Anniken Holst
Marianne Magnussen

Code Flip: A Game for the Norwegian Elective Course of Programming

Master's thesis in Datateknologi
Supervisor: Monica Divitini
June 2019

Master's thesis

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

NTNU
Norwegian University of
Science and Technology

Anniken Holst
Marianne Magnussen

Code Flip: A Game for the Norwegian Elective Course of Programming

Master's thesis in Datateknologi
Supervisor: Monica Divitini
June 2019

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

NTNU
Norwegian University of
Science and Technology

Summary

Programming is a skill that gets more relevant by the day. The fall of 2016, The Elective Course of Programming was introduced to Norwegian lower secondary schools, in an attempt to attract more people to IT studies. The course quickly caught on, and it is now offered by many Norwegian schools. It is however seen that programming is a demanding topic to learn, and that several students lack motivation for learning it. Educational games are seen through research to increase motivation and knowledge. An educational game supporting both students and teachers is thus likely to be useful for the Elective Course of Programming.

This research addresses how a game can introduce basic programming concepts to lower secondary school students, in the context of the Elective Course of Programming. The students' perspective on what to include in the game was covered through a specialization project, performed by the authors of this research. There it was found that popular game elements among lower secondary school students are competition, cooperation and graphics. In this thesis these initial results are integrated with the teachers' perspective in order to develop a game that targets both stakeholders. This is done by interviewing 13 teachers of the Elective Course of Programming. Findings are that challenges faced by teachers of the elective course are connected to the motivation of students, and varying levels of programming skills among both students and teachers. The desires of teachers for a programming game are that it should be possible to monitor the progress of students, that it should encourage problem-solving, and that students should cooperate in teams. It is also seen that the teachers have positive attitudes towards educational games.

Three different game concepts based on the gathered requirements of students and teachers are proposed. Teachers' opinions about game elements of the different concepts are collected through a survey at a teacher conference, in order to evaluate the concepts and choose one to proceed with. Finally, the game Code Flip is developed in an iterative way, where it is continuously evaluated and improved. A final version of the game is evaluated in a class of the Elective Course of Programming.

Code Flip is a web-based game where students compete in teams and solve programming tasks to uncover a picture. The game has two goals, where one is to guess what the picture displays, and the other is to be the first team to solve all the programming tasks. Through offering analytics during and after game-play, teachers of the ECP can monitor their students' progress, which can help with following up on students and evaluating them. A demo of the game can be seen at `https://youtu.be/kevnpYWTbmU`.

The classroom evaluation of Code Flip suggests that the game successfully motivates students for solving programming tasks, and increases their confidence for programming. The game analytics are seen to be useful to the teacher, and some suggestions for new features of the game are gathered. These are suggested for future work. The source code of Code Flip, MIT-licensed, is available on GitHub[1].

[1] `https://github.com/annikh/CodeFlip`

Preface

This master thesis was written for the Department of Computer and Information Science, at the Norwegian University of Science and Technology. It is the final fulfillment for a master's degree in computer science, and was written the spring of 2019.

We would like to thank our supervisor, Professor Monica Divitini, for her excellent guidance and valuable feedback through the work. You have really helped us structure our work, stay on track, and given us many opportunities through your network and expertise.

We would also like to thank:

- The 13 teachers who voluntarily participated in the interviews.
- The 23 teachers from the LKK-conference who answered our survey.
- Our fellow computer science students who volunteered to pilot test the game Code Flip.
- The game and teacher experts who helped us with the expert evaluation of the game.
- The lower secondary school teacher who arranged for us to test the game Code Flip with a class of the Elective Course of Programming.
- The 14 students who participated in the evaluation of Code Flip, playing the game and giving us valuable feedback.

Trondheim, June, 2019

Table of Contents

List of Tables

List of Figures

Abbreviations

IT	Information Technology
LSS	Lower Secondary School
ECP	Elective Course of Programming
NTNU	Norwegian University of Science and Technology
UDIR	Norwegian Directorate for Education and Training (Utdanningsdirektoratet)
LKK	Lær Kidsa Koding
GDPR	General Data Protection Regulation

Chapter 1

Introduction

1.1 Motivation

Programming is a skill that gets more relevant by the day, as it is a demand for a growing amount of the jobs of the future. The Norwegian government wishes to encourage more people to learn to program. By arousing the interest of youths, they hope more people will choose a career in science and technology (Regjeringen 2016a). The fall of 2016 the government started a pilot project with an elective course to teach programming in select lower secondary schools (LSS) for students between 12 and 16 years old. The pilot project was popular, and the next year every LSS in the country could choose to offer the *Elective Course of Programming* to their students (Regjeringen 2017). In 2020, programming will be included in several compulsory courses as well, such as Mathematics, Natural Sciences, Social Studies, Music and Arts & Crafts (Regjeringen 2018). However, the education of teachers in Norway does not yet reflect the introduction of programming in school (Jørgenrud 2016). Programming courses are in fact being introduced to the teacher educations and as an offer to those who are practicing as teachers, for instance NTNU offers two courses for practicing teachers that can be taken online[1]. Nonetheless, the demand for teachers with competence in programming is likely to outgrow the offer quickly.

Many students find programming hard to learn for the first time (Jenkins 2002). The discipline can be perceived as abstract as it is a new way of thinking, and programming tasks popularly chosen as part of education are considered boring. Research also states that the pacing of programming courses is a difficulty. If the course content advances before students have time to gain an understand of basic concepts, they will fall behind and are eventually likely to give up (Moser 1997). Seeing that elective courses, like the Elective Course of Programming, were introduced in Norwegian schools in order to motivate students (Utdanningsdirektoratet 2018), a motivational learning tool is likely to be useful by motivating students and supporting teachers.

Educational games has been a popular field in educational research in the last decade. Previous research has found that games increase motivation and knowledge (Connolly et al. 2012). There already exist a number of games that teach programming. These games are however not specifically designed for use in a classroom setting while targeting a given curriculum. Therefore, this work aims to develop an educational game for teaching of programming for use in the Elective Course of Programming (ECP). The game should motivate students to learn basic

[1] `https://ntnu.no/videre/programmering`

programming concepts, and address the interests of teachers of the ECP, as these will be the end-users.

1.2 Context

This research is done in context of a master thesis at the Norwegian University of Science and Technology, Department of Computer Science. It builds on a specialization project investigating literature of games that teach programming to LSS students, existing programming games, and examining what types of games LSS students like to play. The research is supervised by Professor Monica Divitini.

The thesis takes into account the teacher as a stakeholder, and presents the development and testing of an educational game for teaching programming concepts to LSS students, in the context of the Elective Course of Programming. It is connected to a project at the Center for Excellence in IT Education (Excited), which aims at attracting students to IT education and promoting programming in schools.

1.3 Research Questions

The specialization project (Holst and Magnussen 2018) established a foundation for an educational game that introduces programming concepts to LSS students, focusing mainly on the perspective of students. The authors have yet to conclude with how an educational game can be introduced in the context of the ECP, as it is important to take the teachers into account for this decision. As such, the main research question of the specialization project still remains for the master thesis, but with other sub-questions in order to capture the uncovered areas:

> **RQ1:** How can we make a web-based game that teaches programming concepts to Norwegian lower secondary school students, in the context of the Elective Course of Programming?

For ensuring that the game to be proposed is useful, it should address areas that the teachers find challenging with the Elective Course of Programming. As such, a research question is:

> **RQ1.1:** What challenges do teachers of the Elective Course of Programming face when teaching programming to their students?

