second being 85. However, the first iteration received varied responses with a minimum score of 45 and a maximum of 100, suggesting considerable disagreement between testers on the perceived usability. The second box plot is more compact, supporting the notion that users are more in agreement on the usability score than in the first iteration. From this, we hypothesize an increase in perceived usability, which can be tested statistically. With a confidence level of 95% and an $\alpha = 0.5$, calculating

statistical significance between the SUS scores of test 1 and 2 gives $p = 0,0002$ from the t -distribution. Since our p -value is much lower than our α , there is a statistically significant increase in perceived usability.

As for question 2, it is hard to say anything definite regarding the user experience on tablet devices. There were only two tablet users in the first usability test and none in the second. Observing the average SUS score of each device type in the first test, the PC users had an average of 77,9 and the tablet users 66,3. This may suggest responsiveness or performance is worse on tablets; however, further testing with tablet users is required to pinpoint concrete issues.

Next, to indicate performance per age group, we first have to define the groups. We will use the same age intervals as presented in the earlier age distribution graphs and label ages 24-44 as adults, 45-66 as middle-aged, and 67-81 as seniors. We then calculate the average SUS score for each group.

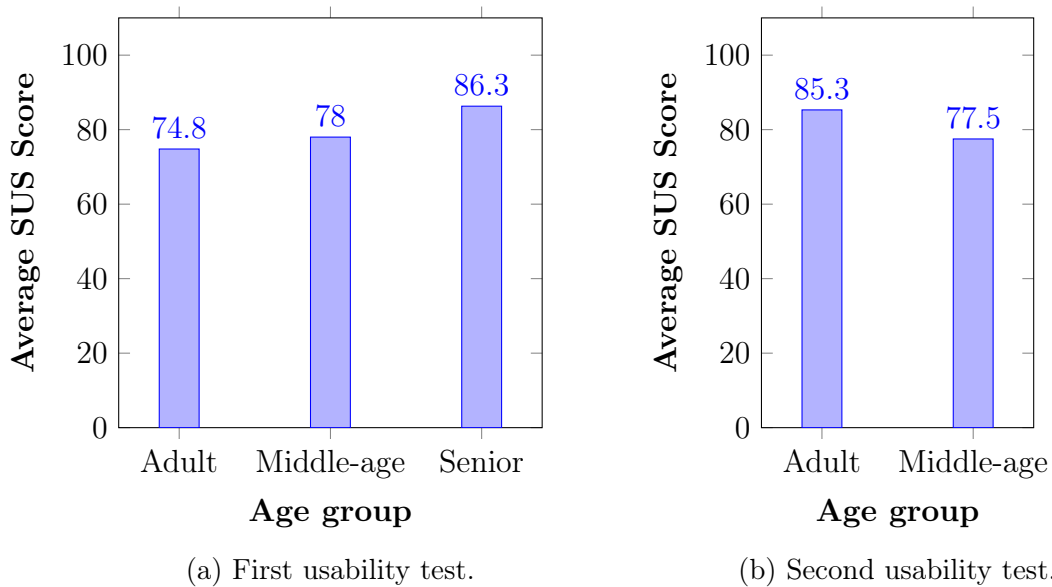


Figure 48: The average SUS scores per age group.

As Figure 48 shows, the average SUS score was highest among seniors and lowest among adults in the first usability test. In the second usability test, adults scored the highest, and middle-aged people scored the lowest. No seniors participated in the second test. We can see that scores varied roughly between the mid-70s to mid-80s for both tests, without significant score gaps. An interesting observation in the first usability test is that the highest scores are among seniors. People’s problem-solving proficiency in technically-rich environments is lower for middle-aged and senior people [63], so it is peculiar that the first usability test contradicts this result. This contradiction might be due to there only being two seniors among the participants. Overall, the discrepancy between age groups is at most roughly 10 points for each test, suggesting that the app performs well for people of different ages.

7.2 Requirements Fulfillment

This section evaluates the application with regards to the functional- and non-functional requirements from Section 4.4, summarized in tables. Each table references the requirements by id, provides an evaluation of the extent to which the requirement is met, and labels them attained, partly attained, or unattained.

7.2.1 Previous Requirements

Table 5 evaluates the app with regards to the requirements brought over from previous iterations. To read their full descriptions, see Table 1.

Id	Evaluation	Fulfillment
FR1	The application implements the random-dot kinematogram and static global patterns as a motion, form fixed, and form random test, described in Section 4.1.1.	Attained
FR2	The application uses screen size, screen resolution and viewing distance to calculate the size of objects on the screen.	Attained
FR3	The application provides a set of default settings.	Attained
FR4	The application calculates a threshold score after completing a test.	Attained
FR5	The test subject is first presented an introduction and a tutorial before accessing the test.	Attained
FR6	The threshold score and settings are stored in a test result object after a completed test run.	Attained
FR7	The test can be exited at any time simply by closing the browser or browser window. It is also exited by clicking the back button or pressing backspace on the keyboard.	Attained
FR8	The application is responsive to devices' screen size.	Attained
NFR1	The last usability test of the application reached a SUS score of 82,7.	Attained
NFR2	The application runs close to 60 FPS on most devices. See the next section for more details.	Partly Attained

Table 5: Requirements brought over from previous iterations of Magno. See Appendix A for all previous requirements, including those not brought over.

Performance Testing

Performance testing was conducted on various devices to verify that NFR2 was met. A selection of high-end and low-end laptops and tablets were used to check performance across different hardware. The devices used were an ASUS TUF gaming laptop (2018), a MacBook Air (2011), an Ipad Pro (2017), and a Samsung Tab A (2015). The Figure 49 shows the performance of the ASUS laptop during a run-through of the motion and form fixed tests.

