Master's thesis

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

Elias Sagmo Larsen
Torkjell Romskaug

Real time stress-aware feedback system for programming.

Master's thesis in Master of Science in Informatics
Supervisor: Kshitij Sharma
June 2022

NTNU
Kunnskap for en bedre verden

Elias Sagmo Larsen
Torkjell Romskaug

Real time stress-aware feedback system for programming.

Master's thesis in Master of Science in Informatics
Supervisor: Kshitij Sharma
June 2022

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

NTNU

Kunnskap for en bedre verden

Abstract

In this master thesis we implement a real time system of continuous feedback based on students predicted cognitive load. The system was tested on two groups of students in a within subject experiment. The participants (n=26) was first introduced to a pretest and subsequently tasked with debugging two games with varying degrees of difficulty, they had a total of 15 minutes on each task. The pretest was used to split the participants into two groups based on their level of expertise. To compute cognitive load, pupillary data from participants was collected using an eye tracker. Providing students with feedback while programming did not increase their cognitive load. Regardless of expertise, receiving feedback had a positive effect on performance on the harder task and had no effect on the easier task. Experts had a trend towards having lower cognitive load than Novices when programming, especially on easier tasks. The results of this experiment are inline with the previous research done on the subject and heavily implies the feasibility of wearable based feedback system for programming.

Oppsummering

I denne masteroppgaven implementerer vi et sanntidssystem med kontinuerlig tilbakemelding basert på studentenes predikerte kognitive belastning. Systemet ble testet på to grupper elever i et mellom subjekt fageksperiment. Deltakerne (n=26) ble først introdusert til en forhåndstest og fikk deretter i oppgave å feilsøke to spill med ulik vanskelighetsgrad, de hadde totalt 15 minutter på hver oppgave. Forhåndstest ble brukt til å dele deltakerne inn i to grupper basert på deres kompetansenivå. For å beregne kognitiv belastning ble pupilldata fra deltakerne samlet inn ved hjelp av en øyesporer. Å gi elevene tilbakemelding under programmering økte ikke deres kognitive belastning. Uavhengig av kompetanse hadde tilbakemelding en positiv effekt på prestasjonen på den vanskeligere oppgaven og hadde ingen effekt på den lettere oppgaven. Eksperter hadde en trend mot å ha lavere kognitiv belastning enn nybegynnere ved programmering, spesielt på enklere oppgaver. Resultatene av dette eksperimentet er i tråd med tidligere forskning gjort på emnet og antyder i stor grad gjennomførbarheten av et smart bærbart tilbakemeldingssystem for programmering.

Acknowledgments

We want to say thank you to our advisor Kshitij Sharma, who with his insight and knowledge has helped us immensely. Additionally we want to thank William, Casper and Petter who had to endure this last year with us at the study hall, Gamle fysikk 353.

Table of Contents

List of Figures **ix**

List of Tables **xi**

1 Introduction **1**

1.1 Motivation 1

1.2 Research Questions 2

1.3 Terminology 2

1.4 Outline 3

2 Theory & related work **5**

2.1 Cognitive load and Stress 6

2.2 Feedback 9

2.3 Experts and Novices 10

3 Research design **13**

3.1 Experiment 14

3.2 Analysis 17

4 Implementation and Data Processing **21**

4.1 Eye tracking 22

4.2 Cognitive Load Estimation & Data Processing 23

4.2.1 Real time Data Processing 24

4.2.2 LHIPA Calculation . . . 25
4.2.3 Mean LHIPA and cognitive load calculation . . . 26
4.3 VSCode Extension . . . 27

5 Results **29**
5.1 Performance . . . 30
5.2 Cognitive load . . . 33
5.3 Perceived mental load (Nasa TLX) . . . 36
5.4 Interactions and number of feedbacks . . . 38

6 Discussion & Limitations **41**
6.1 Findings . . . 41
6.2 Validity of Cognitive load Measurements . . . 42
6.3 Feedback - [**RQ1**] . . . 42
6.3.1 Continuous feedback . . . 43
6.4 Expertise & Cognitive load - [**RQ2**] . . . 44
6.5 Limitations & future work . . . 44

7 Conclusion & Contribution **47**

Bibliography **49**

A Adjusted LHIPA code **53**

B NSD **57**

C Consent form **67**

D Pretest **71**

E Snake debugging task with hints **79**

F Tetris debugging task with hints **83**

List of Figures

3.1 Research process design 14

3.2 Test environment used during experiments 16

4.1 Implementation architecture 22

4.2 Tobii Glasses 2 Eye Tracker Wearable System Tobii I by Tobii Technology is licensed under Attribution 3.0 Unported. 23

4.3 Types of feedback shown to the participant during debugging tasks 28

5.1 Performance distribution on the pretest. 30

5.2 Performance distribution for Snake and Tetris debugging tasks. The Tetris performance is the adjusted Tetris performance. 30

5.3 Performance on Snake and Tetris tasks by expertise. 31

5.4 Sum of Snake and Tetris performance grouped by expertise. 32

5.5 Performance on Snake and Tetris by whether or not the participant was given feedback on the task. 32

5.6 Performance when the participant received feedback independent of which task the feedback was received. 33

5.7 Cognitive load for pretest, Snake task and Tetris task. 33

5.8 Cognitive load by expertise. 34

5.9 Average cognitive load by expertise for both Snake and Tetris. 35

5.10 The participants mean cognitive load by feedback or not. 35

5.11 Cognitive load grouped by receiving of feedback. Independent of which debugging task the participant had feedback on. 36

5.12 Mental demand as reported by a NASA TLX form compared by if the student is classified as an Expert or Novice for respectively the pretest, Snake and Tetris. . . 37

5.13 The self reported mental demand compared on Snake and Tetris tasks by whether or not they received feedback on the task. 38

List of Tables

3.1 Generated participant files 16

5.1 Cognitive load mean scores. 34

5.2 NASA-TLX mean scores. 36

5.3 Interactions with feedback. 38

5.4 Amount of feedback given 39

Chapter 1

Introduction

1.1 Motivation

The use of technology and its applications in everyday life has exploded in the previous several decades. With technology also comes distractions and with distraction, the technology can become less effective. Cognitive load theory is a way of thinking about human memory that is comparable to the way ram works in computers. Cognitive load is the number of elements you have in your memory at one time [1]. Because the human brain's working capacity is limited, it's critical not to overburden it with options and distractions. One could create more effective systems by combining cognitive load theory and technology. Previously, the two most prominent methods for calculating cognitive load were time-consuming surveys [2] that asked individuals about their load or intrusive EEG caps that monitored brainwaves [3]. New research published in the last few years have outlined a method for predicting cognitive load that is significantly less tiresome, intrusive, and, most importantly, far less expensive and time consuming [4]–[6].

We can predict the learners cognitive workload by using portable eye-trackers and wristbands to measure the learners physiological signals, such as pupil dilation [1], [4]–[8]. The application of such a system could be massive and help in an array of areas, raising awareness of the things that are clogging up memory, to make space for the things that matters. With knowledge of a user's cognitive workload, better solutions might be designed to avoid overwhelming the user and help them stay focused. Another research area where cognitive workload could be used is in learning systems as e-learning, giving the teacher or student more tools at their disposal.

The state of the art cognitive load estimation systems have yet to be tested in a programming setting, and has to the best of our knowledge not been used to give feedback in real time. We want to take use of the possibility of real time cognitive load estimation and test how well feedback works when given in such a system. We propose a system that take advantage of the state of the

art theory to give students feedback based on their cognitive load at real time during programming. Our novel system will try to apply the eye tracking measure proposed in Duchowski *et al.* [6] and Duchowski *et al.* [4] for timing feedback to the programmers need without any human intervention.

1.2 Research Questions

With a system that can give the relative cognitive load of the user with little intervention, we wanted to investigate several important aspects. We want to investigate the effects of expertise, which is a key factor in working memory, to see if there is a difference in the effect of our system based on their prior knowledge. Based on this motivation our research questions(**RQs**) are as follows:

- ***RQ1*** *What is the effect of continuous feedback timed with cognitive load during programming tasks?*
- ***RQ2*** *What is the effect of expertise on cognitive load during programming tasks?*

With the use of the following research objectives, this thesis will attempt to address the research questions.

- **RO 1** *Design and develop a system for feedback during programming.*
- **RO 2** *Research, design and develop a cognitive load estimation system for real time cognitive load estimation.*
- **RO 3** *Perform an experiment with the developed system to evaluate its effect.*
- **RO 4** *Analyze the resulting data from the experiment.*

1.3 Terminology

Pupil dilation is how much, in millimeter, the pupils are dilated. Change in pupil diameter is often used as a metric to estimate cognitive load.

Cognitive load is related to the brains working capacity, and we define it as the amount of information the brain can process/is processing at any given moment.

Real time is a term meaning something happening at the same or close to the same time as an event occurs.

Debugging is the process of finding and removing errors, also called bugs, in software code that can cause it to behave unpredictably or crash.

1.4 Outline

Section 1: Introduction

The introduction introduces the Motivation, Research Questions, Terminology and the Outline.

Section 2: Theory & related work

In theory & related work we outline knowledge and previous studies on feedback, cognitive load and stress as well as look into the differences between Experts and Novices.

Section 3: Research Design

In Research Design we outline how we will execute the experiment and how we will analyze the data.

Section 4: Implementation and Data Processing

In Implementation and Data Processing the stress aware feedback system is described. We describe the required functionality, the system architecture, and how we implemented the specific parts of the architecture.

Section 5: Results

In results we present the analyzed results of the experiments in text, graphs and tables.

Section 6: Discussion & Limitations

In discussion the main findings are summarized, interpreted, and implications and limitations discussed.

Section 7: Conclusion & Contribution

In conclusion & Contribution we conclude and summarize our contributions.

Chapter 2

Theory & related work

In this chapter, the theory and related works about cognitive load, stress and feedback is presented and explained. It is necessary to clarify concepts for the rest of the thesis. The studies presented here are selected from a rigid set of criteria.

2.1 Cognitive load and Stress

How we measure and define cognitive load is important to make clear as it should not be ambiguous. Cognitive load is defined in the following way: The amount of information the brain can process at any given moment [1]. Measurements of the ground truth cognitive load is often measured with questionnaires, i.e., rating scales, with the most common being the 9 point scale [9], [10]. Another frequently used measurement is the NASA Task Load Index (NASA TLX) [2]. One of the main advantages of questionnaires is that they provide valid information on an individual level. In addition, they are very simple to implement, and a lot of people can use the same questionnaire. There are some small problems with the questionnaires. One of them is that they are not completely accurate, and they do not differentiate between the different types of cognitive load. [11] Although questionnaires have some problems the biggest one might be that the person has to fill them in after they already have experienced the cognitive load, which leaves time to forget or miss-remember the correct load. To get better more nuanced measurement research has taken to measuring physiological data, for example eye tracking and EEG. [3], [11] Though there are many possible applications of cognitive load theory, we are going to focus on the physiological measurements and the subsequent feedback one could possibly give to students.

Cognitive load predictors

In recent years researchers have tried to predict cognitive load in both students and others to get more accurate and faster insights [3], [7], [11]. When researchers try to predict and understand cognitive load they are reliant on predictors to conduct their analysis. As talked about in Section 2.1, Cognitive load and Stress, the traditional method uses questionnaires. In the following section we explore other predictors which has been used to predict cognitive load. The focus is mainly on physiological predictors, specifically eye-tracking measures, EDA (electrodermal activity), skin temperature, and heart rate measures. A predictor is a variable that maps to the target variable. In machine learning, a predictor is the input variable to the machine learning model, which in our case is an eye-tracker and a wristband. The output variable is the target variable, which in our case is either stress or cognitive load. The next paragraphs will contain information on all the predictors previously mentioned with what they can and have been used to predict.

Eye-tracking has been shown to be a good tool for predicting cognitive load [8], [12]–[15]. Tobii[1] explains eye tracking as a sensor device that can identify a person's presence and monitor their gaze in real time. The system turns eye movements into a data stream that includes pupil position, gaze vectors for each eye, and gaze point information. Essentially, the technology decodes eye movements and converts them into insights that may be utilized in a variety of applications or as a second mode of input. Eye-tracking provides the means to extract several features, including

[1] https://www.tobii.com/group/about/this-is-eye-tracking/

pupil diameter, blink latency, eye fixations, and saccade characteristics.

Pupil dilation is one feature from eye-tracking that seems to be good for measuring cognitive load. [12]–[15] It has been studied in several studies over several decades. [13], [14]. Krejtz *et al.* [12], conducted a study where participants did tasks of varying difficulty while tracking their eyes. To track the eyes, the participants had to keep their head stabilized on a chin rest. They tracked saccades and pupil diameter. The research focused on measuring the averages of the whole session, and did not try to predict the real time cognitive load. Their results shows that microsaccade magnitude and change in pupil diameter maps to task difficulty, and thus inadvertently cognitive load. Additionally, a study by Wel and Steenbergen [13], did a meta-review on pupil dilation as an index of effort in cognitive control tasks. They found that an increase in task demand leads to higher pupil dilation. Pupil dilation does at times predict improved task performance, because more effort can lead to better results.

Going all the way back to 1996, a study by Granholm *et al.* [14] found that task-evoked pupillary responses are a reliable and precise measure of cognitive load. They examined 22 undergrads during a digit span recall task. They used 5, 9 and 13 digits per string (length of the number they had to remember), where 5 is low, 9 is high, and 13 is too excessive. The subjects pupillary responses increased with increased task difficulty. When the demand is below cognitive load limits, the authors suggest that pupillary responses increase methodically with increase in demand. When the limit is reached or the load is close to the limit, the responses change little, but when the cognitive load gets excessive the pupillary response starts to decline. Agreeing with Granholm *et al.* [14] a more recent study by Gollan *et al.* [15] combined the mean pupil diameter, saccade speed, standard deviation, and number of fixations, and demonstrated it to be a reliable and precise measurement of cognitive load.

As already mentioned in the previous paragraph, some studies used other features than just pupil diameter to measure cognitive load. One of these features is saccades, which was found to be a good predictor of cognitive load. [8], [12], [15] Andrzejewska and Skawińska [8] examined cognitive load in students. The authors gave the students either a logical debugging task or a syntactical debugging task. Students were given unlimited time and had to answer orally. The authors used a mounted eye tracker to measure intrinsic cognitive load. They did not look at continuous measuring cognitive load but instead looked at the cognitive load of the whole task compared to another easier task. To find a task's difficulty, the students rated their task. They rated the logical debugging task harder than the syntactical debugging task. To find the statistical significance of the physiological measures, they used eye-tracking measures and considered their averages. They found that SAA("the sum of all saccade amplitudes divided by the number of saccades in the trial") was statistically significant in relation to task difficulty.

More recent studies used pupil dilation and the frequency of pupil oscillations to create new features/ measures of cognitive load. [4], [6]. The Duchowski *et al.* [4] build upon the work of Marshall

[5] and created the index of pupillary activity (IPA). The IPA is capable of discriminating between task difficulty in respect to cognitive load, in experiments where participants do easy and difficult tasks. The IPA and the ICA are thought to be insensitive to lighting conditions, because they are based on moment to moment changes in pupil dilation. [6] However it is thought to be susceptible to the law of initial value, when the participants is seated in a room with very low light, and the pupil is already dilated. Duchowski *et al.* [6] shows that IPA is incapable of distinguishing between high and low cognitive load during n-back task, which they consider a more appropriate measurement of its manifestation. Another metric that claims to be more resistant to changes in light conditions is Duchowski *et al.* [6]'s Low High Index of Pupillary Activity (LHIPA). The LHIPA uses the same concept of pupillary oscillations, but considers the ratio of high and low frequency bands contained within the wavelet composition of pupil diameter signal. Through analysis of three experiments, they show the robustness of the LHIPA. They conclude that the LHIPA is not necessarily successful in distinguishing between levels of task difficulty, but that it is better than the IPA at calculating cognitive load. Their findings show that small angles off-axis distortion negatively affect IPA, but not LHIPA, making the LHIPA a more suitable fit when participants have to move their eyes freely. They also hypothesize the LHIPA's ability to estimate cognitive load in real time.

Because of the success of cognitive load measurements when utilizing eye trackers, Zagermann *et al.* [16] looked into the field of cognitive load and eye tracking to device a model for utilizing eye tracking with focus on Human Computer Interaction (HCI). They created the model with four eye-based indicators; Fixation(Number and duration), Saccade(Length, Angle and Velocity), Pupil(dilation) and Blink(Rate and Velocity). They then theorize that elements from HCI influence the eye-based indicators; Individual Interaction, Workflow, Environment, and Social Interactions and communication.