The game should offer elements that the teachers want and that will make them more likely to use the game. The wishes of students are not necessarily what is of their own best interest, and neither necessarily the same as what the teachers want. Thus, the teachers' wishes for the game must be taken into account:

> **RQ1.2:** What do teachers of the Elective Course of Programming wish for in a game that teaches programming?

Research of the specialization project indicates that students are motivated through games with elements of competition, cooperation and playfulness. The literature also states that for students, motivation is enough to increase learning. Knowledge will often come as a natural side-effect from being interested in a topic. The game should therefore be designed in order to motivate students for programming:

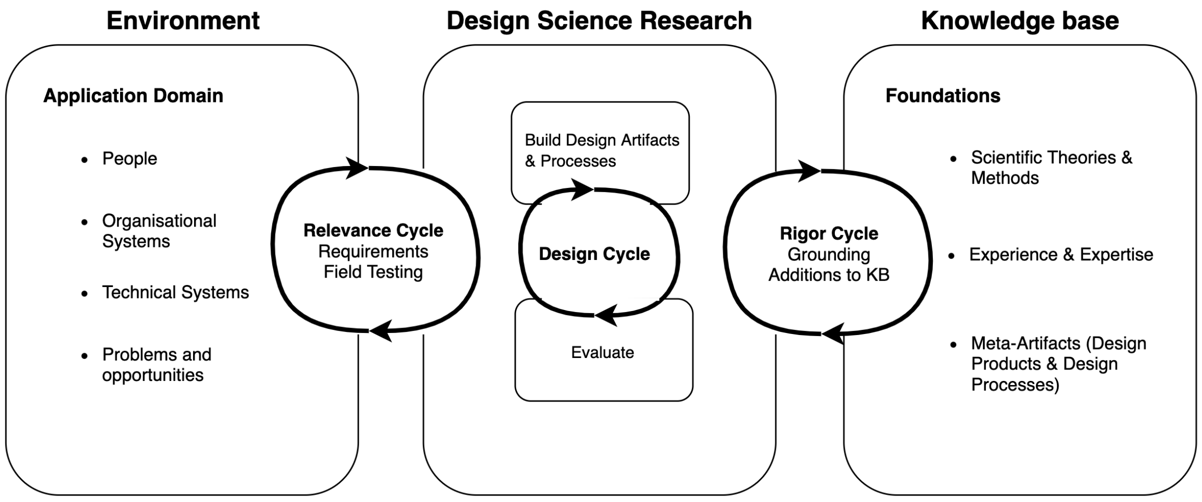


Figure 1.1: Design science research cycles (Hevner 2007).

> **RQ1.3:** By addressing the interests of teachers and students, can we make a game for the Elective Course of Programming that increases LSS students' motivation for programming?

1.4 Research Methods

This master thesis uses a design science approach, which is a methodology for creating innovative artifacts for a specified problem domain (Von Alan et al. 2004). Figure 1.1 shows a three cycle view of design science research, originating from Hevner (2007). The cycles represent three closely related iterative activities that are performed throughout a research in order to improve the functionality of the artifact - the programming game in this case. *The Relevance Cycle* of this project involves interviews, focus groups, surveys, and field testing. *The Rigor Cycle* includes the problem elaboration, state of the art, and the architectural design choices. The design of game concepts, the game development, and the evaluation and modification of the different versions of the game are part of *The Design Cycle*.

The specialization project (Holst and Magnussen 2018) investigated another angle of the main research question **RQ1**, focusing on the students of the Elective Course of Programming. This resulted in a list of high-level requirements for an educational game that teaches programming, based on a systematic literature review, a workshop held with secondary school students, and the course curriculum of the Elective Course of Programming. This research further develops the high-level requirements list in relation to the teachers' perspective.

The project is initiated by holding several interviews with teachers of the Elective Course of Programming. The interviews are meant to answer research questions **RQ1.1** and **RQ1.2**. Thus they investigate what teachers wish for in a game that teaches programming and how they would include the game in their lectures, as well as what kind of challenges they meet in the course. The interview questions also cover how the teachers currently organize the course and what types of tools and resources they use. The interviews are recorded and transcribed, before they are analyzed qualitatively and quantitatively.

An analysis of the high-level requirements from the specialization project, combined with the findings from the teacher interviews, is performed. This results in a new and updated list of requirements for an educational game targeting the Elective Course of Programming. Based on this, three different game concepts are proposed. In order to select one to proceed with, the authors participate in the *Lær Kidsa Koding* conference to collect opinions through a survey.

A fully functional prototype of the chosen game concept is developed. This is an iterative process, where end-users of the game test it several times during the development process. The game is first pilot-tested by fellow CS students, in order to eliminate evident flaws early on. Then the game is evaluated by experts - one game expert and one teacher expert. This is to get a more thorough examination of the usability of the game. Finally, to answer **RQ1.3**, the main target groups - students and teachers of the Elective Course of Programming - evaluate the game in a classroom setting. Data is collected through observations, interviews, focus groups and questionnaires.

1.5 Results

This master thesis presents several contributions: *interviews with teachers of the ECP* - focusing on their perspectives of the course and their thoughts about a game that will support them during lectures, *a finalized list of high-level requirements for a game that teaches basic programming* - from both student and teacher perspectives, an *analysis of what game elements teachers of the ECP prefer*, and *the educational programming game Code Flip* - designed for the ECP. An overview of the contributions can be seen in Figure 1.2.

From the teacher interviews it was seen that a common challenge faced by teachers of the ECP are that the students do not know what they sign up for when they choose the course, and that it is thus hard to motivate the students. Other challenges faced by several teachers are that there is a big gap in the competence level of their students, which means that it is hard to create content accommodating all students, and that the most eager students end up getting more programming competence than the teacher. Wishes for the game were also gathered from the interviews, and these included the ability to monitor the progress of students, that the game should encourage problem solving, and that it should have clear learning goals.

By combining the findings from the teacher interviews with findings from the specialization project, a complete list of high-level requirements was proposed for the game. Three game concepts were then proposed keeping the new list of high-level requirements in mind. In order to choose one concept, different game elements were extracted and presented in a survey. The survey was held during the *Lær Kidsa Koding*-conference in Trondheim, in order for teachers to prioritize the game elements in relation to motivation and learning outcome of programming. It was from this seen that competition, cooperation in teams, and game analytics were important to include in the game. One of the proposed game concepts emphasized these more than the others, and was as such chosen to proceed with.

As a result, the game Code Flip was developed in an agile way, continuously assessing how it met with the requirements. When the game had been developed into a fully functioning prototype, a pilot test and an expert evaluation were conducted. Both evaluations resulted in new versions of the game, improving the usability, content and concept.

Finally, Code Flip was evaluated in the context it was created for - an ECP class at a lower secondary school. The results suggest that Code Flip manages to increase both motivation and confidence for programming.

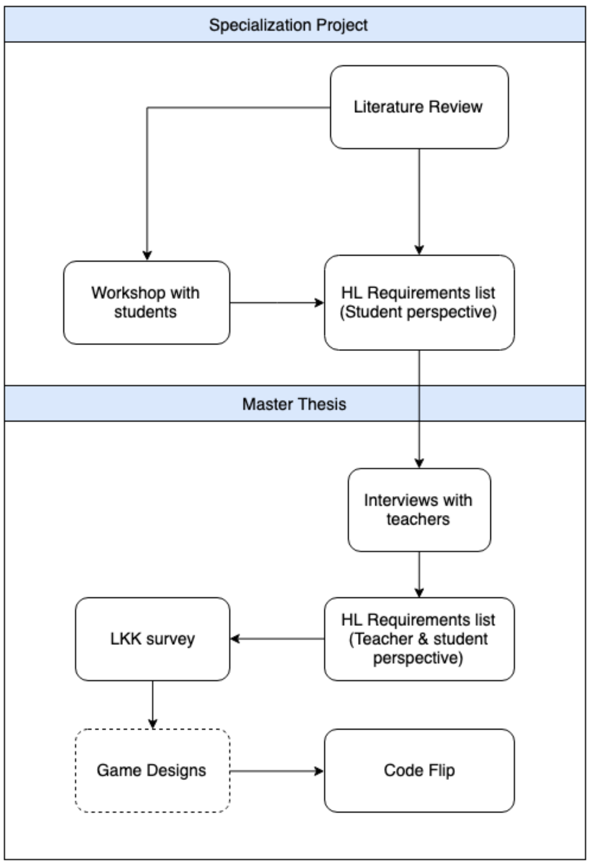


Figure 1.2: Contributions presented in this master thesis. As shown, the work is a continuation of a specialization project.