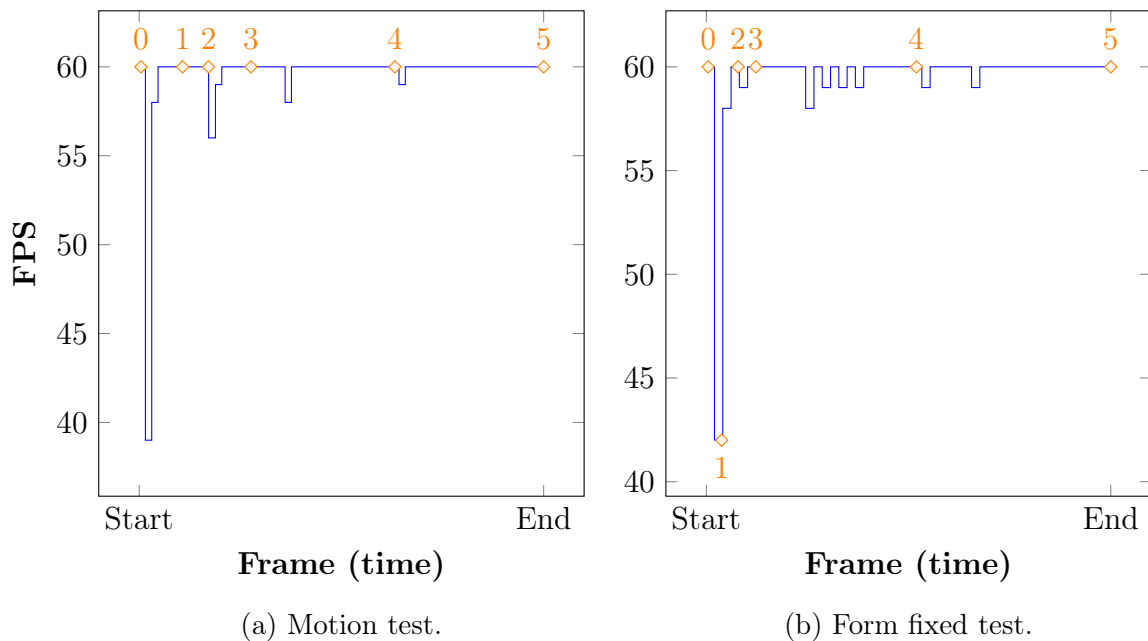
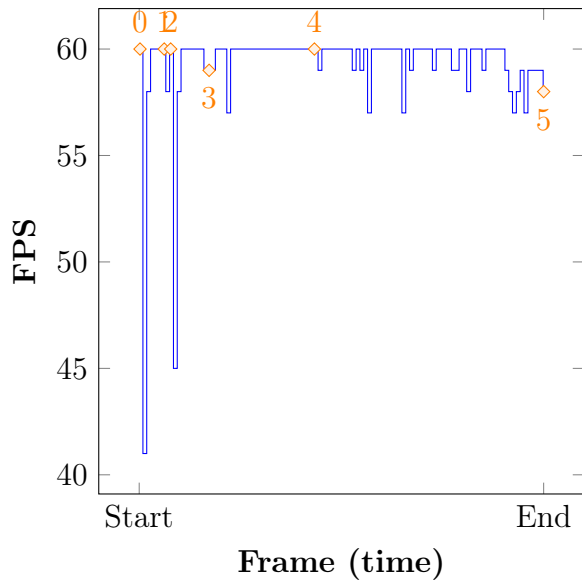
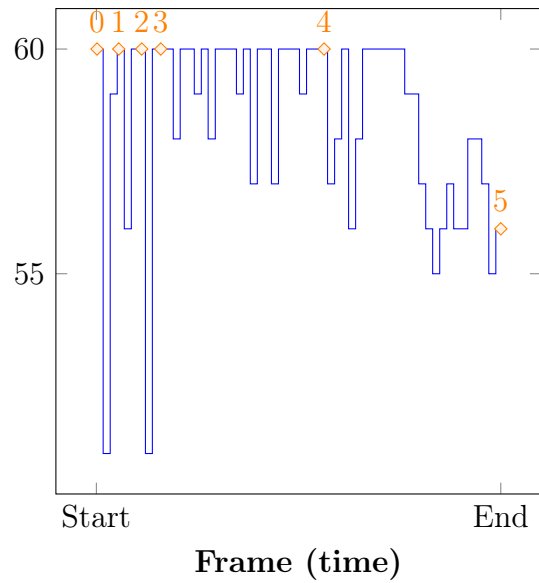


Figure 49: Recorded performance from a complete run-through of the motion and form fixed tests on an ASUS TUF gaming laptop using Brave Browser.

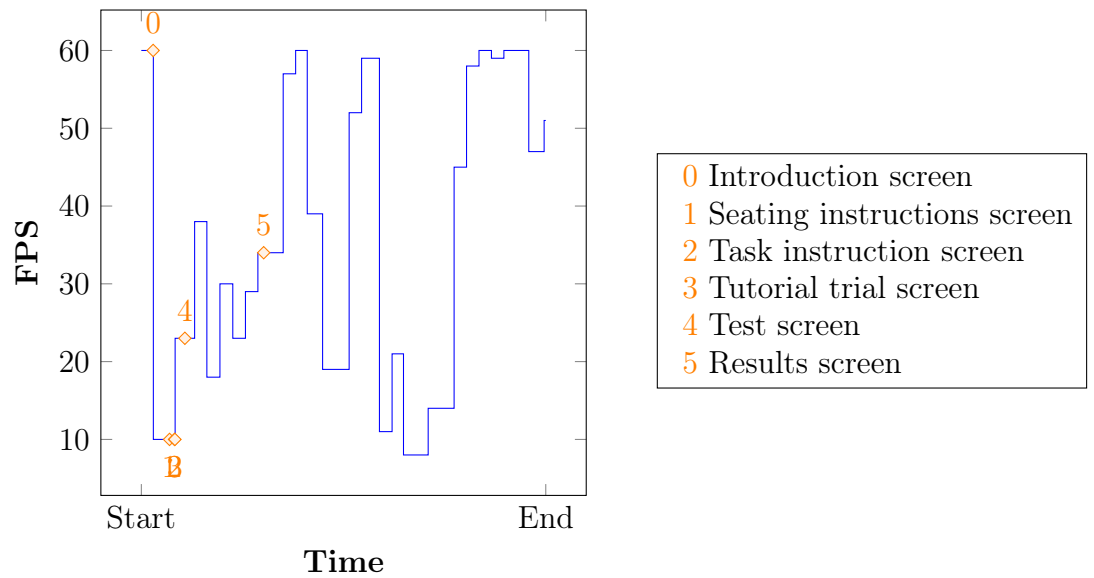
Performance is very similar for both the motion and form fixed test. There is a dip in performance when first loading the web page, but this slowdown is expected. After loading, performance is fairly stable with only a few minor dips. Only the motion test was used for performance testing of the other devices, due to being the most computationally heavy test.