The aforementioned studies shows how eye-tracking can be beneficial for predicting cognitive load in students. Firstly, eye-tracking has successfully been used to map task difficulty with cognitive load. Therefore, we have reason to believe that as a task gets harder, or a student has high cognitive load, eye-tracking will be a good predictor and provide valuable data.

Wristbands is another tool that have been used to investigate cognitive load. Various measurements have been created and used to capture cognitive load. Haapalainen *et al.* [17] found that the electrocardiogram median absolute deviation and median heat flux (rate of heat transfer) measurements were the most accurate at distinguishing between low and high levels of cognitive load, providing a classification accuracy of over 80% when used together. In a more recent study by Gjoreski *et al.* [18], galvanic skin response (GSR) was found to demonstrate changes in germane cognitive load levels.

Stress predictors

Smartwatches and wristbands have been used to investigate physiological stress. The physiological measurements used from the smartwatches and wristbands are mainly EDA, skin temperature, and heart rate and have been used many times to measure physiological stress [19]–[21]. Kyriakou *et al.* [19] studied stressful situations in real-life. The researchers used galvanic skin response, a type of EDA, and skin temperature to measure stress. They developed a model that gained an accuracy of 84% when predicting moments of stress. Gjoreski *et al.* [20] conducted a study on five adults where they tried to detect stress using a commercial wristband. The adults logged their stress level on a smartphone, and their heart rate was measured during the experiment. Both the logged stress and heart rate was fed to a machine learning model. They conducted the study in a lab where they used stress-inducing techniques. Additionally, they tested in real-life scenarios over the course of 55 days. Their detection was significantly better when using context data like the watches' accelerator. The accuracy after collecting real-life data for 55 days, for a 2-class problem, was 92%.

One of the biggest advantages of smartwatches and wristbands are their unobtrusive nature. In contrast to EEG a wristband is invisible. They are also better suited, than other wearables, for collecting measurements when a study is conducted over a longer time period. They are better suited because the user is less obstructed. In addition they can give information at night. Eye-trackers don't give any information when the subject is sleeping, while a wristband or a smart-watch can continuously be on the body.

2.2 Feedback

Feedback is one of the most powerful tools educators has at hand to influence the learning of students, but the influence can be both bad and good and it varies how effective it is [22]. Wisniewski *et al.* [22] conducted a meta-analysis of educational feedback research and is a continuation on Hatties previous study "The Power of Feedback" Hattie and Timperley [23]. The purpose of feedback is to help the learner achieve their desired learning goal. Wisniewski found that good feedback timed well has an effect size of 1.13 on the learners learning achievement.

Wisniewski *et al.* [22] found that there are some points that are important for how effective the feedback is, how informative the feedback is, when the feedback is received and how specific it is. One general rule that seems to apply to most forms of feedback is that more information correlates with a more effective learning outcome for the user. There are now being done a lot of research on the topic of giving feedback based on information gathered from wearable devices such as eye trackers and wristbands [3], [7], [11]. With eye trackers and wristbands being able to predict cognitive load and stress levels we think that this has potential as a tool to solve the issue

of timing the feedback to the participant.

Real time Feedback Systems

Testing of real-time wearable-based feedback systems is barely explored in the literature. However, there is research that looks at the possibilities of such systems. [3], [21] Di Lascio *et al.* [21] was able to use data gathered from of-the-shelf wearable devices to identify the emotional engagement of students during lectures with high precision. They evaluated their methods on 24 students and 9 teachers over the course of 41 lectures. Using a support vector machine they got a 81 percentage recall rate with a precision of 64 percent, the recall rate was 25 percent higher then when using a biased random classifier. The data they used to accomplish this was from a commercial wrist-watch that collected the students electro-dermal activity. In their study they theorize that their models could be used to make a feedback system for students and teachers that would give them information on their engagement, enabling them to self-reflect and adjust. Because student engagement correlates in many ways to learning outcome [24] the feedback would be highly relevant for advancing ones own learning. With the proposed feedback system they would give student or the teacher the information gathered after the session, but there is nothing to suggest this could also be done in real time using the same methods.

2.3 Experts and Novices

Ericsson and Kintsch [25] proposed that when we gain expertise on a task, the information is stored in long-term memory in the form of higher-order structures or representations that are reactivated when the task is performed. The acquisition of a concept or skill is then the process of forming a template or schema for it in long-term memory, gradually integrating multiple elements into a smaller number of more complex elements that we can activate automatically [26]. Expertise is task-specific, but for any activity that involves acquiring expertise, long-term working memory provides enormous processing power. The working memory capacity of novices is in contrast severely limited.

Experts and novices work differently when approaching programming tasks. Wiedenbeck [27] suggest that experts automate some of the simpler subcomponents of their programming tasks, indicating that they make more use of their long term memory than novices. Jessup *et al.* [28] found that experts have more fixations than novices. When solving a sql database problems, the experts studied the sql database schema significantly more than the novice students [29]. Najar *et al.* [29] hypothesize that it is because the experts know to look at the schema to find the correct elements.

Because experts automate some of their actions and have a deeper knowledge, expertise can have

an effect on cognitive load. Novices do not have sufficient knowledge, they may experience higher cognitive load than an intermediate or expert[30]. Expertise effect, where experts have higher performances than novices, has an important impact on cognitive load [31].

Chapter 3

Research design

The literature review containing the related works showed how previous studies had conducted similar experiments to what we wanted to conduct. In Duchowski *et al.* [6] they used pupil dilation and the frequency of pupil oscillations to create new features of cognitive load. The new metrix called the Low/High Index of Pupillary Activity or LHIPA, is a more robust method of detecting cognitive load than the similar Index of Pupillary Activity and a lot more than using just the pupil dilation which is subjected to even small changes in lightning conditions.

Our chosen research strategy for the collection of the data is through experiments. We are going to use eyetracking and wristband technology to asses the participants in real time. The experiments are designed as a Within-Subjects (repeated measures) experiments, meaning that all participants are subjected to both the control and test environment. The debugging tasks are always provided to the participants in the same order, Snake before Tetris, and they only get feedback on Snake or Tetris, never both. We chose to design the experiments in this way to maximize the amount of data collected and we did not think it would contaminate the other results.

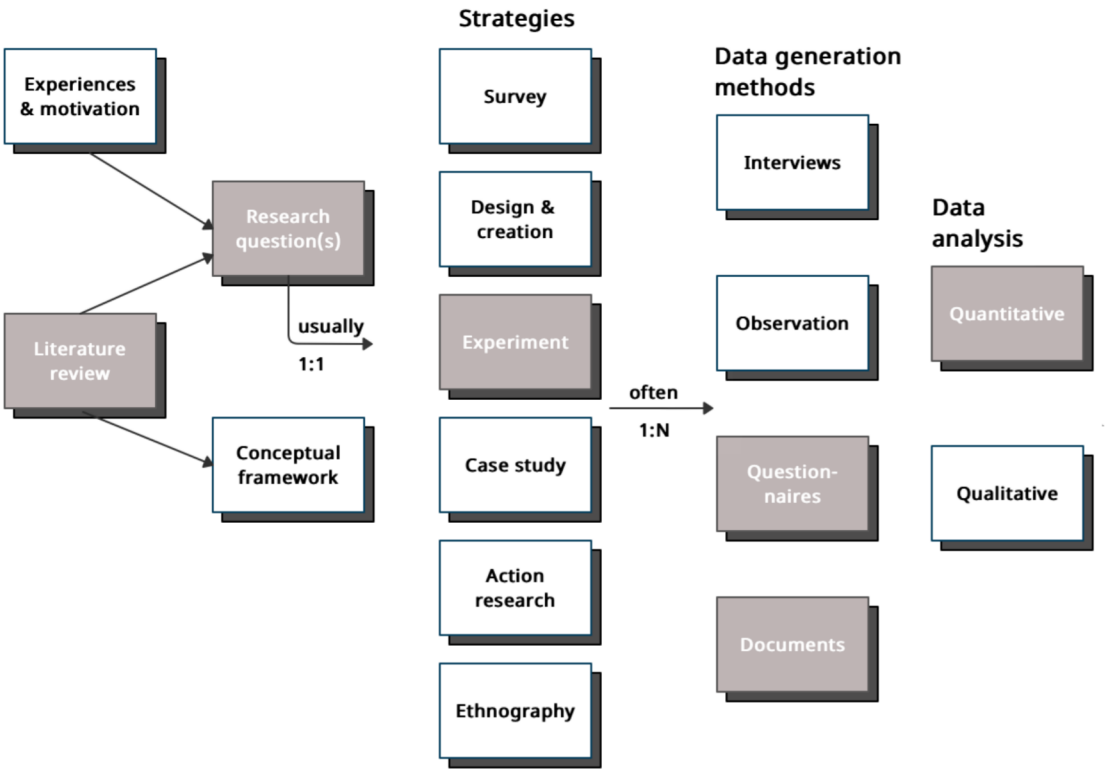


Figure 3.1: Research process design

3.1 Experiment

There are several different methods of eye tracking, but we have chosen mainly to focus on the two most dominant ways, a stationary eye tracking bar and eye tracking glasses. The bar is less obtrusive for the participant as it sits on the computer in front of the participant, but it is limited in that it can't always see the participant's eyes. For example, when the subjects looks in another direction than the eye-tracking bar. The glasses are better at continuously capturing their eyes, but it is tougher to map where the participant is looking, and the frame rate is typically lower. The glasses are also situated closer to the eyes. Glasses are more relevant if the participant needs the option to move freely, and the bar is more relevant if the participant is looking at a screen the whole time. Because we have a pretest on paper we needed the participant to be able to look away from the screen, therefore we chose to use a mobile eye tracker for our experiment.

Originally we were going to also use wristbands to monitor the participants stress levels, but had to abandon this as the hardware being used was too unreliable. The issue made it close to impossible to receive consistent heart rate data, that we needed for the real time feedback we wanted to implement. As a band-aid solution we recorded the heart rate with the Empatica mobile app with the intention to analyze this data at a later stage.

Participants

The participants for our research project was chosen based on several different key criteria. The first criteria was that the participant needed to be a students currently living in Trondheim because the tests were conducted physically on site. Criteria number two was that the student needed to have some computer coding experience, here we put the bar at the second semester intermediate course TDT4140 - Software Engineering, seeing as a second semester course would be sufficient for solving the programming tasks. Lastly the participant needed to be able to work on a monitor without using glasses as the Tobii eye trackers lose a lot of data when using glasses beneath the eye tracker glasses. With theses criteria in mind we reached out to students of informatics and computer science through different social media platforms and word of mouth. The study was conducted with in total 26 participants with ages ranging from 19 to 27.

Experimental Procedure and data collection

The participants went through 3 tasks, one pretest and two debugging coding tasks. The pretest evaluates the participants programming proficiency, while the debugging tasks were for testing of the independent variables. After each of the tasks the participants answered a NASA Task load index questionnaire(Nasa TLX), this was done to assess their perceived workload and achievement, in addition as a validation that the implementation of our cognitive load estimation was correct. The participants was put randomly in one of two groups of participants, which one they were placed in was kept secret and decided if they would receive feedback on the first or the second debugging task.

After signing the consent form and making sure they understood the contents, the participants were seated and fitted to the eye tracker glasses and wristband as seen on Figure 3.2. The participant was set directly in front of the screen they coded on, with one keyboard and a mouse. After the fitting and making sure the participant was comfortable the recording started. Additionally the monitor was also recorded using screen recording software and the participant was filmed from a web camera mounted on the monitor as can be seen on figure 3.2. They had 15 minutes on each task, including the pretest and the entire experiment took approximately 45 minutes to 1 hour. After each participant was finished, either before time or on time we then proceeded with stopping the recording. Then came the review of the answers, how many bugs were found and corrected. The reviewed data would then be decoded, graded and placed in a excel sheet for further analysis. All data from each participant was placed in a folder with their unique id making it anonymous. In the end we had the following list of documents on each participant:

DATA	TYPE
webcamera recording	.mp4
screen recording	.mp4
vscode log data	.csv
scores and answers	.csv
Cognitive load data	.csv
eye tracker data	.csv

Table 3.1: Generated participant files

Technology and Experimental setup

The experiments were conducted in a secluded room where only the participant and tester(s) were present. The eye tracking was conducted using a pair of Tobii Pro Glasses 2 running at at 50hz connected to the computer using an Ethernet cable. The physiological data was acquired using the Empatica E4 wristband and recorded using Empaticas proprietary mobile app *E4 realtime*. Web camera recordings were recorded using the windows camera app with a logitech 1080p monitor camera. The web camera was mounted on the top of the computer monitor. The screen was recorded using OBS Studio, a free screen recording software. We also tracked the participants scrolling and focus when they were programming, this was logged using the VSCode extension API.

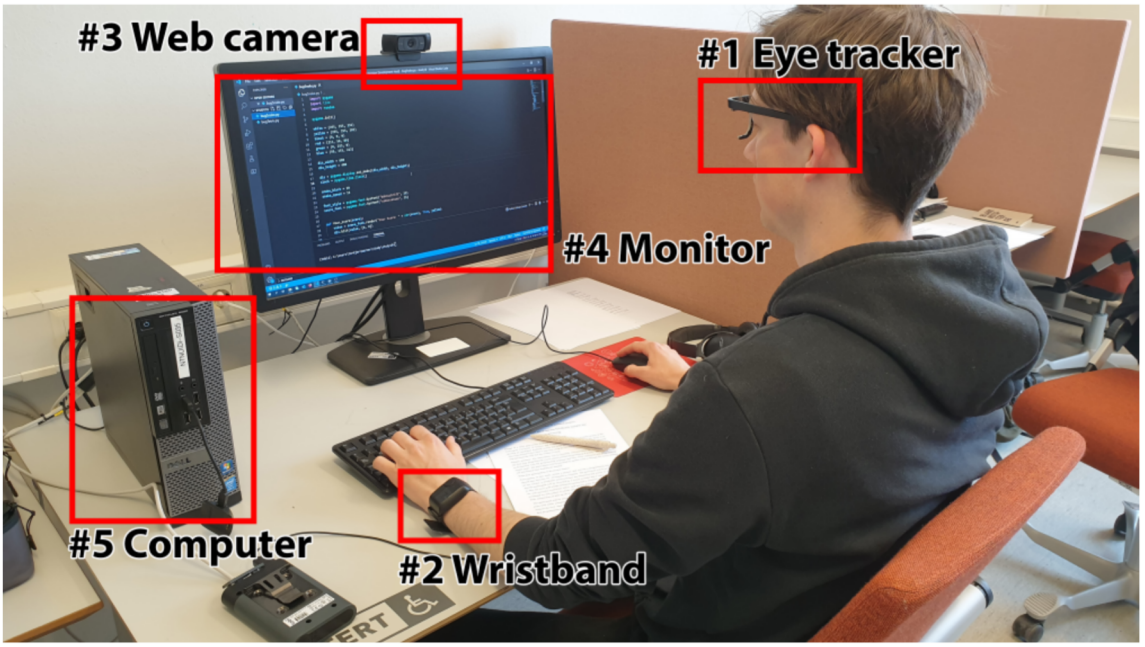


Figure 3.2: Test environment used during experiments

3.2 Analysis

A series of statistical analyses was needed in order to properly answer the research questions. To answer the research questions we introduce two independent variables. The first independent variable is feedback, which is a binary variable that we controlled during the experiment. The second independent variable is programming expertise, which we have split into Expert and Novice and calculate based on the pretest. We used these independent variables to find their effect on other dependent variables, i.e., variables that are dependent on the independent variable. In addition, we compared some of the dependent variables against each other.

Statistical Analysis

During development, before the experiments, it was shown from a pilot that the Tetris debugging task was harder than the Snake debugging task, based on the answers of five (N=5) participants. To adjust for the difference in difficulty we adjusted the grading of Tetris to be 1.5 points per 1 point for Snake. Both debugging tasks had a max score of 6 correct answers, but the adjusted Tetris max score is 9 and Snake is 6. In our analysis we wanted to compare the difficulty of the two tasks, to verify the findings from the pilot. To do this comparison we needed to make **performance** variables for Snake and Tetris, where the performance variable for Tetris is unadjusted, i.e., 1 point Snake is equal to 1 point in Tetris. Therefore, in the analysis when we talk about the differences in difficulty the 1 to 1 grading is used, but everywhere else when analyzing in conjunction with the other independent variables the adjusted Tetris score is utilized. Both performance variables for Snake and Tetris are not normally distributed and therefore all tests with them are non-parametric. To check for normality we used the Shapiro-Wilks test. To investigate the task difficulty a Wilcoxon Signed-Ranks test was applied between the score on Snake and the unadjusted score on Tetris. The experiments being repeated-measures makes score on Snake and Tetris debugging tasks dependent variables because the participants first complete the Snake debugging task, before proceeding to the Tetris debugging task.