1.6 Outline of the Report

The master thesis consists of eleven chapters. Chapter 2 gives an elaboration of the problem definition, and presents findings from the specialization project (Holst and Magnussen 2018). The teacher interviews are presented in Chapter 3, before they are discussed in the context of the findings from the specialization project in order to create a more exhaustive list of requirements in Chapter 4. Chapter 5 describes three different game concepts derived from the updated game requirements, and presents the process of selecting one concept for further implementation and evaluation, which includes holding a survey at the LKK-conference. The two next chapters cover the development of the game, through a technical description and a thorough description of how the game turns out to be. Chapters 8, 9, and 10 cover the different evaluations of the game, with discussions of the feedback and the changes made in between. Finally, a conclusion of the study is presented in Chapter 11.

Appendices include: the information mail sent to teachers informing about the interviews (Appendix A), interview guidelines (Appendix B), consent form (Appendix C), LKK conference survey (Appendix D), predefined task sets of Code Flip (Appendix E, student questionnaire for the main evaluation of Code Flip (Appendix F), and answers to the student questionnaire for the main evaluation of Code Flip (Appendix G).

Chapter 2

Problem Elaboration

Programming is known to be a demanding subject to learn, even for adults. Introductory programming courses are often theoretical, and the concepts can be perceived as abstract. As such, several students find the subject boring and lose the motivation to learn it (Jenkins 2002). Nevertheless, it is a discipline which is important to learn for a growing amount of the population, due to the direction in which our society is headed. The Norwegian government has, as many other countries before them, taken action by focusing more on programming in education. In 2016 the *Elective Course of Programming* (ECP) was introduced in Norwegian lower secondary schools (Regjeringen 2016a).

2.1 The Elective Course of Programming

Elective courses were introduced in Norwegian lower secondary schools in 2012 (Dæhlen 2015). The purpose of including them was to increase the motivation of students by offering a more varied and practical content. Studies show that the elective courses succeed in increasing the well-being of students, and that they are popular among both students and teachers (Dæhlen 2015).

An evaluation report of the elective courses, by Dæhlen (2015), finds that students wish to take those that stand out from the ”regular” classroom courses. They are also seen to choose courses that are perceived as easy. This is reflected in the statistics gathered by the Norwegian Directorate for Education and Training (UDIR) as shown in Table 2.1 (Utdanningsdirektoratet 2018). The most popular courses are the ones that are typically not taught in the classroom, and the ECP appears on place 10 out of 15 with a share of 3% of the students.

Each LSS is obligated to offer at least two elective courses, which makes the choice of each student limited (Dæhlen 2015). On average, schools offer five different elective courses, and the students take one per year. Since the courses have upper limits for the amount of participants, students have to apply for their top choices. They may also apply for the same elective course all three years. Similar to the regular courses of LSS, the elective courses are graded and weighted equal to other courses.

The top courses shown in Table 2.1 are the courses that are the most commonly possible to choose. Dæhlen (2015) shows to a survey which states that the schools take student wishes, teacher competence and teacher interest into account when deciding on which ones to offer. In 2016 the ECP was introduced as a pilot project that was supposed to last for three years, but already in 2017 the Norwegian government authorized all schools to offer the course. As a result,

Elective Course (Original Title)	Students	Share
Physical Activity and Health (Fysisk aktivitet og helse)	57 652	32 %
Production for Hall and Stage (Sal og scene)	22 318	12 %
Nature, Environment and Outdoor Life (Natur, miljø og friluftsliv)	19 684	11 %
Design and Redesign (Design og redesign)	16 958	9 %
Volunteer Effort (Innsats for andre)	11 834	7 %
Technology in Practice (Teknologi i praksis)	10 931	6 %
Production of Goods and Services (Produksjon av varer og tjenester)	8 867	5 %
Traffic (Trafikk)	8 674	5 %
Media and Information (Medier og informasjon)	7 818	4 %
Programming (Programmering)	5 176	3 %
Research in Practice (Forskning i praksis)	4 583	3 %
International Cooperation (Internasjonalt samarbeid)	2 096	1 %
Tourism (Reiseliv)	1 266	1 %
Living Cultural Heritage (Levande kulturarv)	1 255	1 %
Democracy in Practice (Demokrati i praksis)	1 127	1 %

Table 2.1: Amount and share of students taking each of the elective courses in the school year 2017-18 (Utdanningsdirektoratet 2018). The ECP had a share of 3% of the Norwegian students.

Learning Goals of the Elective Course of Programming	
Modeling	**Coding**
Explain how computers and programs work, including a selection of widely used programming languages and their area of application	Use several programming languages where at least one has to be text-based
Transform problems into concrete sub-problems, assess which sub-problems that can be solved digitally, and design solutions for these	Use fundamental principles of programming, such as loops, tests, variables, functions and simple user interaction
Document and explain program code by writing appropriate comments and by presenting one's own and others' code	Develop and troubleshoot programs that solve defined problems, including scientific problems and control or simulation of physical objects
	Transfer solutions to new problems by generalizing and adapting existing program code and algorithm

Table 2.2: The learning goals of the ECP, translated from Norwegian.

18.3% of all lower secondary schools in Norway offered the ECP in 2017 (Utdanningsdirektoratet 2018). Due to the novelty of the course, no conclusions can be made about its popularity based on the amount of students who take it.

2.1.1 The Curriculum

The curriculum of the ECP consists of two modules - modeling and coding (Utdanningsdirekoratet 2016a). The modeling module focuses on computational thinking and how to approach problems, while the coding module covers basic programming. Table 2.2 lists the learning goals in detail, translated from Norwegian.

In order to complement the curriculum, UDIR has provided the teachers with a guide on how to approach the learning goals (Utdanningsdirekoratet 2016b). Nevertheless, the guide focuses mostly on the modeling module, or on tasks that already require knowledge about basic programming concepts, like project work. Consequently, the students might not obtain a thorough understanding of the basic programming concepts, especially if the teacher does not have a good understanding of this herself. The programming concepts constitute the most domain specific part of the curriculum, and what is likely to be the most challenging part to teach if the teacher does not have a programming background.

2.1.2 The Teachers of the Elective Course of Programming

As with the elective courses in general, teachers of the ECP are usually persons who already teach other subjects at their school. Dæhlen (2015) finds that most of the teachers feel that they have the competence to teach their elective courses. However, this study was conducted before the introduction of the ECP, and a lot of the other courses have a curriculum consisting of general knowledge, such as ”Physical Activity and Health” and ”Traffic”. All teachers are offered a Massive Open Online Course (MOOC) about teaching programming, but this is not mandatory to take for those who teach the programming course. The teachers also have the option to take other programming courses. NTNU offers several programming courses to teachers, and there is a large selection of courses online, through websites such as Codecademy[1], Code.org[2] and Khan Academy[3]. However, programming is a subject most people are not familiar with, and its prevalence in school is likely to happen faster than the in-service training of teachers.