(a) MacBook Air using Safari.



(b) Ipad Pro using Safari.



(c) Samsung Tab A using Chrome.

Figure 50: Recorded performance from a complete run-through of the motion test on a MacBook Air, Ipad Pro, and Samsung Tab A.

As seen in Figure 50, both the MacBook Air and Ipad Pro performed well with a steady frame rate of 55-60 FPS. However, the Samsung Tab A struggled significantly compared to the other devices, which suggests further optimization is needed to ensure the app runs smoothly on weaker hardware.

7.2.2 New Requirements

Table 6 evaluates the app with regards to the new requirements made in collaboration with my supervisor and deCODE. To read their full descriptions, see Table 2.

Id	Description	Fulfillment
FR10	The cursor changes to a pointer when hovering over clickable content.	Attained
FR11	We consider mobile devices to be those with a screen width of less than 760 pixels. The app does not load if the device width is beneath 760 pixels. It is still technically possible to access the test on mobile devices by opening the page in desktop mode.	Attained
FR12	When accessed on devices with a screen width less than 760 pixels, the app displays a screen with a message telling the user which devices it is unavailable and available for.	Attained
FR13	The motion, form fixed and form random tests are implemented as separate applications so that they can run independently of each other.	Attained
FR14	Settings are stored in a TypeScript class which is compiled with the rest of the code. However, extracting this information and placing it in a configuration file should be a straight forward task.	Unattained
FR15	An HTTP server with routes for handling POST-requests has been created.	Attained
FR16	The app has a function which can be used to send test results to a server.	Attained
FR17	The test results include the additional data for each test trial.	Attained
FR18	The app uses the i18next library to facilitate multi-language support by making it easy to add translation files for new languages. The app includes both a Norwegian and English translation.	Attained
NFR3	Settings are currently only adjustable by altering the <i>Settings</i> class in the source code.	Attained
NFR4	The SUS score from the final usability test suggests that testers were quickly able to grasp test concepts and complete them without too much difficulty. However, feedback shows there is still much room for improvement.	Attained

NFR5	The application is based on TypeScript which transpiles to JavaScript.	Attained
NFR6	The app has been tested on people of 24-81 years old.	Partly Attained

Table 6: New requirements for the web application

8 Discussion, Conclusion and Further Work

This section will first discuss the limitations of the usability test and to what extent the SUS score is satisfactory. The discussion is followed by a conclusion with answers to the research questions. Finally, future work for this project is proposed.

8.1 Discussion

This thesis has focused on porting the Magno Java app to a web-based solution while maintaining usability in the new, self-managed test environment. Remote asynchronous usability tests consisting of motion- or form test completion and a modified SUS questionnaire were conducted to test the application's usability. This test type was mainly used due to COVID restrictions and to reach a reasonably large sample size in a short amount of time while reflecting the intended self-managed test setting. However, several factors may have had an impact on the SUS score.

The first usability test employed the motion test, while the second used the form fixed test. Due to development taking longer than expected, only the motion test was completed after the first sprint. Thus, to receive feedback on both tests, the form fixed test was used in the second usability test. The use of different tests might impact the validity of the comparison between the two usability tests. However, both tests share the same navigation, tutorial structure, and design, so perceived usability on these aspects is assumed to remain consistent across both usability tests.

There are a couple of limitations relating to the participants' demographics. One limitation is that participants were only recruited from employees at NTNU's department of computer science. Employees of this particular department might be more technically proficient than the average person, which could inflate the SUS score. However, though no information on the participants' job positions was collected, they likely hold a range of different positions. Thus, actual technical proficiency may not be higher in these employees. Another demographic limitation is that our youngest participant was 24 years old, which did not meet deCODE's lower target age of 18 years old. However, research from OECD and Nielsen Norman Group has shown that technical proficiency decreases with age [63] [64], so there is reason to believe that an 18-year-old would not perceive usability as worse than a 24-year-old.

A factor that is likely to have greatly impacted the SUS score is the remote asynchronous test method used in this thesis instead of the local, synchronous method used in previous testing. In the previous method, a facilitator voiced the instructions to test subjects and divided the test into smaller sub-tasks. The method employed in this thesis required testers to read instructions independently and there was only one major task, which was to complete the motion or form fixed test. Providing instructions in person is a more reliable way of ensuring the task is understood,

leading testers to complete the test more effectively with a greater sense of satisfaction. Dividing the usability test into simpler sub-tasks could also contribute to increased user satisfaction. These could both partly explain this usability test's decrease in SUS score from the previous testing. Additionally, nine testers completed the questionnaire in two minutes or less, which could make their answers a crude representation of their experience.

8.2 Conclusion

In conclusion, the app performs well on most devices but still requires optimizing for use on weaker hardware. The SUS score decreased from an average of 92,7 to 82,7 compared to the digital prototype of 2017. This decrease is likely due to the changed usability test method. Still, the score meets the requirement NFR1 of reaching a minimum SUS score of 80. Furthermore, a score of 82,7 is considered *Excellent* on the adjective scale, and an *A* on the grade scale of Jeff Sauro [43]. With this in mind, we will now review the research questions.

RQ-1 How is app performance maintained when porting a Java app to JavaScript?

If the Java app requires a lot of graphics rendering, performance is maintained by using a fast renderer, like the one provided by PixiJS. PixiJS was mainly used to render the motion test's dots quickly. Further, performance is kept by optimizing slow algorithms. The bottleneck of the Java implementation was the algorithm responsible for the initial placement of dots. The Java implementation generated a random location for placement and checked that the new spot was unoccupied before placing a dot. As the number of dots increases, so do the number of collisions. The algorithm has to generate a new random location for every collision, causing a significant performance slowdown. The need for collision checks was removed entirely by creating a grid of equally spaced locations, guaranteeing proper spacing between dots. The grid solution caused a major performance boost. To further increase performance, all screens are loaded when first visiting the website to reduce loading times when changing between screens.

RQ-2 How can Magno's user interface support a self-managed test setting?

The user interface was prepared for a self-managed test setting by:

- Having clearer button design.
- Changing the mouse to a pointer when hovering buttons.
- Clearly distinguishing patches from right and wrong with a green checkmark and a red cross in the task instruction screen.
- Keeping patch labels consistent across the app to avoid confusion when they appear in the test.