In order to answer the questions about the cognitive load experienced during the different stages of testing we created new variables from the collected data. We took the mean of each participant's cognitive load during each of the individual segments to compute the mean **Cognitive load** for the pretest, Snake and Tetris. The cognitive load calculation is explained in Section 4.2.3.

As stated in chapter 2 cognitive load can be an indicator of how difficult a task is. The higher the load the more difficult the task is considered. To investigate the task difficulty and more precisely the tasks cognitive load, a Wilcoxon Signed-Ranks test was applied between the cognitive load measured on the Snake and Tetris debugging tasks. The cognitive load variables for Snake and Tetris are dependent variables, because each participant first goes through one task where the cognitive load is measured, and then another. The cognitive load of the second tasks might be

influenced by the participant already having gone though a possibly mentally demanding task, which is why they are considered dependent. The difference between the dependent variables is not normally distributed, which warrants the use of a non-parametric test. The sample size is also so small that it is important to check the normal distribution if one where to use a t-test.

The first independent variable is **feedback**, which is our controlled variable in the experiments. RQ1: *What is the effect of continuous feedback timed with cognitive load during programming tasks?* is the driving force of many of our hypotheses and the statistical analyses related to them.

The amount of information processed by the brain is defined as cognitive load. Programming with and without feedback should increase and decrease the quantity of information processed by the brain, respectively, resulting in an increase in cognitive load. As a result, we should also expect an increase in perceived cognitive load. To answer RQ1 we want to test the following hypothesises.

- ***H1*** - *Cognitive load increases when the user is given feedback.*
- ***H2*** - *The perceived cognitive load is higher when receiving feedback.*
- ***H3*** - *The user scores better when receiving feedback.*

To investigate the effect of feedback [**RQ2**] during the experiment within each group on debugging performance, cognitive load, and perceived mental demand, a Mann-Whitney U test was applied between the feedback and the no feedback groups. By within each group, we mean the effect when looking at feedback and no feedback for only one of the debugging tasks. Feedback is a categorical variable, while cognitive load, performance and perceived mental load variables are continuous. All the continuous variables are, when splitting on feedback, either not normally distributed or failed the test for equality of variances. Normal distribution was tested for with a Shapiro-Wilks Normality test, and variance was tested for with a Levene Test for Equality of Variances. To investigate the effect of feedback during the experiment, looking at both Snake and Tetris, on debugging performance. A Wilcoxon Signed-Ranks test was applied between the feedback group and the no feedback group. We considered using a paired t-test, but because the debugging performance variables, with and without feedback, was either not normally distributed or failed the test for equality of variances, therefore we went with a non-parametric test.

To answer our research question pertaining to expertise [**RQ2**] we split the participants into Experts and Novices, creating the variable **Expertise**. Experts are the participants that scored high on the pretest, and Novices are the ones that scored low. The split is based on the mean of all the pretest scores ($M = 4.519$), which makes the novices the ones that scored from 0 to 4 and the Experts the ones that scored from 5 to 7.

Experts are defined as the participants that score better on the pretest than Novices. We assume that they therefore need less time to understand the code, and faster understands the bigger concepts. Experts have better schemata, and might have a lower cognitive load than Novices [30].

If the assumption stands we would generally expect the Experts to score lower on cognitive load than the Novices, because they already understand some underlying concepts, that the Novices need to hold in their memory, while the Experts don't. To answer RQ2 we formulate the following hypothesis.

- ***H4****: Experts have a lower cognitive load than Novices when solving programming tasks.*
- ***H5****: Experts perceives tasks to be easier than Novices when solving programming tasks.*

To investigate the effect of the participants expertise on performance and cognitive load. A Mann-Whitney U test was applied between the Expert and Novice group on Snake and Tetris debugging tasks. Expertise is an independent categorical variable, while performance and cognitive load for Snake and Tetris are continuous dependent variables. All the continuous variables were either not normally distributed or did not have equality of variances. To investigate the effect of expertise on the total score (Snake + Tetris performance) and the average cognitive load (cognitive load Snake + cognitive load Tetris / 2) for the two tasks, a Mann-Whitney U test was applied between the Expert and Novice groups. The Expert and Novice group are independent and the total score and average cognitive load are continuous variables. When splitting the variables on knowledge level they where either not normally distributed or did not satisfy the equality of variances.

To investigate the effect of expertise on perceived mental load two different tests were applied between the Expert group and the Novice group. A Mann-Whitney U test for the pretest and a independent t-test for Snake and Tetris. When splitting on expertise the pretest perceived mental load was not normally distributed, while both Tetris and Snake were normally distributed and had equality of variances.

All the analyses were performed with python 3.10 using SciPy stats, and the diagrams were created using seaborn and matplotlib.pyplot.

Chapter 4

Implementation and Data Processing

Our goal is to implement a system that takes advantage of knowing a individuals cognitive load while they are solving programming tasks to give them relevant continuous feedback.

To implement the system and achieve this goal, we need to achieve multiple sub-goals:

1. We need to reliably collect data from participants while they are programming.
2. We need to process the data in real time.
3. We need to calculate the cognitive load in real time.
4. We need to, based on the cognitive load, give appropriate feedback.
5. We need to implement the feedback in a programming environment.

Figure 4.1 shows the outline of the system architecture. The system is divided between front end, which handle the feedback, and back-end which handles the eye tracking and cognitive load estimation. The eye tracking is explained in Section 4.1. The Cognitive Load estimator is explained in Section 4.2, along with how data is fetched from the eye tracker. The LHIPA calculation and cognitive load estimation is further elaborated on in Section 4.2.2 and 4.2.3. Finishing off the chapter with a walk trough of how the front end VSCode extension works in Section 4.3.

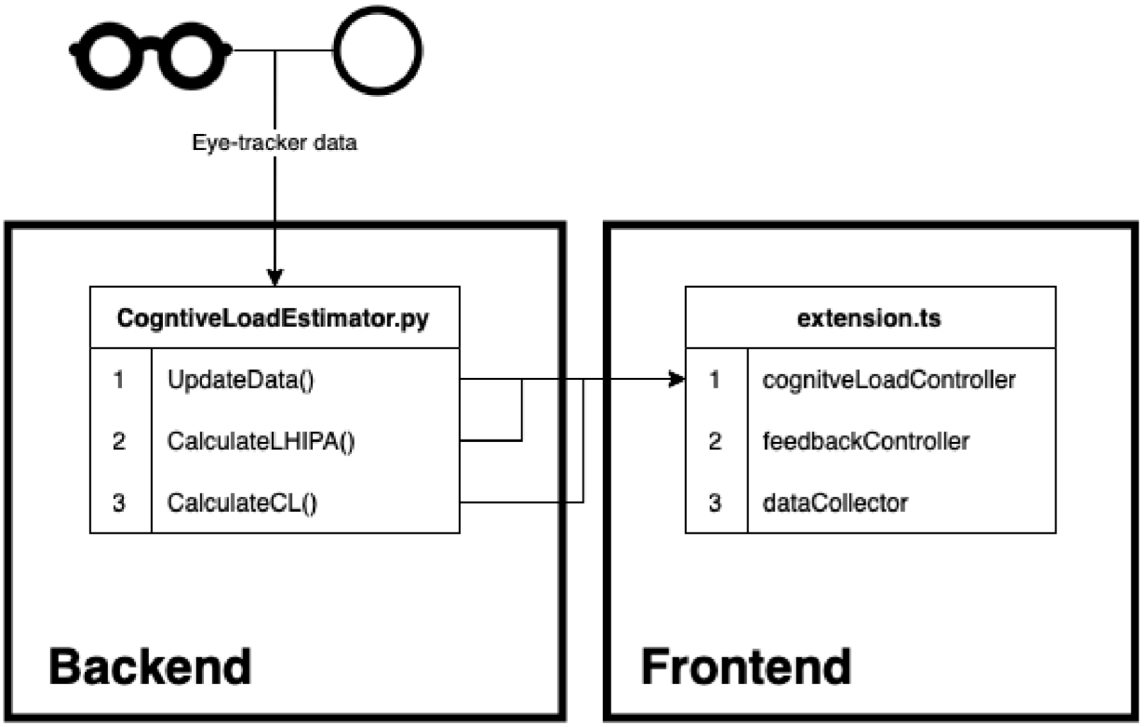


Figure 4.1: Implementation architecture

4.1 Eye tracking

This subsection explains how we achieved goal 1.

> We need to reliably collect data from participants while they are programming.

Eye tracking is carried out by employing the tobii pro glasses 2 mobile eye tracker [32] as can be seen in Figure 4.2. The eye tracker records at 50Hz. To communicate with the eye tracker we use De Tommaso and Wykowska [33]'s TobiiGlassesPyController, which is based on the tobii_research library [34] written in Python.

The TobiiGlassesPyController constantly retrieves data from the eye tracker and updates variables that may then be accessed, but when it saves data, it overwrites the previous value. Overwriting values is useful when you simply want to retrieve the most recent value, but it becomes a problem when the software has to save the data for analysis. We modified the TobiiGlassesPyController to save all new raw data without overwriting existing data. The raw data is now accessible for retrieval from the Cognitive Load Estimator. We have updated the controller to save only pupil dilation and gaze point data, rather than all non-important data. Tobii pro glasses 2 mobile can transfer data by WiFi, or through an Ethernet cable. Due to restrictions in the available gear, a Ethernet cable was used.

Figure 4.2: Tobii Glasses 2 Eye Tracker Wearable System Tobii I by Tobii Technology is licensed under Attribution 3.0 Unported.

4.2 Cognitive Load Estimation & Data Processing

This subsection explains how we achieved goal 2 & 3.

> We need to process the data in real time.
>
> We need to calculate the cognitive load in real time.

In this section we go through the elements of the core update loop, to update data, update the LHIPA based on the new data, and to calculate the cognitive load based on the new LHIPA values, as seen in Listing 1.

```
1  # 1. update data
2  self._update_data_()
3
4  # 2. Calculate and update LHIPA
5  self._update_lhipa_()
6
7  # 3. Update mean based cognitive load
8  self._calculate_load_()
```

Listing 1: Cognitive Load Estimator core update loop

4.2.1 Real time Data Processing

The Cognitive Load Estimator is responsible for calling the customized TobiiGlassesPyController, and fetch the newest eye tracking data. It calls the TobiiGlassesPyController every 500 ms, and stores the data in the object.

Data cleaning

After fetching the data, the data needs to be cleaned. The data cleaning steps can be seen in Section 4.2.1. First the right and left eye data stream have to be synchronized, as they are sent and handled separately by the eye tracking glasses. Then the users blinks have to be removed, aswell as the data affected by the users blinking.

- Synchronization
- Remove blinks
- Remove data affected by blinking

To remove blinks and the affected data the Cognitive Load Estimator searches for all data points that are blinks, as identified by the eye tracker, and removes all data 200 ms forwards and backwards from the blinking. This method of removing blinks is consistent with research in the field of cognitive load [35]. The pupil data is not further processed and is used in its raw form. Following data cleaning, the clean data is added to a clean data array, which is then used to calculate the most recent cognitive load.

Real time Cognitive Load Calculation

The real time cognitive load calculation is based on Duchowski *et al.* [6]'s Low High Index of Cognitive Load implementation. We use their implementation of the LHIPA, and alter it to work in real time. Then we use the LHIPA values to look at cognitive load over a 15 second window (mean_LHIPA), to see if the participants load stays higher, lower or about the same as a baseline cognitive load, estimated from a pretest. This calculation, results in our final cognitive load values which are used to assert what type of feedback the participant is given and when they should get it.

The specific implementation of LHIPA and mean_LHIPA as well as how to reach the final cognitive load values is explained respectively in Section 4.2.2 and Section 4.2.3. In this section we go over the aspects that makes our implementation real time.

In Duchowski *et al.* [6]'s paper they calculate LHIPA for each task. For example for a 12 seconds task, they calculate LHIPA based on the whole 12 seconds. If we used the same approach and calculated the LHIPA based on the whole session we could not give feedback before after the session

was over, which is not real time. Instead, we calculate the LHIPA based on the newest cleaned values going 6 seconds backwards in time. Therefore, when we every 500 milliseconds update the data and clean it, a new LHIPA value is calculated, and can be used to give feedback. Calculation only happens after we have accumulated 6-12 seconds of data, which through testing has shown to be enough data to calculate the LHIPA when using the db6 wavelet. Therefore, the first 6-12 seconds of any trial will be without LHIPA values. Having no LHIPA values at the start is not a problem for us, because we do not need to give feedback at the very start of a task.

4.2.2 LHIPA Calculation

Our LHIPA calculation comes from the original paper [6], and adjusted to run on python3. Our adjusted code can be found in Appendix A and the original python 2.7 code can be found in the original paper. In addition we changed the wavelet function, to db6 as recommended by Duchowski *et al.* [4], because the eye tracker has a frame rate of 50Hz. The LHIPA is calculated using the average raw pupil size of the right and left eye. It is important to point out that the LHIPA values are counter-intuitive. A low LHIPA value is equivalent to high cognitive load, while a high LHIPA value indicates low cognitive load.

Selection of the LHIPA values

When selecting values to use for the real time calculation we select the last n_seconds * frames_per_second clean values. This means that if the participants has blinked a lot in the last seconds the data will be an aggregate of data further back than n_seconds of raw data. By always taking the last seconds of clean data we get a consistent number of values to calculate the LHIPA from. If we only selected values from the last 10 seconds, and there were enough blinks/ void data, then calculating of the LHIPA could result in getting 0 values instead of the actual LHIPA value. This is because the LHIPA needs a certain amount of values to be able to give a number. With db6 it needs 176 values, and because we use a headset with a frame-rate of 50Hz that means more than 3 seconds of clean data. Therefore we select values as far back as needed to consistently calculate the LHIPA.

Minimum number of values

The reason we need 176 values is because of the way the LHIPA works. The max decomposition level calculation can be seen in Equation 4.1. When not enough values are used to calculate the LHIPA, the max decomposition level comes out to less than 4. When the max decomposition level is below 4 the lower frequency band is equal to the higher frequency band.

$$max\_level = |\log_2(\frac{data\_len}{filter\_len - 1})| \tag{4.1}$$

The two bands are then used to obtain the HI/LO frequency. The HI/LO frequency is calculated by dividing the low frequency at position i by the high frequency at a position dependent on the

max decomposition level, which results in, when the low and high frequency bands are equal, dividing the low frequency at position i by the high frequency at position i. And, because they are equal to each other, the result comes out as an array filled with 1s. When the calculation later uses these ones to check for oscillations, it finds none and returns 0.

To get a max decomposition level above 4, using Equation 4.1, when using db6, which has a length of 12, we need a data_len of at least 176.

4.2.3 Mean LHIPA and cognitive load calculation

The mean LHIPA is the mean of he last 30 LHIPA values, the same as 15 seconds of data collection when using a .5 second sampling window. The mean LHIPA is used to figure out if the participant is in high, medium or low cognitive load. As a baseline value to compare the mean LHIPA with during debugging of task 1 and 2, we use the mean of all the LHIPA values from the entire pretest, see Equation 4.2. The selection of 30 LHIPA values is based on a test on 4 participants during development of the system. By looking at a 15 second window the system is able to see the greater trends in cognitive load, and ignore the small variances in the data.

$$baseline = \frac{\sum_{i=1}^{n} pretest\_LHIPA_i}{n} \tag{4.2}$$

Using mean LHIPA to find cognitive Load

When calculating the users cognitive load we check if the current mean LHIPA is 0.1 higher or lower than the baseline mean LHIPA from Equation 4.2. Which translates to: When the participant has a mean_lhipa of 0.1 away from the baseline over the previous 30 LHIPA values assign either high or low cognitive load. When the mean LHIPA is higher than 0.1 above the baseline the participant has low cognitive load. When the mean LHIPA is lower than 0.1 below the baseline the participant has high cognitive load. And when the mean LHIPA is closer than 0.1 from the baseline the participant has medium cognitive load. Cognitive load calculation as code can be view in Listing 2.

```
1  # take mean of last 30 lhipa values
2  mean_lhipa = np.mean(lhipa_values[-30:])
3
4  # if the mean is 0.1 higher than the baseline the load is low
5  if mean_lhipa > (baseline + 0.1):
6      load = LoadType.LOW
7  # if the mean is 0.1 lower than the baseline the load is higher
8  elif mean_lhipa < (baseline - 0.1):
9      load = LoadType.HIGH
10 else:
11     load = LoadType.MEDIUM
```

Listing 2: mean cognitive load to high medium or low

The 0.1 change from the baseline to calculate cognitive load is based on tests on four (N=4) subjects, that tested the system after development and before the final study. The 0.1 deviation ensures that when the subject has a high load over the set time period, they will be getting feedback. The change from the baseline could have been set higher or lower based on if we wanted the feedback to be given by smaller or bigger deviations from the baseline. A change of 0.2 from the baseline would drastically reduce the number of times the person is considered to be in high and low load, and drastically increased the number of times they were in the middle.