The curriculum of the ECP is vaguely specified, and the teachers have to create the learning plan on their own (Dæhlen 2015). Variations in both quality and content of the teaching in each school are therefore likely to occur. Hence, the game proposed in this thesis needs to be a tool the teachers feel a need for, and that addresses difficulties teachers have when teaching programming. Their thoughts on what type of game to develop is vital since they decide whether or not to use the game in the course. Accordingly, all of the research questions target the teachers of the ECP.

2.1.3 The Teaching Methods

Case studies following the pilot project of the ECP, performed by Verstad (2017), Corneliussen and Tveranger (2018), and Tveranger (2017), show that several teachers use activities from the teaching guide provided by UDIR. Common for the case studies is that the teachers start by introducing programming concepts in a block-based manner using resources like Code.org[2] and Scratch[4]. Later on, the students learn how the same concepts work in text-based programming languages through the programming language Python[5]. Teachers also include projects where the students can work in groups and use either block-based or textual languages to apply their knowledge through creating their own programs.

According to the case study of Corneliussen and Tveranger (2018), two thirds of the participating students achieved a good understanding of basic programming concepts through block-based programming. However, when later being introduced to text-based programming, only a quarter of the students still felt comfortable with the concepts. This may suggest that it is harder for LSS students to learn textual programming languages than block-based languages, and that the transit from block-based to text-based is a challenge. Seeing that a learning goal of the ECP is to master a textual programming language, this is a challenge that can be addressed by a programming game like the one to be developed in this master project.

2.2 Introducing Programming to Children

It is common to struggle when learning programming for the first time (Robins, J. Rountree, and N. Rountree 2003; Jenkins 2002). It requires a new way of thinking, and it is important to both be skilled at problem solving and seeing the big picture, as well as having an eye for details. Due to the complexity of the discipline, and the fact that programming is a multi-layered skill,

[1]`https://codecademy.com/`
[2]`https://code.org/`
[3]`https://khanacademy.org/`
[4]`https://scratch.mit.edu/`
[5]`https://python.org/`

it is important to focus on understanding one concept at a time (Moser 1997). Hence, pacing is an area of concern since students will not be able to follow the progress of the course without basic knowledge about the different programming concepts.

The domain of tasks students are given in introductory programming courses is also something that can take focus away from the concepts that are important to learn. A common domain is for instance computation of math problems. Byrne and Lyons (2001) states that science and mathematical aptitude often is a pre-requisite for learning programming. This means that students who struggle with mathematics might also struggle with programming, especially when the tasks that are given include solving mathematical problems. Since a lot of the tasks of introductory programming courses include solving math problems, the course is oftentimes considered boring and hard (Jenkins 2002). This affects the reputation, and the students might have poor expectations when entering the course, which again can affect the learning outcome.

Numerous contributions have been made in the field of teaching approaches for introduction of programming concepts. In order to address these contributions, a systematic literature review was conducted as part of the specialization project (Holst and Magnussen 2018). The review focused on ways of teaching programming to secondary school students in a playful manner, and was assessed in the context of three dimensions - teaching approaches, teaching settings and motivational elements. Effective teaching approaches from the literature are two-fold learning, and physical computing, through the use of either block-based or textual programming languages.

2.2.1 Two-Fold Learning

A consensus from the literature is that hands-on experience is an effective way of learning to program for children, especially when complemented by direct instructions from a teacher (Mayer 2013; Kaučič and Asič 2011; Tangney et al. 2010; Majgaard 2015). The reasoning is that learning is complex, and should thus be guided in order to avoid misconceptions or incomplete knowledge. The learning outcome can be further enhanced if there are specific learning goals to be covered by the practical tasks the students solve.

2.2.2 Physical Computing

An approach to teaching programming specifically targeted at children is physical computing, which involves manipulation of physical objects. Examples of physical computing tools are the tiny programmable computer micro:bit[6], and the programmable Lego Mindstorms[7] robots. The teaching approach is based on the concept of *constructionism* as defined by Papert and Harel (1991). Papert states that learning happens effectively when constructing real and visible things by applying pre-existing knowledge. Transferring this to teaching of programming, is meant to decrease its abstraction level through the element of immediate feedback, as the feedback makes it easier to understand the connection between the source code and what it acts out.

In the reviewed literature, physical computing is used through robots or other physical objects such as game boards (Shim, Kwon, and Lee 2017; Roscoe, Fearn, and Posey 2014; Berland et al. 2011). In addition to *constructionism*, physical elements can offer *body syntonicity*. This is another term defined by Papert, and it means that an object can help a programmer think like the object she is programming, by using the object to act out her code in real life.

[6]`https://microbit.org`
[7]`https://mindstorms.lego.com`

2.2.3 Text-Based Languages

Most people associate programming with writing lines of code in an editor, and this is also the essence of the textual teaching approach. One of the learning goals of the ECP is to master a textual programming language. The languages used in introductory programming courses are often the ones that are commonly used in the industry according to Jenkins (2002) and Pears et al. (2007), which as of February 2019 are Java, C, and Python according to the TIOBE Index[8]. Java and C are verbose languages that require focus on syntax, and are therefore hard to learn for beginners (Pears et al. 2007).

Schneider (1978) states that languages used for introductory courses should have a simple syntax. Python is a programming language designed for educational purposes and known for its simplicity (Pears et al. 2007). This programming language has characteristics of a small and clean syntax, immediate feedback and dynamic typing. Python has become more popular in the industry, according to TIOBE, and is also now commonly used in introductory programming courses. Additionally, a case study comparing the use of Python, C and Java in Finnish Secondary School suggests that Python reduces the notational overhead of the students compared to languages like Java and C (Grandell et al. 2006).

2.2.4 Block-Based Languages

An approach to teaching programming which avoids having to remember a specific syntax, is the use of block-based programming languages. Block-based languages let users manipulate graphical elements instead of text according to Kaučič and Asič (2011), avoiding some of the struggles with learning programming. The cognitive load of the programmer is reduced and focus is taken away from the specific syntax of a language. Hence more focus can be directed at understanding the fundamental concepts of programming (Shim, Kwon, and Lee 2017). Popular visual programming languages are Scratch[9] and Blockly[10].

Block-based languages are designed to be intuitive to use and understood immediately, so that children can learn programming concepts from just playing with the language (Shim, Kwon, and Lee 2017). Studies of the block-based languages Scratch and A-bricks show that children can manage to learn the most apparent programming concepts this way (Shim, Kwon, and Lee 2017; Maloney et al. 2008). However, less apparent concepts, such as functions and sequences, are not as easily picked up. Employing a two-fold teaching approach, can help omit this challenge.

2.3 Motivational Elements in Educational Games

From reviewing the literature of introducing programming to children, it is evident that motivation is essential. In fact, motivation may be seen as more important than the actual learning outcome when teaching programming to children (Spieler et al. 2016; Berland et al. 2011; Shim, Kwon, and Lee 2017). How to learn best is individual, but being motivated for a task helps with concentration and involvement, and is hence likely to increase learning. Different factors can be employed as means for motivation, and those below are through the literature seen to be

[8]`https://tiobe.com/tiobe-index/`
[9]`https://scratch.mit.edu/`
[10]`https://developers.google.com/blockly/`

effective in educational games for teaching programming. Seeing that the goal of this thesis is to develop such an educational game, these motivational elements are highly relevant.