- Adding a start button in the test screen to provide a clear path for the user to initiate the test.
- Highlighting the selected patch to communicate that selection is registered.
- Guiding the user when a test is paused.
- Supporting language translations for increased accessibility.

RQ-2.1 What changes to Magno’s tutorial instructions are needed to sufficiently prepare the test taker?

The following changes were made to provide clearer instructions:

- Shortening the introduction screen’s text and changing it to describe the overall test structure.
- Increasing the level of detail describing the task in the tutorial description.
- Stating that continuation from the tutorial trial marks the end of the tutorial.

RQ-3 How does a self-managed test setting affect the usability of the system?

The usability tests in this project were constructed to reflect the intended self-managed test setting. From the final usability test, we observe the SUS score climbing down from 92,7 to 82,7, despite maintaining the general layout of the Java app. The remote and asynchronous usability test method could play a role in this decline. Asynchronous testing asks more of the tester in that they have to read and understand instructions themselves without consulting an expert. In conclusion, a self-managed test setting appears to result in reduced perceived usability.

RQ-4 How can motion test results be validated in a self-managed test setting?

Motion test results are validated using the form fixed test to check that a participant has no issues with their parvocellular system. In addition, the total test completion time can be inspected to determine whether a tester rushed through a test without thoroughly considering which patch is correct.

8.3 Further Work

This section will discuss possible future work of the project with proposed actions going forward.

Before deCODE can integrate the web solution into Svipgerd, the tests must be validated against the Magno Java application. Test subjects should be recruited from two categories of reading ability: those with low reading competence and a group of subjects with high reading competence. The test results should be compared

to previous research using the Java app, to confirm that those with low reading competence perform worse than those with high reading competence.

Further, more extensive browser testing is required to ensure that the app works correctly for most modern browsers. The app has only been tested on recent Chrome, Brave, Microsoft Edge, and Safari versions. Although demonstrably working smoothly on a few devices, the app needs further optimization for devices with weaker hardware. The initial performance bottleneck was the spawning of dots. Where the greatest potential for optimization lies now remains to be discovered.

The remaining proposals are summarized in the following list:

- Further adjustments based on feedback from the questionnaires, including showing progress to the user, having a *Learn more*-link for details on the magnocellular dyslexia theory, and a third button for when testers deem the patches indistinguishable.
- Reimplement the app's UI components in a JavaScript framework or library, such as React, which uses HTML and CSS to handle responsiveness.
- Adding an Icelandic translation for deCODE's use case.
- Implement canvas fallback to support older browsers that do not yet support WebGL.
- Add text-to-speech functionality to allow people with low reading competence to hear the instructions read out loud.
- The results screen shows up a bit abruptly, so it might be sensible to include a transition. For example, by showing the loading wheel with the text "*Calculating/sending results...*" before changing to the results screen.
- Consider replacing the exit-button with a text saying: "You may close the window," as it is not reliable in closing the browser window.
- Conduct a separate usability test for tablet devices only, as there were just two participants using tablets.
- Host the project repository on a public account for easing further development and multiple versions, for example, for use in dyslexia detection among children or connecting to Angell's backend project from 2018 to collect and analyze test results.

8 References

- [1] Benny J. Oates. *Researching Information Systems and Computing*. SAGE Publications (2006).
- [2] Emily M. Livingston and Linda S. Siegel. *Developmental dyslexia: emotional impact and consequences*. *Australian Journal of Learning Difficulties* (2018).
- [3] Tore Angell Petersen. *App for Early Detection of Dyslexia* (2018).
- [4] Bjørnar H. Wold. *App for Early Detection of Dyslexia* (2016).
- [5] Thea Johansen and Maja Kirkerød. *Magno: An Application for Detection of Dyslexia - Dyslexia and Interface Design* (2017).
- [6] *Dysleksi Norge - Fagstoff*. URL <https://dysleksinorge.no/fagstoff/>, [Online], accessed: 28.11.2020.
- [7] U. Strehlow, R. Kluge, H. Möller and J. Haffner. *Long-term course of dyslexia beyond the school years: catamnesis from pediatric psychiatric ambulatory care*. *Zeitschrift für Kinder und Jugendpsychiatrie* (1992).
- [8] Jonathan Glazzard. *The impact of dyslexia on pupils' self-esteem. Support for Learning* (2010).
- [9] Dr Neil Humphrey. *Self-concept and self-esteem in developmental dyslexia*. *Journal of Research in Special Educational Needs (JORSEN)* (2004).
- [10] John Stein and Vincent Walsh. *To see but not to read; the magnocellular theory of dyslexia*. *Trends in Neurosciences* (1997).
- [11] Kaja Egset, Bjørnar H. Wold, John Krogstie and Hermundur Sigmundsson. *Magno App: Exploring Visual Processing in Adults with High and Low Reading Competence*. *Scandinavian Journal of Educational Research* (2019).
- [12] Bjarni Þorbjörnsson. private communication. Video meeting with deCODE, 01.10.2020.
- [13] John Brooke. *SUS: A Retrospective*. *Journal of Usability Studies* (2013).
- [14] Dysleksi Norge. *Dysleksi Norge - faglige retningslinjer*. Dysleksi Norge (2017). ISBN 978-82-90503-15-9.
- [15] G. Emerson Dickman, G. Reid Lyon, Jack Fletcher, Bennett, Sally Shaywitz, Susan Brady, Hugh Catts, Guinevere Eden, Jeffrey Gilger, Robin Morris, Thomas Viall and Harley Tomey. *IDA: Definition of Dyslexia* (2002). URL <https://dyslexiaida.org/definition-of-dyslexia/>, [Online], accessed: 05.12.2020.
- [16] Sir Jim Rose. *Identifying and Teaching Children and Young People with Dyslexia and Literacy Difficulties* (2009). URL <https://webarchive.org>.