4.3 VSCode Extension

This subsection explains how we achieved goal 4 & 5.

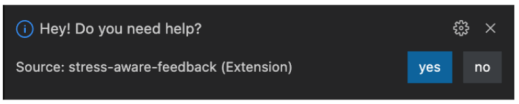
> We need to, based on the cognitive load, give appropriate feedback.
>
> We need to implement the feedback in a programming environment.

The participants had to use VSCode to perform the programming tasks and the system was specifically tailored to this code editor. VSCode was picked because of its easy to use extension API that enables editing of its user interface and notifications. The extension is written in Typescript and uses Node.js and the VSCode extension API. The system has three main functions, receive and send data from the Cognitive Load Estimator, display appropriate feedback based on data from the Cognitive Load Estimator and to log relevant data for analysis.

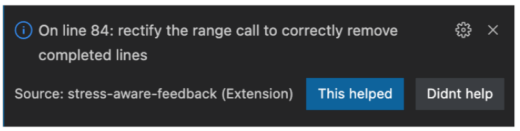
The first function of the system to receive the data from the Cognitive Load Estimator was done using one asynchronous fetchdata() method. The data received from the backend was the current cognitive load, the average cognitive load, and the percentage of load. All the received data is shown, when feedback is given, to the participant in the bottom bar of the VSCode window seen in Figure 4.3d. Based on the information received from the Cognitive Load Estimator the program displays the appropriate feedback.

When the cognitive load is high for a significant amount of time, the program gives the feedback from Figure 4.3c and if the cognitive load is low it displays the feedback from Figure 4.3a. When the participant is shown the low cognitive load example (Figure 4.3a) and press yes, indicating that they want help, they are shown one of the prewritten hints as shown in Figure 4.3b. The hints are selected based on where in the code the user currently is located as provided by the VSCode API. I. e., when the user is 15 lines from a bug on line 75, but only 3 lines from a bug on line 57 the feedback provides the hint associated with line 57. When the user receives a hint it is provided with two options: "This helped", and "Didn't help" as can be view in the bottom right of Figure 4.3b. When the user press "This helped" the hint/feedback is removed from the pool of possible hints the user can receive, in such a manner that if the user is in low cognitive load at a

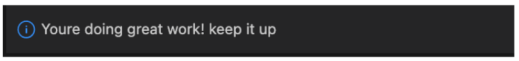
later stage and close to the same buggy lines, it gets a hint it has not been given before. When the user presses "Didn't help" the feedback from Figure 4.3a is displayed again and the user can select "Yes" or "No" again.



(a) Low cognitive load feedback



(b) Low cognitive load hint - example



(c) High cognitive load feedback

(d) Continuous feedback example

Figure 4.3: Types of feedback shown to the participant during debugging tasks

Overloading or spamming the user with popups during the programming was not wanted, therefore a cooldown period for popups was implemented. Whenever the user has not interacted with a popup such that the popup vanishes, or presses this helped the cooldown period begins. The cooldown period is such that the user will have some time to find the hint and think about how to solve the issue. The cooldown period is set to 2 minutes, and when the timer is done the extension can send popups again.

Chapter 5

Results

After the study there were in total n=26 students who participated in the experiments. All participants went through the same procedure, 3 tests with 15 minutes each to complete, starting with the pretest and finishing with debugging a Snake game and a Tetris game. Of the collected data no participant was excluded, but one participant had errors with the pretest collection. For that one participant we sat the cognitive load variables to the mean of all the other participants. The Empatica E4 wristband data is not included in the results and is excluded from the rest of the study. Due to unforeseen circumstances, time limits and issues with the data collection procedure, the data had to be left out.

The results are divided into three main categories, first we look at the performance, before moving on to cognitive load and perceived mental load. We end the results by looking at interactions and the amount of feedback. Within the three first parts the independent variables expertise and feedback, are both analysed.

5.1 Performance

Figure 5.1 and Figure 5.2 shows 3 histograms of the results for the participants, pretest score ($M = 4.5$), Snake score ($M = 1.9$) and Tetris score ($M = 1.6$). On the pretest the highest score was 7 out of a maximum of 10 and the lowest was 1. On Snake, left in Figure 5.2, 10 participants found zero bugs and scored a 0 and 2 found all 6. In addition the ones that found all bugs finished before their time was up. On Tetris, right in Figure 5.2, 6 participants found zero bugs and scored a 0, one scored 6 out of 9, and one scored 4.5 out of 9 (after the score was multiplied by 1.5).

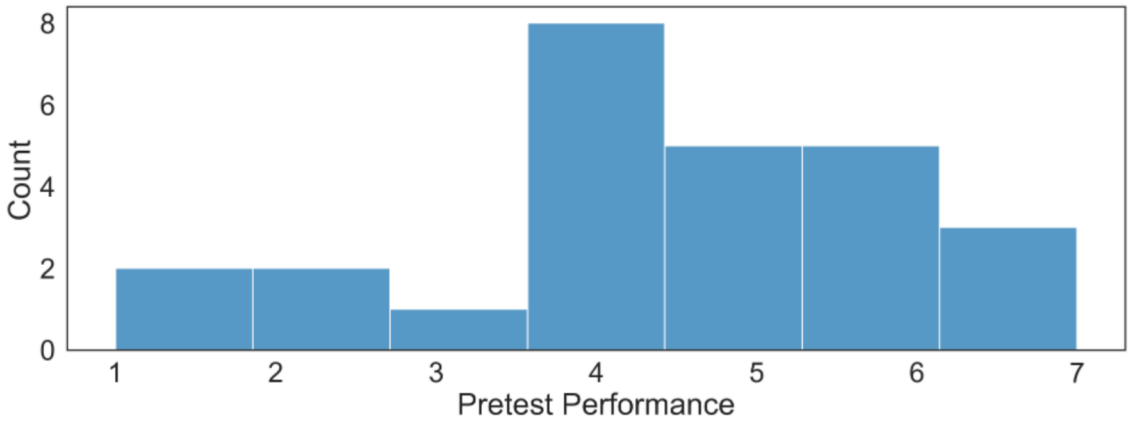


Figure 5.1: Performance distribution on the pretest.

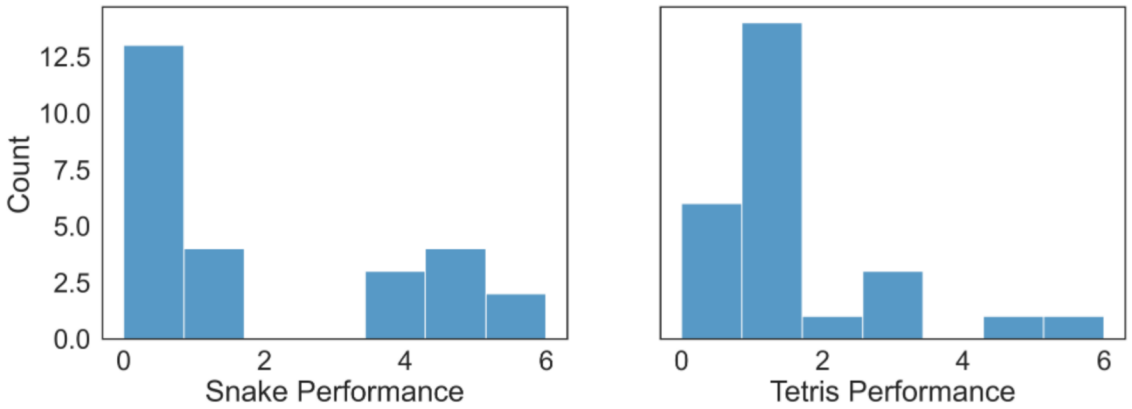


Figure 5.2: Performance distribution for Snake and Tetris debugging tasks. The Tetris performance is the adjusted Tetris performance.

Task difficulty

During the development a pilot was conducted to check the difficulty of Snake and Tetris debugging tasks with five participants. In the pilot, Tetris was discovered to be harder than Snake. Therefore, when comparing feedback and expertise against performance the adjusted Tetris score is used, for example in Figure 5.2. To validate the new scoring we investigate the task difficulty. A Wilcoxon

Signed-Ranks test indicates that score on Snake ($M = 1.942$) is greater than the score on Tetris ($M = 1.096$); Score on Snake is significantly greater than score on Tetris. ($t = 162.5$, $p = .05$). The test confirms the findings from the pilot, and the use of the adjusted Tetris score is valid.

Expert vs Novice

A Mann-Whitney U test indicates that the "expert" group ($M = 3.154$) had a greater score than the "novice" group ($M = 0.731$) when working on solving Snake debugging task; there is a significant difference in the Snake score ($U = 23.0$, $p = .0006$). A Mann-Whitney U test indicates that the "novice" group ($M = 1.385$) had the same score as the "expert" group ($M = 1.904$) when working on solving Tetris debugging task; there is no significant difference in the Tetris score ($U = 65.5$, $p = .149$). This again confirms that the Tetris task was more difficult than the Snake debugging task.

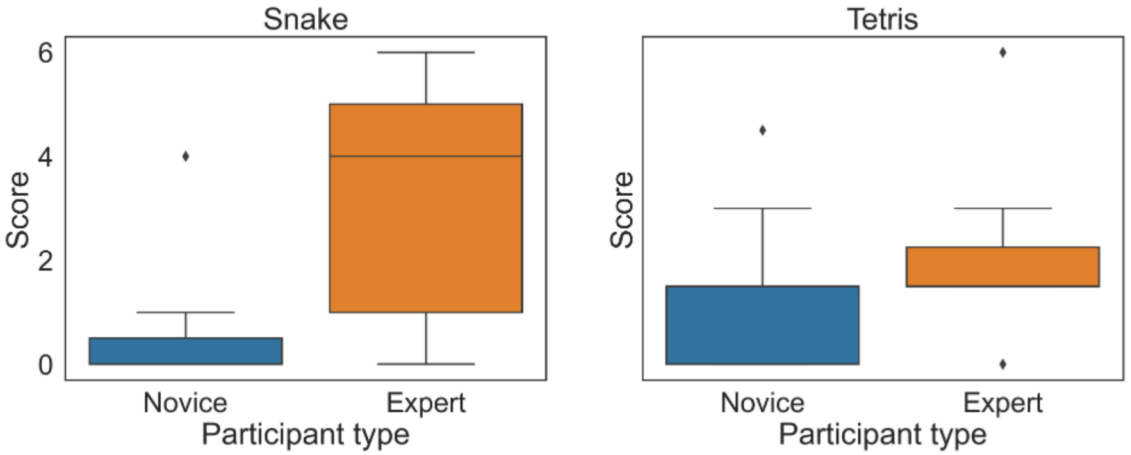


Figure 5.3: Performance on Snake and Tetris tasks by expertise.

The Expert group does better than the Novice group on Snake, but not significantly on Tetris. We also looked at the experts and novices scores collectively for the sum of Snake and Tetris. A Mann-Whitney U test indicates that the "expert" group ($M = 5.057$) had a greater score than the "novice" group ($M = 2.115$) for the sum of Snake and Tetris tasks; there is a significant difference in the total score ($U = 138, p = .003$). Showing that the Expert group did better than the Novice group when looking at the sum of both tasks.

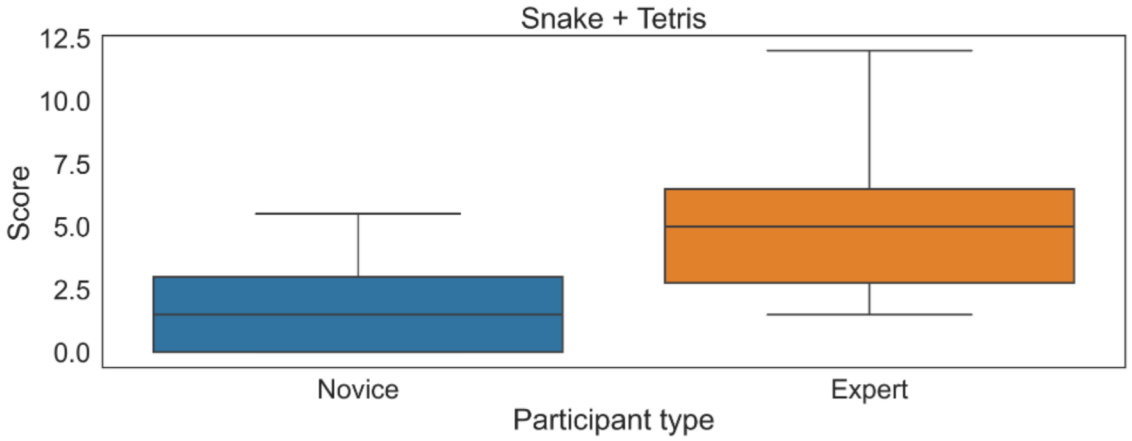


Figure 5.4: Sum of Snake and Tetris performance grouped by expertise.

Feedback vs No feedback

In Figure 5.5, on the right, it looks like the feedback might have an effect on the score on the Tetris task. A Mann-Whitney U test indicated that the "feedback" group ($M = 2.25$) scored greater than the "no feedback" group (mean $M = 1.04$); there was a significant increase in test score ($U = 124.5$, $p = .013$), confirming the suspicion. Of the participants that received feedback on Tetris, 8 are categorized as Experts and 5 as Novices. Vice versa for the no feedback group.

For the Snake task a Mann-Whitney U test indicated that the "feedback" group ($M = 1.73$) scored the same as the "no feedback" group ($M = 2.15$); there is no significant difference in test score ($U = 68$, $p = .198$). Of the participants that received feedback on Snake, there were 5 Experts and 8 Novices. Vice versa for the no feedback group.

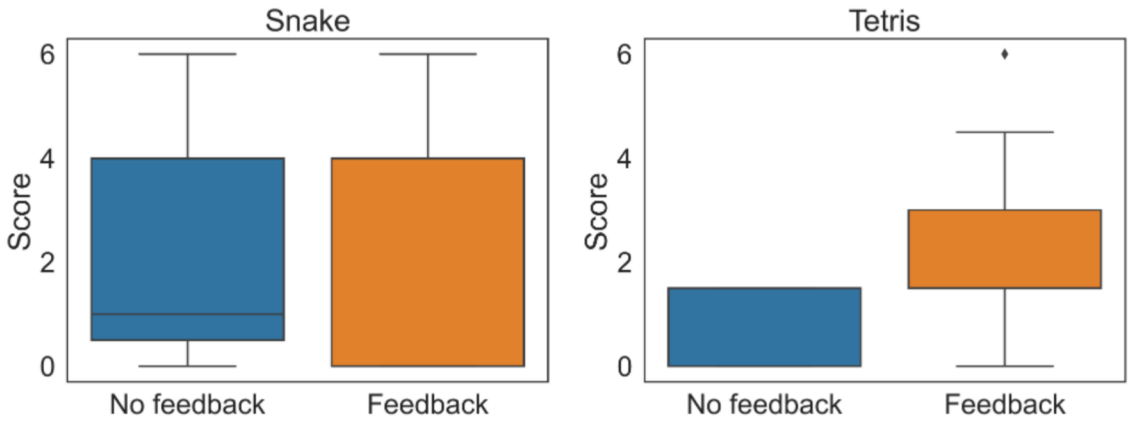


Figure 5.5: Performance on Snake and Tetris by whether or not the participant was given feedback on the task.

To see if there was a difference between the group that received feedback and not we ran a Wilcoxon Signed-Ranks test. The Wilcoxon Signed-Ranks test indicated that the "feedback" group ($M =$

1.615) scored the same as the "no feedback" group ($M = 1.423$); There was no significant difference in test score ($t = 125.5$, $p = .364$).