Competitions

Competitions can help engage children to learn a topic in order to perform in a competition, and can make them keep thinking about a topic in the aftermath of the competition (Tomcsányiová 2013; Berland et al. 2011). If the conditions they compete under vary, more students can become engaged as it becomes less probable that the same students win every time, according to Shim, Kwon, and Lee (2017).

Cooperation

Cooperation is another motivational element seen in several publications, either through group work, pair programming or by consulting each other for help. Cooperative game activities are seen to help students focus and immerse themselves in learning programming, even if they do not find the discipline interesting (Shim, Kwon, and Lee 2017).

Playfulness

All of the contributions in the literature review employ playfulness to motivate learning, through the use of game-play and robots. The playful setting of the tasks can help players understand problems better. Jovanov et al. (2016) reasons that children like games and learn new things while they do what they enjoy. Berland et al. (2011) and Shim, Kwon, and Lee (2017) combine familiar contexts with the yet unfamiliar and more abstract concept of programming, and succeed in improving the programming attitudes of the participating students, as well as increasing the learning outcome.

2.4 Popular Game Elements among LSS Students

From the literature review, the authors were not given a deep insight into specific programming games to be used as part of a course curriculum, nor did they discover what types of games young teenagers enjoy playing. In order to further investigate these matters, a workshop was performed. The purpose of the workshop was to further examine the existing games as part of the state of the art, and to come up with ideas for an educational game that teaches programming.

In the first half of the workshop the students played two different programming games, before analyzing and discussing the games. The games CodeCombat[11] and Grasshopper[12] were selected by comparing and evaluating popular existing games that aim at teaching programming to children. From the game-playing sessions, the authors got insights into how much the students learned from the games, what the students liked or disliked with them, and how the games affected their motivation for programming. Followed by the game-playing session was a co-design session where the students came up with ideas for a programming game. This session gave more insight into what type of game the students want and different game elements they enjoy.

[11]`https://codecombat.com/`
[12]`https://grasshopper.codes/`

From analyzing the results of the workshop as a whole, a list of popular game elements was proposed. The list is shown in Table 2.3, and originates from the specialization project of the authors (Holst and Magnussen 2018).

Game Element	Description
Meaningful Story	The game should have a meaningful story that the player can follow, keeping her engaged.
Dependent Tasks	The game's tasks should be dependent on each other, so that each task feels meaningful to solve.
Increasing Difficulty	The game's tasks should increase in difficulty throughout the game and in a decent pace to keep the player interested.
Immediate Feedback	The game should let the player be able to *see* the outcome of her actions.
Specific Tasks	The game should have specific task descriptions so that the player can focus on problem-solving and learning instead of being creative.
Tasks that Motivate	The game's tasks should not be about learning programming. They should instead contribute to the story.
Appropriate Amount of Text	The game should have an appropriate amount of text to increase comprehensibility and avoid making the game boring.
Optional Difficulty	The game should let the player be able to choose freely between the levels of the game.
Multiplayer	The game should let the player be able to play with or against friends to increase motivation.

Table 2.3: Popular game elements from the workshops of the specialization project (Holst and Magnussen 2018).

2.5 High-level Requirements for a Game that Teaches Programming to LSS Students

The main contribution of the specialization project was a list of high-level requirements for an educational game that teaches programming to LSS students. The requirements are based on the motivational elements and teaching methods from the systematic literature review, popular game elements found in the workshop with LSS students, and the course curriculum of the ECP. The list is shown in Table 2.4. The requirement codes are given the prefix SR, short for Student Requirement, since they represent the student perspective. These high-level requirements are further discussed in Chapter 4, in relation with the interests of teachers, presented in Chapter 3.

Requirement	**Description**
SR01	The game should offer immediate feedback to the player.
SR02	The game should use a textual programming language, but must offer elements that eliminate the need for remembering a specific syntax.
SR03	The game should use a programming language that is suitable for beginners, by having a simple and clean syntax.
SR04	The game should encourage two-fold learning in the teaching setting it is used in.
SR05	The game should be separated into levels, where each level introduces a programming concept.
SR06	The game should have tasks that advance in difficulty for each level, in order to motivate learning and keep the player engaged.
SR07	The game should include the motivational elements of competition and/or cooperation.
SR08	The game's tasks should have a motivational purpose, and not just focus on learning programming.
SR09	The game should be complemented by a physical object that offers body syntonicity.
SR10	The game should be integrated into the Elective Course of Programming.

Table 2.4: High-level requirements: The student perspective.

Chapter 3

The Teachers of the Elective Course of Programming

Before proposing an educational game to be used in the ECP, it is important to address the teachers of the course. Since the course curriculum is vaguely specified, and the teachers need to create a learning plan on their own (Dæhlen 2015), the game should be something that they want to use, and that can assist them as they teach the course. The fact that the course is quite new and the curriculum is vaguely specified, may also mean that there is a greater possibility that teachers are open to introducing a new tool to the course.

Several interviews are conducted in order to address the teachers and answer **RQ1.1** and **RQ1.2**. The results are used to form a foundation for the game designed in this thesis, together with the high-level requirements from the specialization project. Before presenting the results, this chapter elaborates about the purpose of performing the interviews, the participants and how they were chosen, as well as the procedure.

3.1 Purpose

The purpose of the interviews was to find out what kind of background the teachers of the Elective Course of Programming have, how they organize the course, what types of tools and resources they use, and the challenges they meet when teaching the course. As the main goal of this master thesis is to develop a game that teaches programming to lower secondary school students, the teachers were also asked about educational programming games, their opinion of them, and what they would want in such a game.

3.2 Participants

The target group of the interviews was teachers of the ECP. The Norwegian Government has provided a list of all of the schools that started offering the course in 2016 (Regjeringen 2016b), and the authors used this to know which schools to contact. Several other schools are likely offering the course as of 2019, however the authors preferred speaking with those who had already taught the course for a while, and thus were more likely to have experiences and opinions to share.

In order to recruit programming teachers, an email was sent out to the administration of each school. The email contained information about the master thesis, and a request to be set in contact with the programming teacher(s) at the school if they were interested in contributing. The email content is shown in Appendix A. In total 139 schools were contacted, and replies were received from 23 of the schools. 13 of these were able to perform the interview, all of them teachers or assistants of the ECP from different schools all over Norway.

3.3 Process

The interviews were planned with inspiration from Oates (2005). They were initiated by a short introduction of what the authors wanted to find out, and what the collected data was to be used for. Then the questions were asked, according to a list of predefined interview guidelines. The interviews were semi-structured, so the guidelines were not strictly followed. Questions were added if an interviewee described something the authors found to be of particular interest. It was however made sure that the most important questions were answered by all interviewees.

As the participants were spread all over Norway, most of the interviews were conducted over the phone. Some were held in person, as one of the authors was temporarily situated near three of the schools. The teachers could freely choose a time slot for their interview. Data was gathered through recordings, using a digital voice recorder of the brand Olympus provided by the Department of Computer Science at NTNU. After conducting the interviews, the recordings were transcribed and analyzed both qualitatively and quantitatively.

The data collection was approved by the NSD[1] beforehand. Before participating in the interviews, each teacher signed a consent form with information about what their participation would entail, how data would be gathered, and what it would be used for. The consent form can be found in Appendix C. One teacher requested the interview guide in advance, and this was thus provided. All interviewees were reminded that the participation was voluntary in the beginning of the interview. The authors made sure to transcribe the recordings exactly as they were spoken in order to ensure integrity. No personal data is revealed in the transcriptions, and all recordings will be deleted once the censorship of the thesis falls.

3.3.1 Interview Guidelines

In advance of the interviews, guidelines consisting of fifteen questions were created, and these were mainly divided into three different topics. The first topic was about the teacher. This included information about their programming background, and why they wanted to teach the course. The second topic was about the course - organization of the course, tools and resources that are used, and the challenges encountered when teaching the course. The last topic was about educational programming games and what the teachers would want in such a game.