- [nationalarchives.gov.uk/20130321060616/https://www.education.gov.uk/publications/eOrderingDownload/00659-2009DOM-EN.pdf](https://www.nationalarchives.gov.uk/publications/eOrderingDownload/00659-2009DOM-EN.pdf), [Online], accessed: 05.12.2020.
- [17] BDA. *Definition of Dyslexia* (2010). URL <https://www.bdadyslexia.org.uk/news/definition-of-dyslexia>, [Online], accessed: 05.12.2020.
- [18] Schulte-Körne G., Deimel W., Jungermann M. and Remschmidt H. *Follow-up of a sample of children with reading-spelling disorders in adulthood. Zeitschrift für Kinder- und Jugendpsychiatrie und Psychotherapie* (2003).
- [19] Johannes Schumacher, Per Hoffmann, Christine Schmä, Gerd Schulte-Körne and Markus M. Nöthen. *Genetics of dyslexia: the evolving landscape. Journal of Medical Genetics* (2007).
- [20] Mike Battista. *What is the Digit Span test?* URL <http://help.cambridgebrainsciences.com/en/articles/624895-what-is-the-digit-span-test>, [Online], accessed: 04.12.2020.
- [21] John Stein. *The current status of the magnocellular theory of developmental dyslexia. Neuropsychologia* (2019).
- [22] R.L. Peterson, B.F. Pennington, R.K. Olson and S.J. Wadsworth. *Longitudinal stability of phonological and surface subtypes of developmental dyslexia. Scientific Studies of Reading* (2014).
- [23] Trichur R. Vidyasagar. *Reading into neuronal oscillations in the visual system: implications for developmental dyslexia. Frontiers in Human Neuroscience* (2013).
- [24] M.S. Livingstone, G.D. Rosen, F.W. Drislane and A.M. Galaburda. *Physiological and anatomical evidence for a magnocellular deficit in developmental dyslexia. Proc. Natl. Acad. Sci.* (1991).
- [25] M. Giraldo-Chica, J.P. Hegarty and K.A. Schneider. *Neuroimage: Clinical* (2015).
- [26] G.F. Eden, J.W. VanMeter, J.M. Rumsey, J.M. Maisog, R.P. Woods and T.A. Zeffiro. *Abnormal processing of visual motion in dyslexia revealed by functional brain imaging. Nature* (1996).
- [27] G. A. Gescheider, J. M. Thorpe, J. Goodarz and S. J. Bolanowski. *The effects of skin temperature on the detection and discrimination of tactile stimulation. Somatosensory Motor Research* (1997).
- [28] K. Britten, M. Shadlen, W. Newsome, J. Movshon, Britten, Shadlen and Movshon. *The analysis of visual motion: a comparison of neuronal and psychophysical performance. Journal of Neuroscience* (1992).
- [29] P. Cornelissen, A. Richardson, A. Mason, S. Fowler and J. Stein. *Contrast*

- sensitivity and coherent motion detection measured at photopic luminance levels in dyslexics and controls. Vision-Res.* (1995).
- [30] A.L. Downie, L.S. Jakobson, V. Frisk and I. Ushycky. *Periventricular brain injury, visual motion processing, and reading and spelling abilities in children who were extremely low birthweights. J. Int, Neuropsychol. Soc.* (2003).
- [31] J. Skoyles and B.C. Skottun. *On the prevalence of magnocellular deficits in the visual system of non-dyslexic individuals. Brain and language* (2004).
- [32] Peter C. Hansen, John F. Stein, Sam R. Orde, Jonathan L. Winter and Joel B. Talcott. *Are dyslexics' visual deficits limited to measures of dorsal stream function? Neuroreport* (2001).
- [33] Mario Zechner. URL <https://libgdx.com/>, [Online], accessed: 09.12.2020.
- [34] Luz Rello and Ricardo Baeza-Yates. *Good fonts for dyslexia. ASSETS '13: Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility* (2013).
- [35] Sivananda Rajananda, Hakwan Lau and Brian Odegaard. *A Random-Dot Kinematogram for Web-Based Vision Research. Journal of Open Research Software* (2018).
- [36] Josh de Leeuw. *jsPsych*. URL <https://github.com/jspsych/jsPsych>, [Online], accessed: 06.21.2021.
- [37] deCODE. *Svipgerð.is, deCODE's application platform*. URL <https://www.svipgerd.is/>, [Online], accessed: 08.12.2020.
- [38] Bjarni Þorbjörnsson. private communication. Email received by Fredrik Jenssen, 08.12.2020.
- [39] *ISO 25010: Software Quality Model*. URL <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?start=0>, [Online], accessed: 31.05.2021.
- [40] Scrum.org. URL <https://www.scrum.org/resources/what-is-scrum/>, [Online], accessed: 08.12.2020.
- [41] Morten Sieker Andreasen, Henrik Villemann Nielsen, Simon Ormholt Schrøder and Jan Stage. *What happened to remote usability testing?: an empirical study of three methods* (2007).
- [42] Anders Bruun, Peter Gull, Lene Hofmeister and Jan Stage. *Let your users do the testing: a comparison of three remote asynchronous usability testing methods* (2009).
- [43] Jeff Sauro. *A Practical Guide to the System Usability Scale: Background, Benchmarks Best Practices*. CreateSpace Independent Publishing Platform (2011). ISBN 1461062705.

- [44] Thomas S. Tullis and Jacqueline N. Stetson. *A comparison of questionnaires for assessing website usability. Usability professional association conference. Vol. 1* (2004).
- [45] Brendan Eich. *JavaScript*. URL <https://no.wikipedia.org/wiki/JavaScript>, [Online], accessed: 21.06.2021.
- [46] Microsoft. *TypeScript*. URL <https://www.typescriptlang.org/>, [Online], accessed: 21.06.2021.
- [47] Alexandre Vinet. *AngularJS + TypeScript*. URL <https://www.spiria.com/en/blog/web-applications/angularjs-typescript/>, [Online], accessed: 03.12.2020.
- [48] Goodboy Digital and PixiJS community. *PixiJS*. URL <https://www.pixijs.com/>, [Online], accessed: 07.12.2020.
- [49] Sebastian McKenzie. *Babel*. URL <https://babeljs.io/blog/>, [Online], accessed: 21.06.2021.
- [50] Devon Govett. *Parcel*. URL <https://v2.parceljs.org/>, [Online], accessed: 21.06.2021.
- [51] Node.js Foundation and Ryan Dahl. *Node.js*. URL <https://nodejs.org/>, [Online], accessed: 07.12.2020.
- [52] Isaac Z. Schlueter and Inc. npm. *Node Package Manager (NPM)*. URL <https://www.npmjs.com/>, [Online], accessed: 07.12.2020.
- [53] Adriano Raiano and Jan Mühlenmann. *i18next*. URL <https://www.i18next.com/>, [Online], accessed: 21.06.2021.
- [54] Junio Hamano et al. *Git*. URL <https://git-scm.com/>, [Online], accessed: 07.12.2020.
- [55] Tom Preston-Werner, Chris Wanstrath, P. J. Hyett and Scott Chacon. *GitHub*. URL <https://www.github.com>, [Online], accessed: 04.06.2020.
- [56] Microsoft. *GitHub Pages*. URL <https://pages.github.com/>, [Online], accessed: 04.06.2020.
- [57] Tim Schaub. *gh-pages*. URL <https://github.com/tschaub/gh-pages>, [Online], accessed: 04.06.2020.
- [58] Microsoft. *Visual Studio Code*. URL <https://www.code.visualstudio.com/>, [Online], accessed: 04.06.2020.
- [59] Joel Spolsky and Michael Pryor. *Trello*. URL <https://www.trello.com/>, [Online], accessed: 04.06.2020.
- [60] University of Oslo. *Nettskjema*. URL <https://nettskjema.no>, [Online], accessed: 04.06.2020.