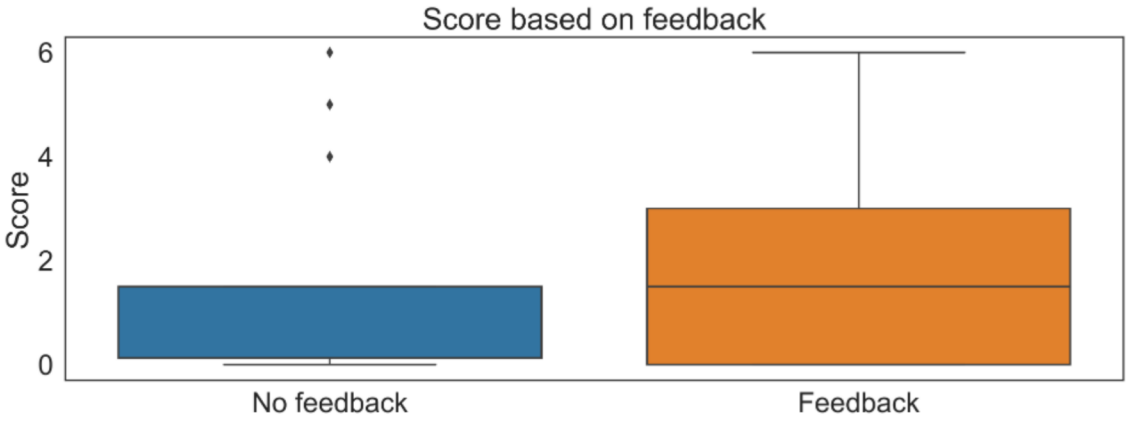


Figure 5.6: Performance when the participant received feedback independent of which task the feedback was received.

.

5.2 Cognitive load

Figure 5.7 shows 3 box plots with the calculated mean cognitive load of the participants. Figure 5.7 shows a steady increase in the cognitive load across all participants. The cognitive load of the participants during both of the debugging tasks were close, Snake ($M = 2.04$) and Tetris ($M = 2.07$). On the pretest ($M = 1.84$) there was no baseline to compare to, so a constant from earlier tests was used for all participants. The cognitive load should therefore not be used to compare the cognitive load from the pretest with Snake and Tetris debugging tasks.

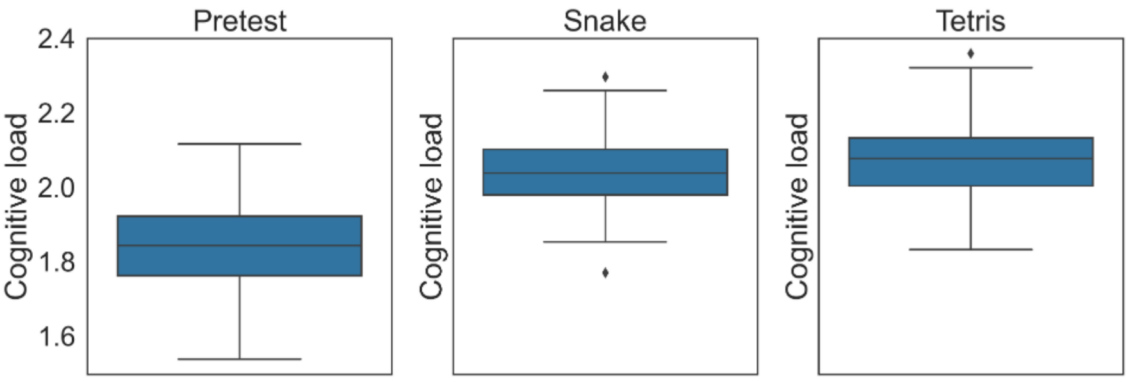


Figure 5.7: Cognitive load for pretest, Snake task and Tetris task.

Cognitive load difference between Tetris and Snake

A Wilcoxon Signed-Ranks test indicates that cognitive load on Snake ($M = 2.03$) is lower than the cognitive load on Tetris ($M = 2.07$); cognitive load on Snake is significantly smaller than cognitive load on Tetris ($t = 79$, $p = .007$).

	Snake	**Tetris**
CL	2.036 (0.126)	2.073 (0.124)

Table 5.1: Cognitive load mean scores.

Expert vs Novice

A Mann-Whitney U test indicates that the "expert" group ($M = 2.000$) had lower cognitive load than the "novice" group ($M = 2.072$) when working on solving the Snake debugging task; there is a significant difference in the mean cognitive load ($U = 51.0$, $p = .045$). A Mann-Whitney U test indicates that the "novice" group ($M = 2.106$) had the same cognitive load as the "expert" group ($M = 2.043$) when working on solving the Tetris debugging task; there is no significant difference, but there is an indication that the "expert" group has a lower mean cognitive load ($U = 59$, $p = .100$).

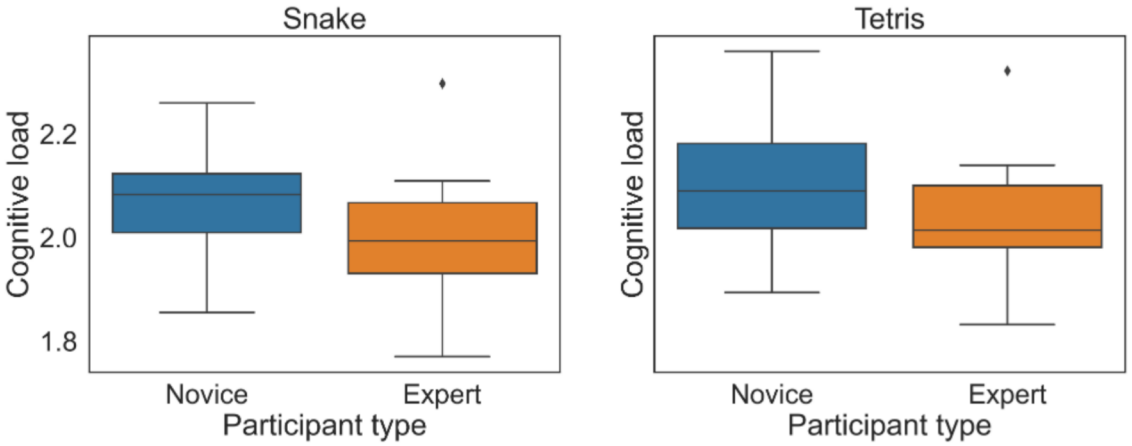


Figure 5.8: Cognitive load by expertise.

We also looked at the "expert" and "novice" performers mean cognitive load collectively for the sum of Snake and Tetris. A Mann-Whitney U test indicates that the "expert" group ($M = 4.043$) had the same cognitive load as the "novice" group ($M = 4.178$) for the sum of Snake and Tetris tasks; there is not a significant difference in the sum of cognitive load ($U = 53$, $p = .056$). Though it is not statistically significant, there is a strong trend towards the experts having a lower cognitive load.

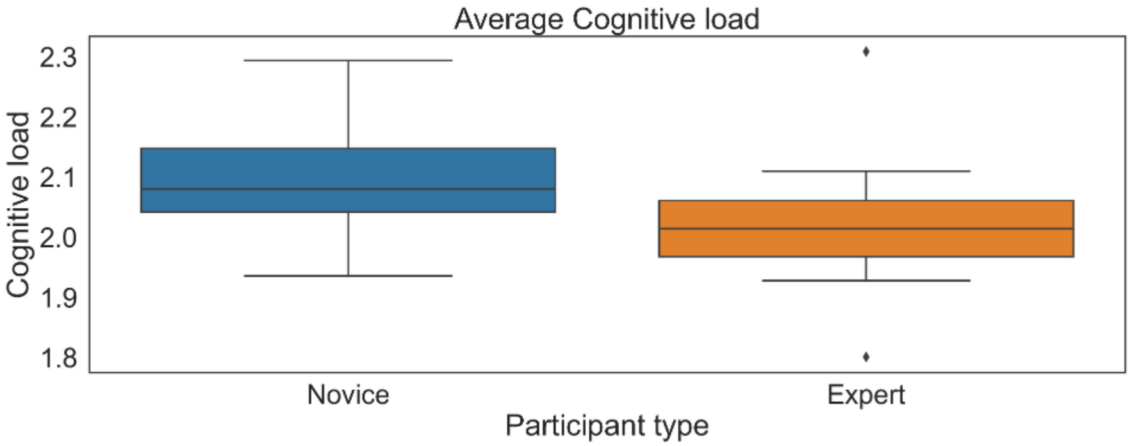


Figure 5.9: Average cognitive load by expertise for both Snake and Tetris.

Feedback vs No Feedback

A Mann-Whitney U test indicates that the "feedback" group ($M = 2.02$) had the same cognitive load as the "no feedback" group ($M = 2.05$) when working on solving the Snake debugging task; there is no significant difference in the mean cognitive load ($U = 80$, $p = .419$). A Mann-Whitney U test indicates that the "feedback" group ($M = 2.08$) had the same cognitive load as the "no feedback" group ($M = 2.06$) when working on solving the Tetris debugging task; there is no significant difference in the mean cognitive load ($U = 84$, $p = .5$).

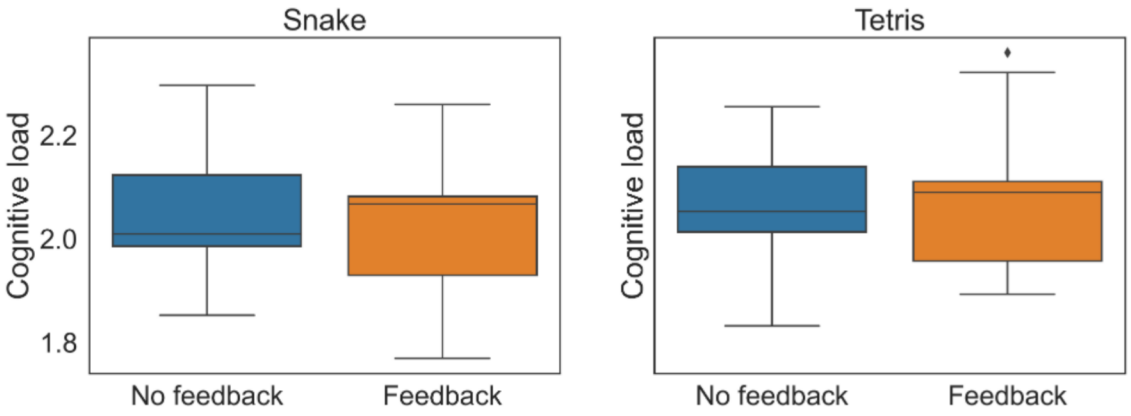


Figure 5.10: The participants mean cognitive load by feedback or not.

We also looked at the "feedback" and "no feedback" groups cognitive load independent of which task they received feedback on. A Wilcoxon Signed-Ranks test indicates that the "feedback" group ($M = 2.051$) had the same cognitive load as the "no feedback" group ($M = 2.058$); there is no significant difference in cognitive load($t = 165$, $p = .79$).

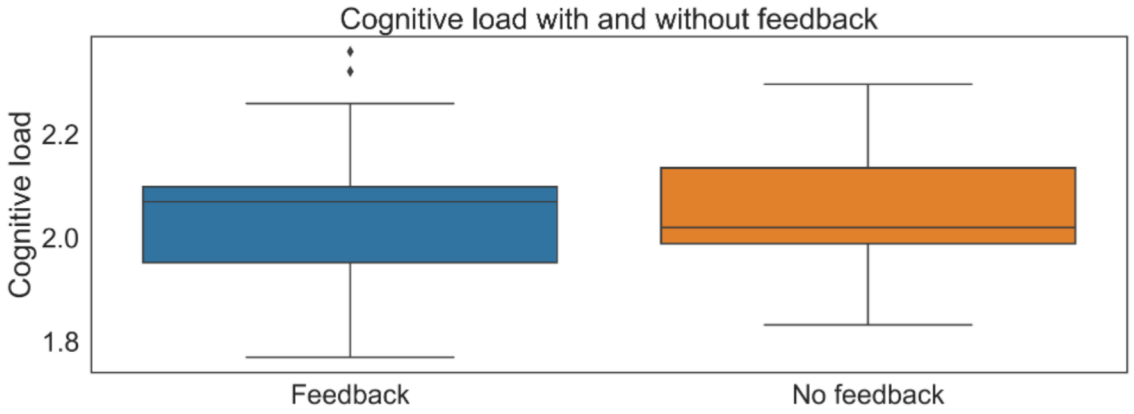


Figure 5.11: Cognitive load grouped by receiving of feedback. Independent of which debugging task the participant had feedback on.

5.3 Perceived mental load (Nasa TLX)

The NASA Task Load Index(TLX) was a survey conducted after every test and the results can be found in Table 5.2 with mean values for each question separated by task. The survey answers showed that the participants perceived that the pretest and Snake debugging used a similar amount of mental demand with scores of 13.85/20 and 13.88/20 respectively, but giving the Tetris debugging task a higher score of 15.04/20. All mental demand scores are closer to 20 (very high) than 0 (very low). On a scale from perfect(1) to failure(20) on the question of performance there is a steady increase of about 1 point for each test starting with 13.73 on the pretest and 15.08 and 16.04 on the Snake and Tetris debugging tasks respectively. The participants answered that they put somewhat above average effort with scores ranging from 13.38 to 13.96 of 20 on the tests where 20 is high effort and 1 is low effort.

	Pretest	**Snake**	**Tetris**
Mental Demand	13.85(3.54)	13.88(3.27)	15.04(3.38)
Physical Demand	3.31(2.92)	3.46(2.8)	4.0(3.45)
Temporal Demand	11.5(3.83)	12.0(4.24)	11.77(4.18)
Performance	13.73(4.56)	15.08(5.66)	16.04(3.93)
Effort	13.38(3.94)	13.38(4.18)	13.96(4.77)
Frustration	12.85(4.5)	11.77(5.42)	13.31(4.58)

Table 5.2: NASA-TLX mean scores.

Mental demand difference

A Wilcoxon Signed-Ranks test indicates that perceived mental load on the Snake debugging task ($M = 13.88$) is lower than the perceived mental load on the Tetris debugging task ($M = 15.04$); Perceived mental load on Snake is significantly lower than perceived mental load on Tetris ($t = 35.5$, $p = 0.05$).

Expert vs Novice

A Mann-Whitney U test indicates that the "expert" group ($M = 13.462$) had the same perceived mental load as the "novice" group ($M = 14.231$) when working on the pretest; there is no significant difference in the perceived mental load($U = 70.5$, $p = .243$).

A independent t-test indicates that the "expert" group ($M = 13.385$) had the same perceived mental load as the "novice" group ($M = 14.385$) when working on the Snake debugging task; there is no significant difference in the perceived mental load($t = -0.774$, $p = .446$).

A independent t-test indicates that the "expert" group ($M = 13.385$) had the same perceived mental load as the "novice" group ($M = 14.385$) when working on the Tetris debugging task; there is no significant difference in the perceived mental load($t = 1.618$, $p = 0.119$).

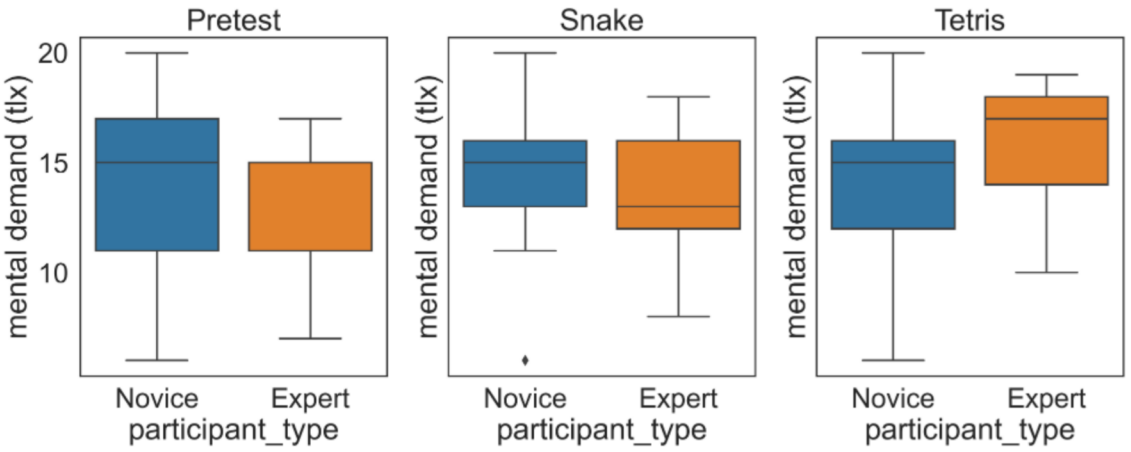


Figure 5.12: Mental demand as reported by a NASA TLX form compared by if the student is classified as an Expert or Novice for respectively the pretest, Snake and Tetris.