The list below shows the questions of the interviews. They are translated from Norwegian. The original interview guidelines can be seen in Appendix B.

- Why did you choose to teach the Elective Course of Programming?
- How have you acquired programming competence?
- What other subjects do you teach?
- What do you find most challenging when teaching programming to children?

[1]NSD - Norwegian Center for Research Data

  - How do you solve this?

- What kind of tools and resources do you use in your lectures?
- Which programming languages do you use?
- How do you organize the course?
- About how many of your students have taken the course before?
- Do you use any games in your lectures?
- Have you heard of any games that teach programming?
- What is your attitude towards using educational games in lectures?
- How would you include a programming game in lectures? (As introduction to programming concepts, repetition assignments, group activities, competition?)
- Do you use any group assignments or competitions in your lectures?
- How do you evaluate the performance of your students?

3.4 Results

Thirteen interviews were recorded, transcribed and analyzed. As the interviews were semi-structured, the main topics were emphasized to differing extents following the course of the conversation. The interview results were transcribed and analyzed by the authors in parallel. The analysis was performed after all transcriptions were done. Each author analyzed the interviews the other had transcribed, so that both authors were well familiar with all of the interview content, and could easily proofread each others analyses.

To ensure consistency, an analysis approach was agreed on beforehand. The interview results were coded in a combination of deductive and inductive manners, depending on the questions. For the topics where the authors had already been given some insight through the literature, a deductive manner was chosen, such as programming languages and resources used in the course. For the questions related to other topics, like the challenges faced by the teachers, and their competence levels, an inductive manner was employed.

Each transcription was first coded according to the categories, before the coded results were copied into a common document for an overview of each category. This document was used for examining each category and looking for trends between them. Each copied part was given a code to mark the location in the transcription it was taken from, and which teacher it belonged to. The common document was created by the authors together, in order to further ensure consistency of the analysis.

Each of the following sections present a result category. The data of each category is described through text, charts, as well as quotes translated from Norwegian.

3.4.1 Reasons for Teaching the Course

The teachers specified similar reasons for choosing to teach the Elective Course of Programming. Eleven of the thirteen teachers said they were asked by their school if they could teach it, as the school administration wanted to sign up for being a part of the pilot project in 2016. The teachers were targeted due to having shown an interest, or because they already had some experience with programming. One was asked simply due to having some extra hours not used for teaching. Common for them all was that they found programming interesting, and thought it was about time that it was integrated as a course in LSS.

Some of the teachers seemed to have been a driving force for introducing programming to their school. One had participated in developing the learning goals of the course. Others stated that:

> *"I think programming is fun, and I have tried to introduce it here and there when teaching Arts and Crafts. Some students have shown a particular interest for programming, and those I have enlisted for programming competitions and made sure they could use some of their school time for preparations. I have been running it as far as it has been possible for all years, and continued in the elective course Technology in Practice, and now I can do it through the Elective Course of Programming."* - Teacher A

> *"I have programming competence, so I was asked and said yes. I think programming is fun, and I think I would have tried to convince the school to offer it if they did not already have it when I started here."* - Teacher B

3.4.2 Competence Level

As mentioned, all of the interviewees already had an interest for programming, and it follows that most also had competence to some degree before taking on the course. Two out of thirteen stated to have no prior programming knowledge. These prepared for the course by taking some of the Massive Open Online Courses (MOOC) that are offered to all teachers, as mentioned in Section 2.1. Out of the remaining eleven teachers, seven had formal programming competence, either through having taken some courses at a university, or having a full master's degree. The distribution of competence level the teachers had before teaching the course can be seen in Figure 3.1.

3.4.3 Challenges Faced

The challenges the teachers said they faced when teaching the ECP can be categorized into five main areas: students are not prepared, lack of patience, knowledge transfer, level of programming skills, and students get more programming competence than the teacher.

Students are not Prepared

One challenge of the ECP is that many students do not know what they have signed up for. A lot of them think that the course is about playing or creating complex games like Fortnite or Grand Theft Auto. When one of the teachers asked a student why she chose programming, the answer was *"Because my dad told me I should take programming"*. This leads to a lot of students being

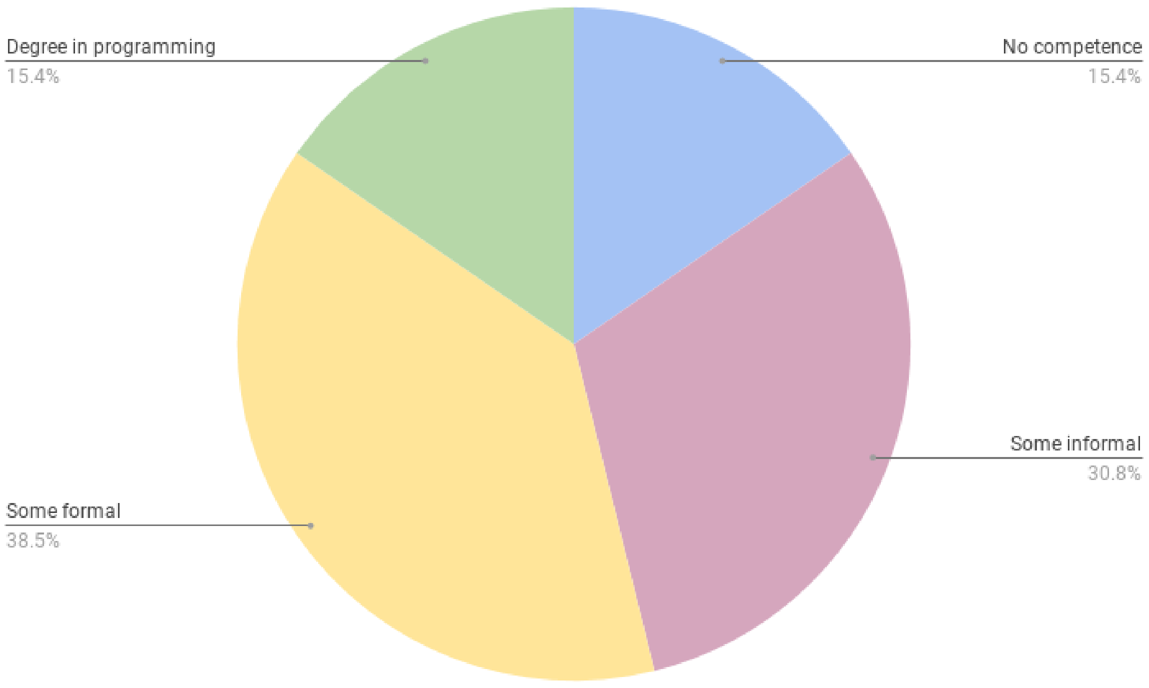


Figure 3.1: The competence level of the teachers prior to teaching the course.

surprised by the complexity of the course, and they might regret choosing it either because they do not find it interesting, or because they want to take an elective course that is easier.

Some of the schools address this challenge by letting students change elective course during the school year. Other schools try to keep them as motivated as they can, usually by letting them stay at a very basic level, and giving them simple and fun assignments. Teacher C said that:

> *”I try to give them fun assignments. However, there’s not much to do about students not being interested in programming. It is what it is, and you can’t force interest. The students who struggle the most are usually engaged by Scratch, so these are allowed to work with that throughout the year.”*

Lack of Patience

Another challenge of the ECP is that many students lack patience. They are not trained as problem-solvers, which makes it hard to approach the tasks they are given, and to troubleshoot their code when it does not work as intended. They are as such likely to give up when they face challenges. A lot of students ask for help as soon as they get stuck, or when they lose focus. Teachers solve this by providing hints about the tasks, making them turn to each other for help before asking the teacher, or encouraging them to google their problems.