- [61] *A Detailed Explanation of JavaScript Game Loops and Timing*. URL <https://www.isaacsukin.com/news/2015/01/detailed-explanation-javascript-game-loops-and-timing>, [Online], accessed: 08.06.2021.
- [62] *MainLoop.js*. URL <https://github.com/IceCreamYou/MainLoop.js>, [Online], accessed: 08.06.2021.
- [63] OECD. *Skills Matter: Further Results from the Survey of Adult Skills*. OECD Publishing, Paris (2016). ISBN 978-92-64-25805-1. (page 79).
- [64] Jakob Nielsen. *Middle-Aged Users' Declining Web Performance*. *Jakob Nielsen's Alertbox*. URL <https://www.nngroup.com/articles/middle-aged-web-users/>, [Online], accessed: 18.06.2021.

Appendix

A Previous requirements for Magno

Id	Description	Priority
FR1	The application must use screen size, screen resolution and viewing distance to calculate the size of objects on the screen.	High
FR2	It should be possible to tune the behavior of the tests through a settings-Screen.	High
FR3	Due to the number of settings parameters, default values are to be provided.	Medium
FR4	Settings are to be stored for the next time the application runs.	High
FR5	The application should calculate a threshold score after a successful test run.	High
FR6	Test results along with settings used during a test should be possible to share.	Low
NFR1	The application should be able to run close to 60 frames per second on any device.	Medium
NFR2	A comprehensive user guide should be made, describing test procedures and settings.	Medium

Table 7: Requirements from the first iteration of Magno [4].

Id	Description	Priority
FR1	The application should use the screen size to make the design responsive to fit to different types of tablets and computers.	High
FR2	A tutorial must be implemented in the application to make it possible to use the application without any previous knowledge of the application.	High
FR3	The score view should contain an explanation of the threshold score.	High
FR4	The system should include descriptions that indicate that the user must click the boxes when taking motion, form fixed or form random test.	High
FR5	It should be possible to cancel in the middle of a test without using the computers' keyboard or the back button on an Android tablet.	High
FR6	On desktop versions of the application, the cursor should change form when hovering over clickable content.	High
FR7	The "Settings" view in the application should sort different settings options under logical names.	Medium
FR8	It should be possible to start a test without completing a tutorial.	Medium
FR9	It should be possible to use the application without a supervisor.	Low
FR10	The application should be able to store test results, for use in comparisons based on age groups.	Low
NFR1	It should be possible to reach any given system function from the main view within 3 clicks.	N/A
NFR2	A user should be able to understand the application within 3 minutes.	N/A
NFR3	The application should have a minimum SUS score of 80.	N/A

Table 8: Requirements from the second iteration of Magno [5].

Id	Description	Priority
FR1	The database should store the information needed by the stakeholders	High
FR2	The system should connect to the rest-API seamlessly	High
FR3	The rest-API should collect data and input it to the database	High
FR4	The system should be able to store test results for use in comparisons based on age groups and sex	Medium
FR5	The system should be able to differentiate the different age groups set by the shareholders	Medium
FR6	It should be possible to use and navigate the web page with little to none prior introduction	Medium
FR7	The web page should be able to show the research information based on a selected age group	Medium
FR8	The system should be able to export data to an excel file with some given parameters	Low
NFR1	The system should be available to receive data from the application Magno 99.9% of the day	N/A
NFR2	The system should be running in less than 20 min after any major incident	N/A
NFR3	A user should be able to get research data 99.9% of the day	N/A
NFR4	The system should only show information to authorized users	N/A
NFR5	The system should not store any personal information about the user	N/A
NFR6	The system should be able to handle 50 simultaneous clients	N/A
NFR7	The administrator should be able to scale-up the system to handle 100 simultaneous clients	N/A

Table 9: Requirements from the third iteration of Magno [3].

B Default Setting Values

```

1 //Screen default values
2 private static final int SCREEN_W_MM_DEF_VALUE =