Feedback vs No Feedback

A Mann-Whitney U test indicates that the "feedback" group ($M = 13.31$) had the same perceived mental load as the "no feedback" group ($M = 14.46$) when working on solving the Snake debugging task; there is no significant difference in the perceived mental load($U = 71.5$, $p = .260$). A Mann-Whitney U test indicates that the "feedback" group ($M = 16.23$) had a greater perceived mental

load in comparison to the "no feedback" group ($M = 13.85$) when working on solving the Tetris debugging task; there was a significant increase in the perceived mental load ($U = 117$, $p = .0495$).

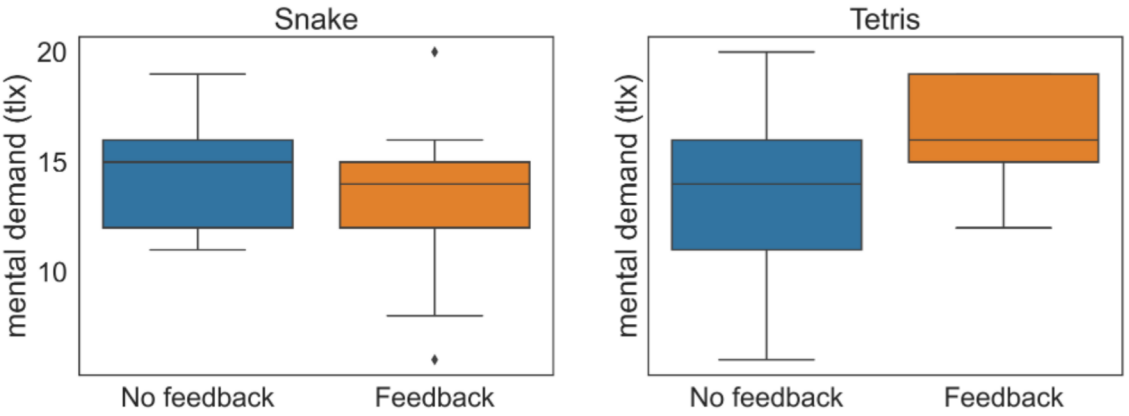


Figure 5.13: The self reported mental demand compared on Snake and Tetris tasks by whether or not they received feedback on the task.

5.4 Interactions and number of feedbacks

In Table 5.3 we can see that more participants interacted with Tetris feedback popups than with Snake feedback popups. With respectively 10 out of 13 and 8 out of 13 of the participants interacting with the popups one or more times, meaning that clicked on one of the buttons of the hint popups seen in Figure 4.3b. Only 8 out of 13 participants interacted with the low cognitive load(CL) popups on Snake, meaning that almost half did not use the popups. Additionally, there were more high cognitive load feedback popups than low cognitive load popups with a total of 74 low cognitive load(CL) popups and 95 high cognitive load(CL) popups, when looking at the total for both tasks. Only two times, did any participant press no on the feedback. Of the 74 low cognitive load(CL) popups no interaction happened many times, as can be seen by the 54 yes or no presses, leaving 22 non interactions.

	no interactions	**interacted**	**high CL popups**	**low CL popups**	**hints received**	**did not need help**
Snake	5	8	44	41	22	2
Tetris	3	10	51	33	30	0
Total	8	18	95	74	52	2

Table 5.3: Interactions with feedback.

In one of the debugging tasks the participant was shown feedback based on their current calculated cognitive load. The types of feedback shown can be seen in Figure 4.3. During the Snake debugging

if the participant was receiving feedback they received a pretty similar amount of both high and low feedback types. If the participant was receiving feedback on the Snake debugging task they received more high load popups than the ones getting feedback on Snake. In Table 5.4 there is shown that for the ones receiving feedback on Tetris they received a mean of 3.92 cheer popups while only receiving 2.54 help popups on average.

	Low CL help popups	**high CL cheer popups**
Feedback on Snake	3.15(1.14)	3.38(1.33)
Feedback on Tetris	2.54(1.27)	3.92(1.19)

Table 5.4: Amount of feedback given

Chapter 6

Discussion & Limitations

In this chapter we will discuss the results from our experiments, with a focus on answering our hypothesises surrounding the research questions.

6.1 Findings

- Performance is positively effected by feedback on the Tetris debugging task
- Performance has no effect on the Snake debugging task.
- Experts perform significantly better than Novices.
- Cognitive load is not effected by feedback.
- Cognitive load is significantly higher on Tetris than on Snake.
- Participants scored significantly lower on the Tetris debugging task than on the Snake debugging task.
- For all tasks, Experts exhibited a strong trend toward having a lower cognitive load than Novices, with the Snake debugging task being significantly lower.
- Perceived mental demand is significantly higher for the feedback group on Tetris than the no feedback group, but equal on Snake.
- Perceived mental demand is equal for the feedback group and the no feedback group on Snake.
- 31% ignored or did not see the feedback popups.

6.2 Validity of Cognitive load Measurements

We wanted to make sure that the cognitive load results we are basing our subsequent discussion on were valid and in line with the previous research done on the topic. By looking at Table 5.1 and the associated analysis, we can see that there is a significant increase in cognitive load from Snake to Tetris. This agrees with previous research that indicates that more difficult tasks are more cognitive demanding [6]. The same increase can be seen on the significant increase in perceived mental demand on the same tasks as shown in Section 5.3. That the measurements agrees with each other making us comfortable concluding with that the assumption that the cognitive measures were valid.

6.3 Feedback - [RQ1]

What is the effect of continuous feedback timed with cognitive load during programming tasks?

We hypothesized that when participants were given feedback, their overall performance would improve as a result of the help they were given. There was no significant difference in performance between scores when provided feedback and scores when not given feedback, as indicated in Figure 5.6. Indicating that receiving feedback did not improve overall performance. When we look at each task separately, we can see that receiving feedback aided the participants in the Tetris debugging task significantly, but not the Snake debugging task.

There could be a number of reasons why participants showed a significant improvement on Tetris whereas they did not on Snake. To begin, if the groups were divided unevenly by knowledge level, a difference would have been expected, because experts performed better than novices overall. The Expert and Novice groups were split nearly evenly, with 5 and 8 in each group, making this less likely. Another reason could be the difference in difficulty between the two debugging tasks, with Tetris being more difficult overall than Snake. This could imply that the provided feedback is more effective on more difficult programming tasks than on easier programming tasks. A third reason is that Snake feedback may be less useful than Tetris feedback. Either the Snake debugging was so simple that the feedback hints were unnecessary, or they were insufficiently useful. Furthermore, because feedback hints were offered depending on code position, it was possible that the user had previously solved the problem that the feedback hinted towards. Because the solve rate on Snake was higher, the likelihood of this increasing and being a reason for the performance difference increases. The potential of a real-time feedback system is demonstrated by some positive, but no negative effects on performance.

A main finding is that the cognitive load measurement did not increase nor decrease between the

feedback and no feedback groups, indicating that the introduction of feedback did not have an effect on the cognitive load of the participant. This disproves our first hypothesis that cognitive load would increase as additional elements appear on screen. There could be several explanations for this though, one reason is that some of the contestants did not interact with the feedback at all. Of the 26 people that participated in the study there was a total of 8 that did not interact with the feedback at all as seen in Table 5.3, meaning that they either did not see them or they ignored them entirely. Another reason could be that the feedback was integrated well with the editor using the default notifications of VSCode resulting in a seamless experience for the users. If we had gone with another solution with more intrusive feedback, adding sound and making the popups larger, we maybe would've seen a larger difference. The potential of a real-time feedback system is demonstrated by no negative effects on cognitive load.

The perceived mental load when receiving feedback was higher than when not receiving feedback on Tetris, but not on Snake. The increased reported perceived mental load can be explained by several factors. Reporting of mental load after completion of a task is less accurate than during the task [11]. The measurements (read cognitive load) made during the tasks could therefore be more accurate and a more representative measure of the participants actual cognitive load. Another explanation is that the term mental load was not explained to the user, but left to the participant to decipher resulting in possible misinterpretations. During the experiment the term cognitive load was not explained, and the term mental demand was also not explained. Thus it was up to each participant to decide for themselves. The choice to not explain mental load and cognitive load was to interfere as little as possible during the test, but in hindsight it could have been important to explain in relation to the self reported scores. Based on these finding we do not think that the NASA TLX questionnaire was the best choice, possibly because of how we carried out the experiment.

6.3.1 Continuous feedback

When speaking with participants after their study was conducted, no one noticed the continuous feedback on the bottom of the screen from Figure 4.3d. This could be because the feedback is so small, and the fact that the numbers change is not very noticeable or simply because the participant did not know what they were looking at. In addition, since the participants use a lot of their attention on trying to figure out what is going on in the code, they might not have a lot of time to look around the screen for additional information. In future work making use of more clear information or limiting the amount of feedback to test the usability and effect of the system is advised.

6.4 Expertise & Cognitive load - [RQ2]

What is the effect of expertise on cognitive load during programming tasks?

From previous research we expected the cognitive load of Experts to be lower than that of Novices [25], [26], [30]. Our findings show that the Expert group had a lower cognitive load than the novices, but only on the Snake debugging task was the difference statistically significant. The only sign of lower cognitive load in the Tetris debugging task was in the Expert group, where we found a not significant, but a slight indication that they could have lower load. The difference could be due to the Tetris task being more challenging, resulting in participants having a higher cognitive load on Tetris. This suggests that both groups were approaching their maximum cognitive load and even overload, which is when pupillary responses begin to fall after having reached their highest point. [14].

There was no indication that Experts perceived the tasks to be less mentally demanding than Novices. There could be several reasons for this, many of which are already discussed in the previous section of why the perceived difficulty was not significantly different in the two games. A reason not discussed is that the relative mental demand is subjective to the participants own expectations, something that could've been avoided if the scale had been more properly explained. Another reason could be because of the within subject design of our experiments, the participant could not adjust their answers for the first test after completing the second. This becomes a issue if the participant rates the first task too high not leaving any room for a higher demanding task. A fix for this would be to let the participants change the scores after each test to account for the new insight.

6.5 Limitations & future work

This study has 5 main limitations that we will present and suggest how we could fix them in the future. Firstly, there are some limitations with the feedback, how the feedback was timed and how it was presented. There is no test to see if giving feedback based on a random timer could have produced the same results as the feedback given in our study. We chose the minimum amount of minutes between popups arbitrarily and there is no telling if another time interval might have been better suited. We focused on the effects of the feedback that was timed, and took the first step towards further studies using real time cognitive load estimation, and cognitive load estimation as a tool for real time feedback during programming. To figure out if the same effect could have just because hints were given, or if there is an additional effect from the timing further work is needed.

The second limitation is the number of people who participated in the system's research and testing. We had 26 students participate, which is sufficient for some analysis but restricts the weight we

can place on our findings. We could have done a lot more categorizations and made a lot more with more volunteers. With more participants, the results would have been clearer, and we might have been able to analyze the data with greater precision, resulting in larger trends in the data. The participants were also chosen because they had little to no prior programming expertise; as a result, it's uncertain whether the findings can be applied to other sorts of feedback. Increased testing time and effort put in to recruit students would have been a solution, however we were unable to do so due to the time constraints in this thesis.

Thirdly, we divided the subjects based on their pretest mean score, which has certain drawbacks. When we split on the mean, we consider all students above the mean to have the same level of expertise, even if the students just above are closer to the ones just below than the very top. A future approach could be to divide the group into additional categories, but this would necessitate more participants in order to achieve the same statistical results. Another option is to employ statistical approaches based on regression. Nonetheless, utilizing a rather basic pretest to establish expertise, we were able to uncover numerous significant parts of the real-time feedback by dividing on mean. Prior information obtained through a questionnaire might have also been used to properly divide the groups and provide context for the participants' expertise.

The fourth limitation stems from the fact that we adopted a within-group design, which can result in an undesired carryover effect. For example, a person may become exhausted from the first task before moving on to the second, negatively impacting the outcome. They may also actually improve at the tasks simply by working on the first one, and then apply what they've learned on the first one to the second. We also used a robust statistical test to try to limit the carryover effect, but in the future, others should explore adopting a between-subjects methodology to confirm or refute this suspicion.

Finally, while our feedback is timed based on cognitive load, our analysis considers the cognitive load throughout the whole of the session. We may very well have examined data closer to the seconds when the feedback is provided to learn more about the effects of feedback on cognitive load, stress, and performance. For example, we could have looked at the cognitive load in the following 30 seconds of receiving feedback(read a popup).

Chapter 7

Conclusion & Contribution

In this work, we implemented a system taking advantage of a robust method for computing cognitive workload based on the low/high frequency ratio of pupil oscillation presented by Duchowski *et al.* [6] to give feedback to student while programming. We base our findings on a within subject experiment with 26 participants where we collected pupillary data using an eye tracker. A pretest was used to split the participants into two groups based on their level of expertise. According to the findings, providing students with feedback while programming did not increase their cognitive load. Regardless of expertise, receiving feedback had a positive effect on performance on the harder task and had no effect on the easier task. The experts had a trend towards having lower cognitive load than novices when programming, especially on easier tasks. The results of this experiment are inline with the previous research done on the subject and heavily implies the feasibility of wearable based feedback system for programming.

Our contribution include:

- A system for giving feedback based on the computation of cognitive load based on input from an eye tracker.
- An insight into the effects of such a system on the participant in a programming environment both looking at cognitive load and their previous expertise.

Bibliography

[1] J. Sweller, 'Cognitive load during problem solving: Effects on learning', *Cognitive science*, vol. 12, no. 2, pp. 257–285, 1988.

[2] S. G. Hart and L. E. Staveland, 'Development of nasa-tlx (task load index): Results of empirical and theoretical research', in *Advances in psychology*, vol. 52, Elsevier, 1988, pp. 139–183.

[3] C. Mills, I. Fridman, W. Soussou, D. Waghray, A. M. Olney and S. K. D'Mello, 'Put your thinking cap on: Detecting cognitive load using eeg during learning', in *Proceedings of the seventh international learning analytics & knowledge conference*, 2017, pp. 80–89.

[4] A. T. Duchowski, K. Krejtz, I. Krejtz *et al.*, 'The index of pupillary activity: Measuring cognitive load vis-à-vis task difficulty with pupil oscillation', in *Proceedings of the 2018 CHI conference on human factors in computing systems*, 2018, pp. 1–13.

[5] S. P. Marshall, 'The index of cognitive activity: Measuring cognitive workload', in *Proceedings of the IEEE 7th conference on Human Factors and Power Plants*, IEEE, 2002, pp. 7–7.

[6] A. T. Duchowski, K. Krejtz, N. A. Gehrer, T. Bafna and P. Bækgaard, 'The low/high index of pupillary activity', in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–12, ISBN: 9781450367080. [Online]. Available: https://doi.org/10.1145/3313831.3376394.

[7] P. Vanneste, A. Raes, J. Morton *et al.*, 'Towards measuring cognitive load through multimodal physiological data', *Cognition, Technology & Work*, vol. 23, no. 3, pp. 567–585, 2021.

[8] M. Andrzejewska and A. Skawińska, 'Examining students' intrinsic cognitive load during program comprehension–an eye tracking approach', in *International Conference on Artificial Intelligence in Education*, Springer, 2020, pp. 25–30.

[9] F. G. Paas, 'Training strategies for attaining transfer of problem-solving skill in statistics: A cognitive-load approach.', *Journal of educational psychology*, vol. 84, no. 4, p. 429, 1992.

[10] F. Paas, J. E. Tuovinen, H. Tabbers and P. W. Van Gerven, 'Cognitive load measurement as a means to advance cognitive load theory', *Educational psychologist*, vol. 38, no. 1, pp. 63–71, 2003.

[11] A. Korbach, R. Brünken and B. Park, 'Measurement of cognitive load in multimedia learning: A comparison of different objective measures', *Instructional science*, vol. 45, no. 4, pp. 515–536, 2017.

[12] K. Krejtz, A. T. Duchowski, A. Niedzielska, C. Biele and I. Krejtz, 'Eye tracking cognitive load using pupil diameter and microsaccades with fixed gaze', *PloS one*, vol. 13, no. 9, e0203629, 2018.