Knowledge Transfer

The teachers claimed that students are great at copying or following recipes. If a teacher demonstrates an example, the students can easily do the same afterwards. The problem is that most of them only manage to do exactly the same. They do not manage to use the same code concepts for something different. The schools try to solve this in various ways. One teacher makes a program skeleton, where the students get enough information to understand the code and then have to further develop the skeleton. Others introduce one task, before assigning the students with new tasks with only a few changes.

Varying Level of Programming Skills

The programming skills of students vary. Some of them already have some knowledge about programming before they start 8th grade, while others have never heard of programming. Through the years of LSS there is also a big difference in the amount of hours each student spends on programming. There is no homework in the course, but some of the most eager students spend time at home at their own initiative. This causes an even bigger gap between the weakest and strongest students. The course is taught 1.5 hours per week, while one of the teachers states that a few students might use up to 10 hours on programming per week.

As a result, the teachers have to create several course plans for the different levels, so that the course does not get too hard for the weakest students, while not too easy and boring for the strongest students. Some of the schools let the weaker students keep programming with Scratch on a basic level, while others instead let the strongest students ”run ahead”, with more flexibility in what they want to do.

Students Get More Programming Competence Than the Teacher

A challenge that may occur if the teachers have limited competence in programming, is that students gain more competence than the teacher. As mentioned, some students are likely to have a greater interest for programming and therefore spend more time on it. This makes it hard for the teacher to create a suitable course plan. One of the teachers solved this by letting the students do what they want to do, and if they need help the teacher googles it. A problem from the teacher perspective is that they might not feel academically strong enough to teach the more skilled students, which was also stated by one of the teachers.

3.4.4 Tools and Resources

The tools and resources used as part of the ECP at each school varies. This is most likely a result of teachers creating the lesson plans on their own. As some of the teachers stated during the interviews, they often listen to the students’ interests. If some students want to program in Java or program a micro:bit, they are welcome to do that. They also justified this by the fact that some students are on a different level than others.

Resources

The most common resources used in the ECP are the websites ”Lær kidsa koding”[2] and Code.org[3]. These websites consist of curriculum plans for teachers of programming courses, and a lot of courses and assignments in different programming languages. The teachers mostly use them to give assignments for Scratch, Python, Lego Mindstorms, or micro:bit.

[2]`https://kidsakoder.no/`
[3]`https://code.org/`

Block-based Programming Tools

For block-based programming the teachers use a wide range of tools, where the most common is Scratch. The physical tools that are most widely used to teach block-based programming in the course are micro:bit, Lego Mindstorms, and Arduino.

Text-based Programming Tools

The most common tool that is used to teach textual programming languages is an editor, either locally or through the browser[4][5]. Other tools that are used are CodeCombat and micro:bit. micro:bit can be programmed by both a block-based programming language and the textual programming language Python.

Two of the teachers also stated that they used the website `http://vesteras.as/processing/` which is an online editor that teaches the textual programming language Processing. The website includes both assignments, theory, and since it is online it is easily accessible for both teachers and students.

3.4.5 Programming Languages

Figure 3.2 shows the distribution of the programming languages most frequently used in the ECP. As stated in Chapter 2, Python is a programming language that is recommended and designed for beginners, and hence this is reflected in the ECP. As can be seen in the figure, Python is used by 92.31% of the teachers. Another programming language that is almost as popular as Python, is Scratch. Programming languages such as HTML & CSS, and JavaScript are mostly used by the students who take the course for the second or third time.

The less frequent programming languages used in the ECP, which are not shown in the figure, are C#, Java, Swift, Beetle Blocks, Objective C, Arduino block programming, and PHP. There are different reasons why teachers have chosen to include other languages than the ones recommended by UDIR. Some teachers use a programming game called Swift Playgrounds, which is built upon the programming language Swift, simply because they think the game is good. Teachers who have experience with programming prefer using languages they know well, which sometimes are others than Python and Scratch. There are also teachers who let students decide what they want to do, as long as the motivation and interest is present.

3.4.6 Course Organization

About one third of the teachers introduce the students to programming through simple activities without a computer to make them understand the algorithmic way of thinking. Examples of such activities are making a person move through a maze by giving articulate commands, or that the teacher follows instructions from the class for spreading jam on a piece of bread and eating it. The most common way of introducing programming is through Scratch. Several teachers claimed that Scratch is useful both for introducing the basic concepts, and for providing immediate and understandable feedback. They also stated that it motivates, as the outcome from working with Scratch is a game developed by the students themselves. This can create a feeling

[4]`https://repl.it/`
[5]`https://trinket.io/`

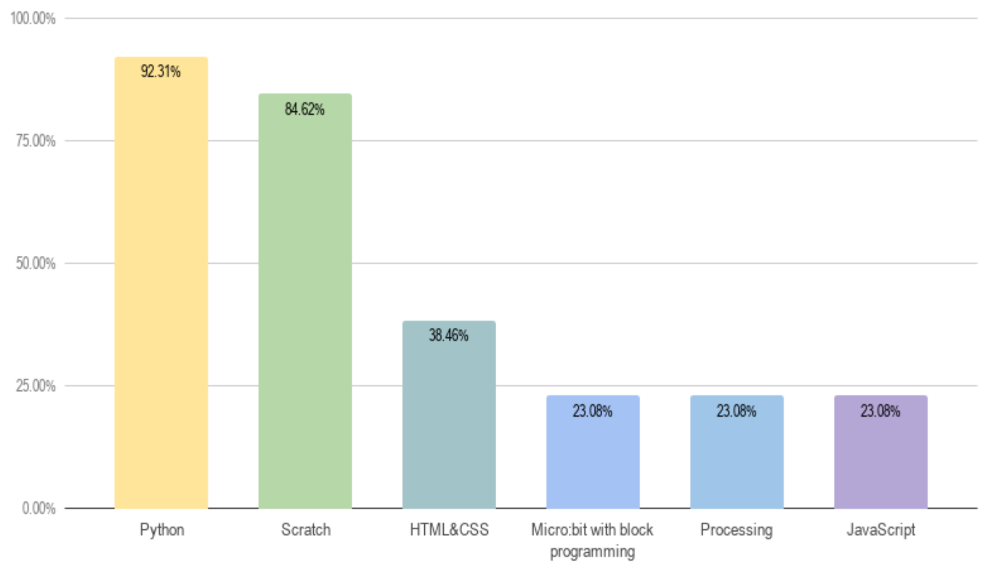


Figure 3.2: Distribution of programming languages used in the ECP.

of confidence and ownership, and in one case students were allowed to spend the rest of the class playing their games if they had time to spare.

After having taught algorithmic thinking and basic concepts through Scratch, the teachers commonly proceed with some sort of physical computing. They claimed that the students find it motivating and educational to see the feedback from their code in physical form. Here the teachers can either choose to make the students program block-based or textual, depending on the physical tool they use.

As the mastering of a textual programming language is a teaching goal of the ECP, all introduce a textual language in some way, but the extent to which they use it varies a lot. Some teachers spend a lot of time teaching textual programming, while others only show how textual languages work, and let them play a game like CodeCombat or Swift Playgrounds to explore them. In most cases, the approach depends on how much the students struggle with learning programming. The textual languages are usually introduced after the students have become familiar with the basic programming concepts.