```

```

3      (Gdx.app.getType().equals(Application.ApplicationType.
Desktop)) ?
4          (int)(Gdx.graphics.getDesktopDisplayMode().width /
(Gdx.graphics.getPpcX() / 10)) :
5          (int)(Gdx.graphics.getWidth() / (Gdx.graphics.
getPpcX() / 10));
6
7  private static final int SCREEN_H_MM_DEF_VALUE =
8      (Gdx.app.getType().equals(Application.ApplicationType.
Desktop)) ?
9          (int)(Gdx.graphics.getDesktopDisplayMode().height /
(Gdx.graphics.getPpcY() / 10)) :
10         (int)(Gdx.graphics.getHeight() / (Gdx.graphics.
getPpcY() / 10));
11
12 private static final int SCREEN_W_PX_DEF_VALUE =
13     (Gdx.app.getType().equals(Application.ApplicationType.Desktop))
?
14         Gdx.graphics.getDesktopDisplayMode().width :
15         (Gdx.graphics.getWidth());
16
17 private static final int SCREEN_H_PX_DEF_VALUE =
18     (Gdx.app.getType().equals(Application.ApplicationType.
Desktop)) ?
19         Gdx.graphics.getDesktopDisplayMode().height :
20         (Gdx.graphics.getHeight());
21
22 private static final int SCREEN_VD_MM_DEF_VALUE = 300;
23 private static final int SCREEN_PATCH_WIDTH_DEF_VALUE = 10;
24 private static final int SCREEN_PATCH_HEIGHT_DEF_VALUE = 14;
25 private static final int SCREEN_PATCH_GAP_DEF_VALUE = 5;
26
27 //Dot behavior default values
28 private static final int DOT_MAX_AMOUNT_DEF_VALUE = 300;
29 private static final float DOT_RADIUS_PIXELS_DEF_VALUE = 1f;
30 private static final float DOT_SPACING_DEF_VALUE = 1f;
31 private static final float DOT_VELOCITY_DEF_VALUE = 50f;
32 private static final float DOT_COHERENCY_DEF_VALUE = 50f;
33 private static final float DOT_ANIMATION_TIME_DEF_VALUE = 5f;
34 private static final float DOT_MAX_ALIVE_TIME_DEF_VALUE = 0.085f;
35 private static final int DOT_TO_KILL_DEF_VALUE = 10;
36 private static final float DOT_HORIZONTAL_REVERSAL_TIME_DEF_VALUE =
0.572f;
37 private static final float DOT_RANDOM_DIRECTION_TIME_DEF_VALUE =
0.572f;
38
39 //Form behaviour default values
40 private static final boolean FORM_AUTO_MODE_DEF_VALUE = true;
41 private static final int FORM_MAX_AMOUNT_DEF_VALUE = 600;
42 private static final float FORM_DIAMETER_WB_DEF_VALUE = 8f;
43 private static final int FORM_CIRCLES_DEF_VALUE = 4;
44 private static final float FORM_CIRCLES_GAP_DEF_VALUE = 0.9f;
45 private static final float FORM_LINE_LENGTH_DEF_VALUE = 0.4f;
46 private static final float FORM_LINE_HEIGHT_DEF_VALUE = 1f;
47 private static final float FORM_LINE_GAP_DEF_VALUE = 0.4f;
48 private static final float FORM_COHERENCY_DEF_VALUE = 100f;
49 private static final float FORM_F_DETECTION_TIME_DEF_VALUE = 4f;
50 private static final float FORM_R_DETECTION_TIME_DEF_VALUE = 1000f;

```



```

51
52 //Staircase default values
53 private static final float STAIR_CORRECT_ANSWER_DB_DEF_VALUE = 1f;
54 private static final float STAIR_WRONG_ANSWER_DB_DEF_VALUE = 3f;
55 private static final int STAIR_MAX_TRIES_DEF_VALUE = 100;
56 private static final int STAIR_REVERSAL_POINTS_DEF_VALUE = 10;
57 private static final int STAIR_REVERSALS_TO_USE_DEF_VALUE = 8;
58
59 //Input default values
60 private static final int INPUT_KEY_LEFT_DEF_VALUE = Input.Keys.S;
61 private static final int INPUT_KEY_RIGHT_DEF_VALUE = Input.Keys.L;
62 private static final int INPUT_KEY_BACK_DEF_VALUE = Input.Keys.
    ESCAPE;
63 private static final boolean INPUT_CONTINUOUS_MODE_DEF_VALUE = true
    ;

```

C Example Test Results

C.1 Motion Test Results Example

```

1 {
2 correctAnswers: 23
3 wrongAnswers: 6
4 threshold: 17.167208
5 lowestCoherency: 11.890287
6 reversalValues: [
7   1: 13.608927
8   2: 19.22312
9   3: 13.010082
10  4: 25.958527
11  5: 11.890287
12  6: 16.795477
13  7: 14.746119
14  8: 20.829447
15  9: 16.056412
16 10: 22.680285
17 ]
18 screenSettings: [
19   screen_w_mm: 508
20   screen_h_mm: 285
21   screen_w_px: 1920
22   screen_h_px: 1080
23   viewing_distance: 300
24   patch_width: 10
25   patch_height: 14
26   patch_gap: 5
27 ]
28 motionSettings: [
29   dot_amount: 300
30   dot_radius: 1.0
31   dot_spacing: 1.0
32   dot_velocity: 80.0
33   dot_coherency: 50.0
34   dot_animation_time: 5.0
35   dot_max_life_time: 0.085
36   dot_kill_percentage: 10

```

```

37 dot_horizontal_reversal_time: 0.572
38 dot_random_direction_time: 0.572
39 ]
40 formSettings: [
41   form_auto_mode: false
42   form_line_amount: 600
43   form_diameter_wb: 12.0
44   form_nr_of_circles: 4
45   form_circle_gap: 0.9
46   form_line_length: 0.4
47   form_line_height: 1.0
48   form_line_gap: 0.4
49   form_coherency: 100.0
50   form_f_detection_time: 4.0
51   form_r_detection_time: 1000.0
52 ]
53 staircaseSettings: [
54   stair_correct_db: 1.0
55   stair_wrong_db: 3.0
56   stair_max_tries: 100
57   stair_reversal_points: 10
58   stair_mean_from_last: 8
59 ]
60 inputSettings: [
61   input_key_left: A
62   input_key_right: S
63   input_key_back: Escape
64   input_continuous_mode: true
65 ]
66 }

```

C.2 Form Test Results Example

```

1 {
2   correctAnswers: 33
3   wrongAnswers: 5
4   threshold: 6.0648336
5   lowestCoherency: 4.0231066
6   reversalValues: [
7     1: 8.437666
8     2: 11.91852
9     3: 5.459271
10    4: 7.7114253
11    5: 4.0231066
12    6: 5.6827893
13    7: 4.989384
14    8: 7.047693
15    9: 6.187744
16    10: 8.740421
17 ]
18 screenSettings: [
19   screen_w_mm: 508
20   screen_h_mm: 285
21   screen_w_px: 1920
22   screen_h_px: 1080
23   viewing_distance: 300
24   patch_width: 10
25   patch_height: 14

```

```

26   patch_gap: 5
27 ]
28 motionSettings: [
29   dot_amount: 300
30   dot_radius: 1.0
31   dot_spacing: 1.0
32   dot_velocity: 80.0
33   dot_coherency: 50.0
34   dot_animation_time: 5.0
35   dot_max_life_time: 0.085
36   dot_kill_percentage: 10
37   dot_horizontal_reversal_time: 0.572
38   dot_random_direction_time: 0.572
39 ]
40 formSettings: [
41   form_auto_mode: false
42   form_line_amount: 600
43   form_diameter_wb: 12.0
44   form_nr_of_circles: 4
45   form_circle_gap: 0.9
46   form_line_length: 0.4
47   form_line_height: 1.0
48   form_line_gap: 0.4
49   form_coherency: 100.0
50   form_f_detection_time: 4.0
51   form_r_detection_time: 1000.0
52 ]
53 staircaseSettings: [
54   stair_correct_db: 1.0
55   stair_wrong_db: 3.0
56   stair_max_tries: 100
57   stair_reversal_points: 10
58   stair_mean_from_last: 8
59 ]
60 inputSettings: [
61   input_key_left: A
62   input_key_right: S
63   input_key_back: Escape
64   input_continuous_mode: true
65 ]
66 }

```

D SUS Form

1. I think I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Figure 51: Original SUS form.

E Usability Test Email

My master student Fredrik Jenssen has made a web-app (Magno) for self-testing of an aspect of visual processing.