[13] P. van der Wel and H. van Steenbergen, 'Pupil dilation as an index of effort in cognitive control tasks: A review', *Psychonomic bulletin & review*, vol. 25, no. 6, pp. 2005–2015, 2018.

[14] E. Granholm, R. F. Asarnow, A. J. Sarkin and K. L. Dykes, 'Pupillary responses index cognitive resource limitations', *Psychophysiology*, vol. 33, no. 4, pp. 457–461, 1996.

[15] B. Gollan, M. Haslgrübler and A. Ferscha, 'Demonstrator for extracting cognitive load from pupil dilation for attention management services', in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, 2016, pp. 1566–1571.

[16] J. Zagermann, U. Pfeil and H. Reiterer, 'Measuring cognitive load using eye tracking technology in visual computing', in *Proceedings of the sixth workshop on beyond time and errors on novel evaluation methods for visualization*, 2016, pp. 78–85.

[17] E. Haapalainen, S. Kim, J. F. Forlizzi and A. K. Dey, 'Psycho-physiological measures for assessing cognitive load', in *Proceedings of the 12th ACM international conference on Ubiquitous computing*, 2010, pp. 301–310.

[18] M. Gjoreski, M. Luštrek and V. Pejović, 'My watch says i'm busy: Inferring cognitive load with low-cost wearables', in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, 2018, pp. 1234–1240.

[19] K. Kyriakou, B. Resch, G. Sagl *et al.*, 'Detecting moments of stress from measurements of wearable physiological sensors', *Sensors*, vol. 19, no. 17, p. 3805, 2019.

[20] M. Gjoreski, H. Gjoreski, M. Luštrek and M. Gams, 'Continuous stress detection using a wrist device: In laboratory and real life', in *proceedings of the 2016 ACM international joint conference on pervasive and ubiquitous computing: Adjunct*, 2016, pp. 1185–1193.

[21] E. Di Lascio, S. Gashi and S. Santini, 'Unobtrusive assessment of students' emotional engagement during lectures using electrodermal activity sensors', *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 3, pp. 1–21, 2018.

[22] B. Wisniewski, K. Zierer and J. Hattie, 'The power of feedback revisited: A meta-analysis of educational feedback research', *Frontiers in Psychology*, vol. 10, p. 3087, 2020.

[23] J. Hattie and H. Timperley, 'The power of feedback', *Review of educational research*, vol. 77, no. 1, pp. 81–112, 2007.

[24] J. A. Fredricks, P. C. Blumenfeld and A. H. Paris, 'School engagement: Potential of the concept, state of the evidence', *Review of educational research*, vol. 74, no. 1, pp. 59–109, 2004.

[25] K. A. Ericsson and W. Kintsch, 'Long-term working memory.', *Psychological review*, vol. 102, no. 2, p. 211, 1995.

[26] J. Sweller and P. Chandler, 'Why some material is difficult to learn', *Cognition and instruction*, vol. 12, no. 3, pp. 185–233, 1994.

[27] S. Wiedenbeck, 'Novice/expert differences in programming skills', *International Journal of Man-Machine Studies*, vol. 23, no. 4, pp. 383–390, 1985.

[28] S. Jessup, S. M. Willis, G. Alarcon and M. Lee, 'Using eye-tracking data to compare differences in code comprehension and code perceptions between expert and novice programmers', in *Proceedings of the 54th Hawaii International Conference on System Sciences*, 2021, p. 114.

[29] A. S. Najar, A. Mitrovic and K. Neshatian, 'Utilizing eye tracking to improve learning from examples', in *International Conference on Universal Access in Human-Computer Interaction*, Springer, 2014, pp. 410–418.

[30] T. Van Gog, F. Paas and J. J. Van Merriënboer, 'Effects of process-oriented worked examples on troubleshooting transfer performance', *Learning and Instruction*, vol. 16, no. 2, pp. 154–164, 2006.

[31] A. Armougum, E. Orriols, A. Gaston-Bellegarde, C. Joie-La Marle and P. Piolino, 'Virtual reality: A new method to investigate cognitive load during navigation', *Journal of Environmental Psychology*, vol. 65, p. 101 338, 2019.

[32] *Tobii pro glasses 2*, https://www.tobiipro.com/product-listing/tobii-pro-glasses-2/, 2022.

[33] D. De Tommaso and A. Wykowska, 'Tobiiglassespysuite: An open-source suite for using the tobii pro glasses 2 in eye-tracking studies', in *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, ser. ETRA '19, Denver, Colorado: ACM, 2019, 46:1–46:5, ISBN: 978-1-4503-6709-7. DOI: 10.1145/3314111.3319828. [Online]. Available: http://doi.acm.org/10.1145/3314111.3319828.

[34] https://www.tobiipro.com/product-listing/tobii-pro-glasses-2-sdk/, 2022.

[35] R. Engbert and R. Kliegl, 'Microsaccades uncover the orientation of covert attention', *Vision research*, vol. 43, no. 9, pp. 1035–1045, 2003.

Appendix A

Adjusted LHIPA code

```
import math
import pywt
import numpy as np


def modmax(d):
    # compute signal modulus
    m = [0.0] * len(d)
    for i in range(len(d)):
        m[i] = math.fabs(d[i])
    # if value is larger than both neighbours , and strictly
    # larger than either , then it is a local maximum
    t = [0.0] * len(d)
    for i in range(len(d)):
        ll = m[i - 1] if i >= 1 else m[i]
        oo = m[i]
        rr = m[i + 1] if i < len(d) - 2 else m[i]

        if (ll <= oo and oo >= rr) and (ll < oo or oo > rr):
            # compute magnitude
            t[i] = math.sqrt(d[i] ** 2)
        else:
            t[i] = 0.0
    return t
```

```
def lhipa(d: np.ndarray, tt: float) -> float:
    """
    Computes the lhipa over the d datapoints

    Parameters:
    -----------
    d: np.ndarray
    Pupil dilation.

    tt: float
    Time in seconds.


    Returns:
    --------
    LHIPA: float

    """
    # find max decomposition level
    w = pywt.Wavelet('db6')
    maxlevel = pywt.dwt_max_level(len(d), filter_len=w.dec_len)

    # set high and low frequency band indices
    hif, lof = 1, int(maxlevel / 2)

    # get detail coefficients of pupil diameter signal d
    cD_H = pywt.downcoef('d', d, 'db6', 'per', level=hif)
    cD_L = pywt.downcoef('d', d, 'db6', 'per', level=lof)

    # normalize by 1/ sqrt(2 ^j)
    cD_H[:] = [x / math.sqrt(2 ** hif) for x in cD_H]
    cD_L[:] = [x / math.sqrt(2 ** lof) for x in cD_L]

    # obtain the LH:HF ratio
    cD_LH = cD_L
    for i in range(len(cD_L)):
        cD_LH[i] = cD_L[i] / cD_H[int(((2 ** lof) / (2 ** hif)) * i)]

    # detect modulus maxima , see Duchowski et al. [15]
```

```
cD_LHm = modmax(cD_LH)

# threshold using universal threshold lambda_univ = sigma_hat sqrt(2 log n)
# where sigma_hat is the standard deviation of the noise
lamda_univ = np.std(cD_LHm) * math.sqrt(2.0 * np.log2(len(cD_LHm)))
cD_LHt = pywt.threshold(cD_LHm, lamda_univ, mode="less")

# compute LHIPA
ctr = 0
for i in range(len(cD_LHt)):
    if math.fabs(cD_LHt[i]) > 0:
        ctr += 1
LHIPA = float(ctr) / tt
return LHIPA
```

Appendix B

NSD

NSD NORSK SENTER FOR FORSKNINGSDATA

Meldeskjema

Referansenummer

983309

Hvilke personopplysninger skal du behandle?

- Navn (også ved signatur/samtykke)
- Bilder eller videoopptak av personer
- Lydopptak av personer
- Bakgrunnsopplysninger som vil kunne identifisere en person
- Helseopplysninger

Beskriv hvilke bakgrunnsopplysninger du skal behandle

Jobbstilling, tidligere arbeid, Navn, alder, kjønn.

Prosjektinformasjon

Prosjekttittel

Design of a smart gaze-aware feedback system for programming.

Prosjektbeskrivelse

Fokuset til denne avhandlingen er å utvikle et intelligent tilbakemeldingssystem, som kan hjelpe studenter mens de programmerer. Tilbakemeldingen (hjelpen) vil gis i sanntid ved bruk av øyesporingsdata, armbåndsdata og logdata fra datamaskinen de utfører oppgaven på.

Det første som skal gjøres er et fokusgruppeintervju, som skal hjelpe med å finne ut hva slags tilbakemeldingsverktøy som kan hjelpe studentene på best måte.

Det andre som skal gjøres er opptak av skjermen til erfarene programmerere for å finne ut hvordan de ville løst programmeringsoppgaven.

Det tredje og siste som skal gjøres er en AB-test. I testen vil studenter få en programmeringsoppgave som de skal løse og tilbakemeldingsverktøyet fra fokusgruppen vil bli brukt her. Hensikten er å finne ut om

tilbakemeldingsverktøyet er effektivt som hjelpemiddel for en student som programmerer.

Begrunn behovet for å behandle personopplysningene

Navn er nødvendig for signatur.
Alder, kjønn er viktig for å forstå kontekst og for å kunne trekke slutninger.
Video og lydopptak trengs for å kunne gå gjennom og analysere fokusgruppen i etterkant.
Jobbstilling er nødvendig for kontekst.
Øyesporer er nødvendig for å kunne få tilgang til kognitiv belastning.
Armbånd trengs for å få tilgang til puls som kan si noe om personens stressnivå.
Video og skjermopptak av utvalg 3 er nødvendig for analysen.

Ekstern finansiering

Type prosjekt

Studentprosjekt, masterstudium

Kontaktinformasjon, student

Torkjell Romskaug, torkjer@stud.ntnu.no, tlf: 94178571

Behandlingsansvar

Behandlingsansvarlig institusjon

Norges teknisk-naturvitenskapelige universitet / Fakultet for informasjonsteknologi og elektroteknikk (IE) / Institutt for datateknologi og informatikk

Prosjektansvarlig (vitenskapelig ansatt/veileder eller stipendiat)

Kshitij Sharma, kshitij.sharma@ntnu.no, tlf: 41209147

Skal behandlingsansvaret deles med andre institusjoner (felles behandlingsansvarlige)?

Nei

Utvalg 1

Beskriv utvalget

Studenter/ansatte ved NTNU.

Rekruttering eller trekking av utvalget

Rekruteres gjennom bekjente og medarbeidere til veileder.

Alder

22 - 50

Inngår det voksne (18 år +) i utvalget som ikke kan samtykke selv?

Nei

Personopplysninger for utvalg 1

- Navn (også ved signatur/samtykke)
- Bilder eller videoopptak av personer
- Lydopptak av personer
- Bakgrunnsopplysninger som vil kunne identifisere en person

Hvordan samler du inn data fra utvalg 1?

Gruppeintervju

Grunnlag for å behandle alminnelige kategorier av personopplysninger

Samtykke (art. 6 nr. 1 bokstav a)

Informasjon for utvalg 1

Informerer du utvalget om behandlingen av opplysningene?

Ja

Hvordan?

Skriftlig informasjon (papir eller elektronisk)

Utvalg 2

Beskriv utvalget

Studenter innenfor programmering og innformatikk som har gått mer enn 3 års skolegang.

Rekruttering eller trekking av utvalget

Rekrutering gjennom nettverk.

Alder

23 - 26

Inngår det voksne (18 år +) i utvalget som ikke kan samtykke selv?

Nei

Personopplysninger for utvalg 2

- Navn (også ved signatur/samtykke)
- Bilder eller videoopptak av personer

Hvordan samler du inn data fra utvalg 2?

Annet

Beskriv

Dataskilder:

Øyesporer (tobii eyetracker 60hz): pupilldilatasjon, pupilldiameter, gazepoints (blikk-koordinater).

datamaskin: Tastetrykk.

Utvalg 2 sitter foran en pc og gjennomfører en oppgave.

Grunnlag for å behandle alminnelige kategorier av personopplysninger

Samtykke (art. 6 nr. 1 bokstav a)

Informasjon for utvalg 2

Informerer du utvalget om behandlingen av opplysningene?

Ja

Hvordan?

Skriftlig informasjon (papir eller elektronisk)

Utvalg 3

Beskriv utvalget

Studenter ved ntnu med litt/ moderat kunnskap om programmering.

Rekruttering eller trekking av utvalget

Rekruttering vil gjøres ved at skolen tar kontakt med elevene og spørr om de vil være med. Det kan også tenkes at det gjøres ved å bruke netverket vårt.

Alder

19 - 30

Inngår det voksne (18 år +) i utvalget som ikke kan samtykke selv?

Nei

Personopplysninger for utvalg 3

- Navn (også ved signatur/samtykke)
- Bilder eller videoopptak av personer
- Helseopplysninger

Hvordan samler du inn data fra utvalg 3?

Annet

Beskriv

Dataskilder:

Øyesporer (tobii eyetracker 60hz): pupilldilatasjon, pupilldiameter.

armbånd: (1) hjerterytme ved 1 Hz, (2) elektrodermisk aktivitet (EDA) ved 64Hz, (3) kroppstemperatur ved 4Hz og (4) slagvolum puls ved 4Hz.

datamaskin: Tastetrykk, opptak av skjerm og bevegelse av mus.

videokamera: video av deltager.

Grunnlag for å behandle alminnelige kategorier av personopplysninger

Samtykke (art. 6 nr. 1 bokstav a)

Grunnlag for å behandle særlige kategorier av personopplysninger

Uttrykkelig samtykke (art. 9 nr. 2 bokstav a)

Redegjør for valget av behandlingsgrunnlag

Informasjon for utvalg 3

Informerer du utvalget om behandlingen av opplysningene?

Ja

Hvordan?

Skriftlig informasjon (papir eller elektronisk)

Tredjepersoner

Skal du behandle personopplysninger om tredjepersoner?

Nei

Dokumentasjon

Hvordan dokumenteres samtykkene?

- Manuelt (papir)

Hvordan kan samtykket trekkes tilbake?

Deltakere får mulighet til å trekke tilbake samtykket ved å sende en email til de ansvarlige for prosjektet.

Hvordan kan de registrerte få innsyn, rettet eller slettet opplysninger om seg selv?

De får mailen vår og kan sende etterspørsel etter sin egen data.

Totalt antall registrerte i prosjektet

1-99

Tillatelser

Skal du innhente følgende godkjenninger eller tillatelser for prosjektet?

Behandling

Hvor behandles opplysningene?

- Maskinvare tilhørende behandlingsansvarlig institusjon

Hvem behandler/har tilgang til opplysningene?

- Student (studentprosjekt)
- Prosjektansvarlig

Tilgjengeliggjøres opplysningene utenfor EU/EØS til en tredjestat eller internasjonal organisasjon?

Nei

Sikkerhet

Oppbevares personopplysningene atskilt fra øvrige data (koblingsnøkkel)?

Ja

Hvilke tekniske og fysiske tiltak sikrer personopplysningene?

- Opplysningene anonymiseres fortløpende
- Adgangsbegrensning

Varighet

Prosjektperiode

08.11.2021 - 01.07.2022

Skal data med personopplysninger oppbevares utover prosjektperioden?

Nei, data vil bli oppbevart uten personopplysninger (anonymisering)

Hvilke anonymiseringstiltak vil bli foretatt?

- Lyd- eller bildeopptak slettes
- Personidentifiserbare opplysninger fjernes, omskrives eller grovkategoriseres

Vil de registrerte kunne identifiseres (direkte eller indirekte) i oppgave/avhandling/øvrige publikasjoner fra prosjektet?

Nei

Tilleggsopplysninger

La til videoopptak av deltager på utvalg 3 og opptak av pupildilatasjon og pupildiameter på utvalg 2. La også til at det kommer til å være en forhåndsprøve der deltagerene i utvalg 3 vil bli gitt flere små programmerings oppgaver der man bruker det samme innsamling som var tilstede i nsden fra før.

Appendix C

Consent form

Request for participation in research project

Design of a smart gaze-aware feedback system for programming

Background and Purpose

Design of a smart gaze-aware feedback system for programming is a master project at the Norwegian University of Science and Technology (NTNU). The focus of the thesis is to develop an intelligent feedback system that helps the students while they are programming. This help should be provided in real-time using the eye-tracking data from the student and the log data from the IDE that the student is using.

Sample selection: The participants will be primarily selected based on their participation in the first-year programming course object-oriented programming at NTNU. The students need to be familiar with the programming language and have some experience with programming simple tasks. The participants will be recruited by spoken word or through the course sending out information that students are needed.

What does participation in the project imply?

Participation in this study entails a pretest, and solving a programming task that together should take between 30 - 60 minutes to complete. The participant will be programming while an eye tracker, a wristband, and keystrokes are monitored and stored. In addition, the screen and the participant will be recorded by a screen recorder and a video camera. Eye-tracking is to film the eyes with a high resolution camera that is zoomed in on the eyes. For eye-tracking we will use:(tobii eyetracker 60hz): and extract pupil dilation and pupil diameter. From the wristband we will collect: (1) heart rate at 1 Hz, (2) electrodermal activity (EDA) at 64Hz, (3) body temperature at 4Hz og (4) blood volume pulse at 4Hz.

The participant will be assigned randomly to group A or group B. Group A will be given an extra tool while completing the programming task, while group B will complete the task without the tool. Then they will switch and the ones that did not receive the extra tool for the first programming task will get it for the next one and vice versa.

After the programming tasks the participant will rank the tasks with a NASA-TLX questionnaire.

What will happen to the information about you?

The data will be collected with the respective devices at NTNU. The devices will be connected to the internet, but the data will be analysed locally from the researchers that are taking part in this project. Your data will be anonymized (the names will be stored separately from the data) after we have transcribed and reviewed the data. The project is scheduled for completion by 2022. Once the interview is conducted, the data will be stored in the one drive that is password protected and will be kept, and used by us until the end of spring 2022. The data might be kept anonymized for further research in the future.

Voluntary participation

Participation in this study is voluntary, and you as participants can choose to withdraw your consent at any time during the project without stating any reason. If you decide to withdraw, all your personal data will be deleted and not used in the project. If you have any questions concerning the project or the study, please contact Elias Larsen at EliasSL@stud.ntnu.no, Torkjell Romskaug at TorkjeR@stud.ntnu.no, or Kshitij Sharma at kshitij.sharma@ntnu.no.

The study has been notified to the Data Protection Official for Research, NSD - Norwegian Centre for Research Data.

Your rights

So long as you can be identified in the collected data, you have the right to:

- access the personal data that is being processed about you
- request that your personal data is deleted
- request that incorrect personal data about you is corrected/rectified
- receive a copy of your personal data (data portability), and

send a complaint to the Data Protection Officer or The Norwegian Data Protection Authority regarding the processing of your personal data.

Where can I find out more?

If you have questions about the project, or want to exercise your rights, contact:

- Norwegian University of Science and Technology via *Kshitij Sharma.*
- NSD – The Norwegian Centre for Research Data AS, by email: (personverntjenester@nsd.no) or by telephone: +47 53 21 15 00.

Consent for participation in the study

I hereby confirm that I have been fully informed about the aims and purposes of the study and the project in general. I understand that my participation is entirely voluntary and, if I no longer wish to participate, I may at any stage withdraw my participation. I have been informed and understand my participation in the focus group interview, and that the data will be stored and analyzed for the purposes of the project.

I have received information about the project and am willing to participate

---

(Signed by participant, date)

Appendix D

Pretest

```
for i in range(10):
    if i == 5:
        break
    else:
        print(i)
else:
    print("Here")
```

a) 0 1 2 3 4 Here
b) 0 1 2 3 4 5 Here
c) 0 1 2 3 4
d) 1 2 3 4 5

```
for i in range(5):
    if i == 5:
        break
    else:
        print(i)
else:
    print("Here")
```

a) 0 1 2 3 4 Here
b) 0 1 2 3 4 5 Here
c) 0 1 2 3 4
d) 1 2 3 4 5

```
a = [0, 1, 2, 3]
i = -2
for i not in a:
    print(i)
    i += 1
```

a) -2 -1
b) 0
c) error
d) none of the mentioned

```
class Demo:
    def __new__(self):
        self.__init__(self)
        print("Demo's __new__() invoked")

    def __init__(self):
        print("Demo's __init__() invoked")

class Derived_Demo(Demo):
    def __new__(self):
        print("Derived_Demo's __new__() invoked")

    def __init__(self):
        print("Derived_Demo's __init__() invoked")

def main():
obj1 = Derived_Demo()
obj2 = Demo() main()
```

a)
Derived_Demo's __init__() invoked
Derived_Demo's __new__() invoked
Demo's __init__() invoked
Demo's __new__() invoked

b)
Derived_Demo's __new__() invoked
Demo's __init__() invoked
Demo's __new__() invoked

c)
Derived_Demo's __new__() invoked
Demo's __new__() invoked

d)
Derived_Demo's __init__() invoked
Demo's __init__() invoked

```
class Test:
	def __init__(self):
		self.x = 0

class Derived_Test(Test):
	def __init__(self):
		self.y = 1
def main():
	b = Derived_Test()
	print(b.x,b.y)

main()
```

a) 0 1
b) 0 0
c) Error because class B inherits A but variable x isn't inherited
d) Error because when object is created, argument must be passed like Derived_Test(1)

```
count={}
count[(1,2,4)] = 5
count[(4,2,1)] = 7
count[(1,2)] = 6
count[(4,2,1)] = 2
tot = 0
for i in count:
        tot=tot+count[i]
print(len(count)+tot)
```

a) 25
b) 17
c) 16
d) Tuples can’t be made keys of a dictionary

```
def fn(**kwargs):
    for emp, age in kwargs.items():
    print ("%s's age is %s." %(emp, age))

fn(John=25, Kalley=22, Tom=32)
```

OUTPUT

```
class PC: # Base class
    processor = "Xeon" # Common attribute
    def set_processor(self, new_processor):
        processor = new_processor

class Desktop(PC): # Derived class
    os = "Mac OS High Sierra" # Personalized attribute
    ram = "32 GB"

class Laptop(PC): # Derived class
    os = "Windows 10 Pro 64" # Personalized attribute
    ram = "16 GB"

desk = Desktop()
print(desk.processor, desk.os, desk.ram)

lap = Laptop()
print(lap.processor, lap.os, lap.ram)
```

OUTPUT

```
class PC: # Base class
    processor = "Xeon" # Common attribute
    def __init__(self, processor, ram):
        self.processor = processor
        self.ram = ram
    def set_processor(self, new_processor):
        processor = new_processor
    def get_PC(self):
        return "%s cpu & %s ram" % (self.processor, self.ram)

class Tablet():
    make = "Intel"
    def __init__(self, processor, ram, make):
        self.PC = PC(processor, ram) # Composition
        self.make = make

    def get_Tablet(self):
        return "Tablet with %s CPU & %s ram by %s" % (self.PC.processor, self.PC.ram, self.make)

if __name__ == "__main__":
    tab = Tablet("i7", "16 GB", "Intel")
    print(tab.get_Tablet())
```

OUTPUT:

```
def multiply_number(num):
    def product(number):
        'product() here is a closure'
        return num * number
    return product

num_2 = multiply_number(2)
print(num_2(11))
print(num_2(24))

num_6 = multiply_number(6)
print(num_6(1))
```

OUTPUT

Appendix E

Snake debugging task with hints

```
import pygame
import time
import random

pygame.init()

white = (255, 255, 255)
yellow = (255, 255, 102)
black = (0, 0, 0)
red = (213, 50, 80)
green = (0, 255, 0)
blue = (50, 153, 213)

dis_width = 600
dis_height = 400

dis = pygame.display.set_mode((dis_width, dis_height))
clock = pygame.time.Clock()

snake_block = 10
snake_speed = 15

font_style = pygame.font.SysFont("bahnschrift", 25)
score_font = pygame.font.SysFont("comicsansms", 35)
```

```
def Your_score(score):
    value = score_font.render("Your Score: " + str(score), True, yellow)
    dis.blit(value, [0, 0])



def our_snake(snake_block, snake_list):
    for x in snake_list:
        pygame.draw.rect(dis, black, [x[0], x[1], snake_block, snake_block])


def message(msg, color):
    mesg = font_style.render(msg, True, color)
    dis.blit(mesg, [dis_width / 6, dis_height / 3])


def gameLoop():
    game_over = False
    game_close = False

    x1 = dis_width / 2
    y1 = dis_height / 2

    x1_change = 0
    y1_change = 0

    snake_List = []
    Length_of_snake = 1

    foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
    foody = round(random.randrange(0, dis_height - snake_block) / 10.0) * 10.0

    while not game_over:

        while game_close == True:
            dis.fill(blue)
            message("You Lost! Press C-Play Again or Q-Quit", red)
            Your_score(Length_of_snake - 1)
            pygame.display.update()
```

```
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_q:
                game_over = True
                game_close = False
            if event.key == pygame.K_c:
                gameLoop()

for event in pygame.event.get(): #1 change in this loop to rectfy moving
    if event.type == pygame.QUIT:
        game_over = True
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_LEFT:
            x1_change = snake_block
            y1_change = snake_block
        elif event.key == pygame.K_RIGHT:
            x1_change = snake_block
            y1_change = snake_block
        elif event.key == pygame.K_UP:
            y1_change = snake_block
            x1_change = snake_block
        elif event.key == pygame.K_DOWN:
            y1_change = snake_block
            x1_change = snake_block

if x1 > dis_height or y1 > dis_width: #2 change this to rectify game over
    game_close = True
x1 = x1_change #3 change this to rectify moving
y1 = y1_change #4 change this to rectify moving
dis.fill(blue)
pygame.draw.rect(dis, green, [foodx, foody, snake_block, snake_block])
snake_Head = []
snake_Head.append(x1)
snake_Head.append(y1)
snake_List.append(snake_Head)
if len(snake_List) > Length_of_snake:
    del snake_List[0]
```

```
        for x in snake_List[:-1]:
            if x == snake_Head:
                game_close = True

        our_snake(snake_block, snake_List)
        Your_score(snake_List[0]) #5 change this for increasing the score properly

        pygame.display.update()

        if x1 == foodx and y1 == foody:
            foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
            foody = round(random.randrange(0, dis_height - snake_block) / 10.0) * 10.0
            Length_of_snake = 1 #6 change this for increasing the length of the snake with the

        clock.tick(snake_speed)

    pygame.quit()
    quit()


gameLoop()
```

Appendix F

Tetris debugging task with hints

```
import pygame
import random

colors = [
    (0, 0, 0),
    (120, 37, 179),
    (100, 179, 179),
    (80, 34, 22),
    (80, 134, 22),
    (180, 34, 22),
    (180, 34, 122),
]


class Figure:
    x = 0
    y = 0
    # Rectify this array to obtain correct tetris figures
    figures = [
        [[1, 5, 9, 13], [4, 5, 6, 7]],
        [[4, 5, 9, 10], [2, 6, 5, 9]],
        [[6, 7, 9, 10], [1, 5, 6, 10]],
        [[1, 2, 5, 9], [0, 4, 5, 6], [6, 7, 9, 10], [4, 5, 6, 10]],
        [[1, 2, 6, 10], [5, 6, 7, 9], [2, 6, 10, 11], [6, 7, 9, 10]],
        [[1, 4, 5, 6], [6, 7, 9, 10], [4, 5, 6, 9], [1, 5, 6, 9]],
        [[1, 2, 5, 6]],
```

```
    ]

    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.type = random.randint(0, len(self.figures) - 1)
        self.color = random.randint(1, len(colors) - 1)
        self.rotation = 0

    def image(self):
        return self.figures[self.type][self.rotation]

    def rotate(self):
        self.rotation = (self.rotation + 1) % len(self.figures[self.type])


class Tetris:
    level = 2
    score = 0
    state = "start"
    field = []
    height = 0
    width = 0
    x = 100
    y = 60
    zoom = 20
    figure = None

    def __init__(self, height, width):
        self.height = height
        self.width = width
        self.field = []
        self.score = 0
        self.state = "start"
        for i in range(height):
            new_line = []
            for j in range(width):
                new_line.append(0)
            self.field.append(new_line)
```

```
    def new_figure(self):
        self.figure = Figure(3, 0)

    def intersects(self):
        intersection = False
        for i in range(10):  # Rectify both for loops to properly insert the tetris
            # figures in the building
            for j in range(10):
                if i * 4 + j in self.figure.image():
                    if i + self.figure.y > self.height - 1 or \
                            j + self.figure.x > self.width - 1 or \
                            j + self.figure.x < 0 or \
                            self.field[i + self.figure.y][j + self.figure.x] > 0:
                        intersection = True
        return intersection

    def break_lines(self):
        lines = 0
        for i in range(self.height):  # rectify the range call to correctly remove
            # completed lines
            zeros = 0
            for j in range(self.width):
                if self.field[i][j] == 0:
                    zeros += 1
            if zeros == 0:
                lines += 1
                for i1 in range(i, 1, -1):
                    for j in range(self.width):
                        self.field[i1][j] = self.field[i1 - 1][j]
        self.score += lines ** 2

    def go_space(self):
        while not self.intersects():
            self.figure.y += 1
        self.figure.y += 1  # rectify this to properly pull down the tetris piece
        self.freeze()

    def go_down(self):
```

```
        self.figure.y += 1
        if self.intersects():
            self.figure.y += 1  # rectify this to properly move the tetris piece down
            self.freeze()

    def freeze(self):
        for i in range(4):
            for j in range(4):
                if i * 4 + j in self.figure.image():
                    self.field[i + self.figure.y][j + self.figure.x] = self.figure.color
        self.break_lines()
        self.new_figure()
        if self.intersects():
            self.state = "gameover"

    def go_side(self, dx):
        old_x = self.figure.x
        self.figure.x += dx
        if self.intersects():
            self.figure.x = old_x

    def rotate(self):
        old_rotation = self.figure.rotation
        self.figure.rotate()
        if self.intersects():
            self.figure.rotation = old_rotation


# Initialize the game engine
pygame.init()

# Define some colors
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
GRAY = (128, 128, 128)

size = (400, 500)
screen = pygame.display.set_mode(size)
```

```
pygame.display.set_caption("Tetris")

# Loop until the user clicks the close button.
done = False
clock = pygame.time.Clock()
fps = 25
game = Tetris(20, 10)
counter = 0

pressing_down = False

while not done:
    if game.figure is None:
        game.new_figure()
    counter += 1
    if counter > 100000:
        counter = 0

    if counter % (fps // game.level // 2) == 0 or pressing_down:
        if game.state == "start":
            game.go_down()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                game.rotate()
            if event.key == pygame.K_DOWN:
                pressing_down = True
            if event.key == pygame.K_LEFT:
                game.go_side(1)  # rectify this to move the piece left
            if event.key == pygame.K_RIGHT:
                game.go_side(1)
            if event.key == pygame.K_SPACE:
                game.go_space()
            if event.key == pygame.K_ESCAPE:
                game.__init__(20, 10)
```

```
    if event.type == pygame.KEYUP:
            if event.key == pygame.K_DOWN:
                pressing_down = False

    screen.fill(WHITE)

    for i in range(game.height):
        for j in range(game.width):
            pygame.draw.rect(screen, GRAY, [game.x + game.zoom * j, game.y + game.zoom * i, gam
            if game.field[i][j] > 0:
                pygame.draw.rect(screen, colors[game.field[i][j]],
                                 [game.x + game.zoom * j + 1, game.y + game.zoom * i + 1, game

    if game.figure is not None:
        for i in range(4):
            for j in range(4):
                p = i * 4 + j
                if p in game.figure.image():
                    pygame.draw.rect(screen, colors[game.figure.color],
                                     [game.x + game.zoom * (j + game.figure.x) + 1,
                                      game.y + game.zoom * (i + game.figure.y) + 1,
                                      game.zoom - 2, game.zoom - 2])

    font = pygame.font.SysFont('Calibri', 25, True, False)
    font1 = pygame.font.SysFont('Calibri', 65, True, False)
    text = font.render("Score: " + str(game.score), True, BLACK)
    text_game_over = font1.render("Game Over", True, (255, 125, 0))
    text_game_over1 = font1.render("Press ESC", True, (255, 215, 0))

    screen.blit(text, [0, 0])
    if game.state == "gameover":
        screen.blit(text_game_over, [20, 200])
        screen.blit(text_game_over1, [25, 265])

    pygame.display.flip()
    clock.tick(fps)

pygame.quit()
```

Elias Sagmo Larsen, Torkjell Romskaug

Real time stress-aware feedback system for programming.

NTNU
Kunnskap for en bedre verden