The app provides a motion test intended for use in detection of probability of having dyslexia.

We here want to evaluate the usability of the app, and would like to get help from you in connection to that. The task consists of two steps:

1. Visit «website» and follow the instructions to take the motion test. You will receive a score, but that score is not used further
2. Then complete a survey on your experience on using the app at «website»

The test and evaluation will take around 15 minutes in total.

No data is recorded to link you to the use of the app or what you fill in the form.

Thank you for your help, and have a nice day.

Best regards, John Krogstie

F First Usability Test: Free Text Answers

User Id	Was there anything in the application that was particularly difficult to understand?	Do you have any comments on the design of the application?
13748216	It was easy in the beginning, and then hard to keep them apart. Very confusing.	N/A
13748888	Status of the process	Provide feedback on the state of the process
13749785	Mostly everything was quite clear. Two points: It was not clear if the time mattered in the task. That is - precision versus speed "heuristic" was not elaborated on. Is it better to guess if one is almost certain what to pick, or, should one use up ALL 5 seconds and then pick what one was almost sure earlier. The size of the boxes will depend on the screen size and the resolution used - it was not clear if magnifying the browser window will affect the test or not.	In the tutorial, it just happened that I had to press left arrow for ALL the tests (which were all scored as Correct). That was somewhat biasing me later, in the cases where I was not sure, I tended to pick left also - for no logical reason - just being "primed" with ALL LEFT in the tutorial. Perhaps make sure that the tutorial balances left-right answers if the "left" bias was side effect of some pseudo-randomness.
13786095	N/A	I feel like the instructions should have mentioned that the task difficulty will vary compared to the tutorial examples. I was very surprised I had to guess on half of the tasks when the tutorial was so easy. Survey: I didn't use a tablet or a desktop - I used a laptop.
13786599	No.	Easy to understand. Only problem is that that the patterns disappear after a short time.
13789587	It would be nice to have more examples of what is the correct box	it is simple and understandable. One idea is to have more interactive or less text and more visual presentation with instructions on how to use it.
13792522	no, everything was understandable.	easy and straightforward to use.

13796068	none	The patterns vanished to quickly
13796801	N/A	The examples in the tutorial were very straightforward/easy compared to the second half of the actual test. I would like to have seen at least one difficult example before taking the test itself.
13798018	N/A	made me sick ;)
13876154	The task was a bit ambiguous. What is exactly meant by "the dots moving systematically back and forth"? Which dots? Not all of them move similarly in a given box. And in which direction? Is "back and forth" from left to right, or from top to bottom? Otherwise, on the technical side, it was super simple to use.	N/A
13901748	The sitting arrangement was demonstrated with a desktop computer. There was no demonstration for Tablet users	A design suitable for tablet users. I thought i was in the example experiment not knowing i already transitioned to the actual experiment. There should be a clear message to the user during the transition to the actual experiment. The text read TRIAL and this seems like an example experiment before the actual experiment
13908231	N/A	Some issues with the text displaying on my Chrome browser. Could be related to the aspect ratio of the window I opened but the text was written in full size and did not seem to adapt to change of window's size.
13908425	No, all the instructions were very clear.	I think the design is well structured, easy to use and clear!

13908505	What was the mentioned 'application', or 'system', or intended use? Does this survey relate to the visual test at all?	I did not see an application, I saw two boxes with hardly discernible content, and pushed left- or right-arrow according to the instruction/tutorial (horizontal- or not moving particles) Most of the time I didn't see much difference of movement between the boxes.
----------	--	---

Table 10: All textual feedback received from the first usability test. The motion test was tested.

G Second Usability Test: Free Text Answers

User Id	Was there anything in the tutorial or test that was particularly difficult to understand?	Do you have any comments on the design of the tutorial or test?
13976438	No	Easy to use and self explanatory
13977897	N/A	I think many of the questions in the survey were somewhat vague. I mean, what is "the system" referred to in many of the questions?
13978288	<i>Translated from Norwegian:</i> The pictures turned black pretty quickly. What is the consequence of clicking a black picture?	<i>Translated from Norwegian:</i> A progress bar which shows how far you have progressed / how much of the test is left.
13978844	Not really except when I swear there was no circle I just randomly picked and I sort of wish there was a 'no circle' option maybe by pressing down. Not sure if that works in your circle/dislexia theory though.	It could be interesting to explain or have a learn more section to connect the identification of the circles with dislexia.

13984125	yes - I was confused about which part was the tutorial - thought I was doing the actual test when I was still in the tutorial.	looks like it could be a good tool. But, I was a bit confused doing the test, which maybe was due to programming rather than design - towards the end the images disappeared before I had answered/-chosen a box. And the screen still said 'choose a box', and did not move on until I had done so. Also, in some of the images I was wondering whether the circle should/-could be off center.
14125549	No, everything was perfectly clear	Maybe I wouldn't include at the beginning of the test the information about the "best score". I would leave out the part "where 1 is the best possible score". If a person who would do the test had dyslexia, maybe they would feel bad about not getting "the best possible score".
14125637	N/A	The hard tasks feel very random, they don't make sense, which doesn't feel very good. I understand that the difficulty is supposed to be increasing, but I could imagine a less experienced person being unsure if everything was working correctly when none of the boxes have circular patterns.
14139073	No.	After a few iterations I could really NOT see any circular patterns, so I was selecting randomly. I would expect instructions on what to do in such a case to exist in the tutorial.

Table 11: All textual feedback received from the second usability test. The form fixed test was tested.