The biggest difference between how teachers organize the course, is how they introduce the learning material. Some go through material in plenary before having the students try themselves. Others let them explore new things themselves, and summarize at the end of a class what they should have learned. Some assign the students with small tasks, so that they can solve many tasks during one class, and each task is related to a concept they are covering. Others work with bigger projects where they have several months to create something, and where guidance from the teacher should be sufficient for learning what is needed.

With elective courses of Norwegian LSS in general, students are allowed to choose the same courses several times. This goes for the course of programming as well. Seven of the interviewees said their school has one programming class per grade, while eight of them said they have one class for all students who choose the programming course. Either way, there may be students who take the course for the first, second or third time in the same class. As mentioned, this challenge can be solved by giving students freedom to choose what they want to work with. The teachers may give an assignment which is possible to solve with both blocks and text, or they may hold projects where the students can choose themselves what they want to make. Some of the schools have a set content for those who take the course for the first time, while those who have already taken it get projects to work with. The experienced students may also be used as teaching assistants.

3.4.7 Use of Games in Lectures

Six of thirteen teachers claimed to have used games in their lectures. The mentioned games are CodeCombat, Swift Playgrounds, Minecraft Education and Run Marco. The teachers' perception of the games are diverse. Some teachers think the games are good entrance tools for programming, as they are fun and simple. Others think that the games fail to teach the students the specific programming concepts, which the authors also saw signs of when they had students test CodeCombat during their specialization project (Holst and Magnussen 2018). Another problem with the games is that most of them require a license, which is quite expensive and not all of the schools are willing or able to pay for this.

Nevertheless, all of the teachers showed positive attitudes toward games. Teacher D said that *"The essence of elective courses are that they should be fun, motivating and practical"*. All the teachers claimed that a game might be a good motivational tool for the students, and that more competition elements could make the course more fun. The teachers who had already tried existing programming games, specified different ways of solving the problems that came with the games. These desires are taken into account in Section 3.4.9.

3.4.8 Evaluation of Students

Several teachers stated that the evaluation of students in the ECP is hard. A reason is that they find it difficult to evaluate a course where some students eventually have more knowledge than the teacher. The students also manage to easily do what the teacher does, but struggle when they need to be self-driven, and how does one evaluate the ability to be self-driven? Some of the teachers stated that they had not found a good way to evaluate their students. They sometimes have smaller quizzes, use observations, and evaluate bigger projects which mostly imply looking at the process and explanations, and not as much on the actual result.

3.4.9 Teachers' Game Desire

One of the last questions of the interview was about how the teachers would integrate a game into their lectures. Their thoughts here were mostly based on their challenges in the course, which makes sense as they want something that is both fun and useful for them. The teachers had a lot of different opinions on this particular part. In order to give an overview of them, they are presented in Table 3.1. They are given a code with prefix TR, Teacher Requirement, since they represent the stakeholder group of teachers.

Requirement	Desire	Description
TR01	Monitor	The game should have a teacher mode where they can monitor the students' progress.
TR02	Clear learning goals	The game should give the students a clear perception of what they are about to learn.
TR03	Encourage problem-solving	The game should encourage problem-solving, and not accept hard-coded solutions.
TR04	Web-based	The game should be web-based so that it is easily accessible.
TR05	Teacher guidance	The game should have a teacher guide, because it can be hard and scary to use games in lectures.
TR06	Introduction to programming	The game should teach introductory programming, as games can make theory more fun, and give a visual example on how concepts such as loops work.
TR07	Text-based	The game should use a text-based programming language, as there already exist a lot of tools teaching block-based languages.
TR08	Evaluation tool	The game should be an evaluation tool. For instance by having assignments that are used for certification in different programming concepts.
TR09	Avoid language barrier	The game should have instructions in Norwegian, as poor English skills should not affect the possibility and motivation for learning programming.
TR10	Physical elements	The game should use physical elements in order to motivate students.
TR11	Immediate feedback	The game should offer immediate feedback to increase learning outcome.
TR12	No student login	The game should not require any login for students. This is due to GDPR and the privacy of students.

Table 3.1: High-level requirements: The teacher perspective.

Chapter 4

Discussion of High-Level Requirements for the Programming Game

The specialization project performed prior to this thesis contributed with a list of high-level requirements for a game for teaching programming to children, as described in Chapter 2.5. These requirements were based on the literature and the opinions of lower-secondary school students. In the previous chapter additional requirements were gathered from interviewing teachers of the Elective Course of Programming. In this Chapter, all of the requirements are discussed in detail. The requirements are classified according to priority and difficulty levels.

4.1 The Requirements

The requirements from both the student and teacher perspective, listed respectively in Tables 2.4 and 3.1, are discussed below. Before going ahead and designing the game, the authors selected and prioritized the requirements, as having eighteen requirements might restrict the creativity of the ideas. The ones with prefix SR represent the student stakeholder, while those with prefix TR represent the teacher stakeholder. As the requirements arise from various sources, there are necessarily some conflicts between them, while some overlap. The requirements that overlap are discussed as one.

The requirements are given a *Low*, *Medium* or *High* rate of priority and difficulty by the authors. Difficulty represents the complexity of integrating or including the requirement in a game, and are qualified guesses based on the experience the authors have with software engineering.

The game should be integrated into the Elective Course of Programming (SR10)

Priority: *High*
Difficulty: *Medium*

The long-term goal of this research is to create a game in the context of the ECP, making this a highly prioritized requirement. There are several possibilities for using the game in a course - as homework, complementing lessons, substituting lessons. Students of the ECP are not expected to do any homework. As such, the game should be designed for use in the lectures of the ECP.

The game should complement lectures instead of substituting them, in order to accommodate teachers regardless of their background. This way, the game can be used by those who have a formal programming background, while letting them control their lectures as they like. Teachers without a programming background may find it better if the game substitutes lectures, but by providing an extensive teacher guide with the game, the threshold for using it should be low, and the teachers can continue to use the game in lectures as they obtain more programming knowledge.

The game should be web-based so that it is easily accessible (TR04)

Priority: *High*
Difficulty: *Low*

At the beginning of the specialization project the authors decided that the game should be web-based. There were mainly two reasons for setting this requirement from the beginning. Firstly, a web-based game is easily accessible for all schools. In Norway it is a requirement that Norwegian students have access to computers at school (Utdanningsdirekoratet 2014). Secondly, the authors may be able to allocate more time for user-testing by setting this constraint, as they have experience with web development.

This requirement is further emphasized in the teacher interviews, as many teachers have restrictions for what they are allowed to download on the students' computers. A few of the teachers claim to be bound to exclusively use computers in their lectures, as their school lack sufficient funding to purchase other tools, like micro:bit, or program licenses. As such, this requirement is given a high priority. Since the authors already have some familiarity with web-development, contrary to other game development, the difficulty level is considered low.

The game's tasks should have a motivational purpose, and not just focus on learning programming (SR08)

Priority: *Low*
Difficulty: *High*

This requirement originates from the co-design workshop held with secondary school students where the goal was to capture what the students enjoy in a game, and how they would prefer to learn through a game. They explained that in order to be interested in playing an educational game, there needs to be a purpose that goes beyond learning. It is not enough to try to disguise educational tasks in a few elements of gamification. The tasks of the game must make the player excited about solving them. Game elements seen to be popular among students are having a meaningful story, or being able to play with or compete against classmates.

Since the game should not be used at home, but in the classroom, the sources of motivation do not need to be the game-play in itself, but can come from elements like competition or cooperation in the classroom. Competition and cooperation are covered by another requirement, so this one may be redundant. Besides, there should be a certain amount of learning outcome from an educational game, as the teachers are to evaluate their students. As seen from the literature, it is hard to measure a clear learning outcome from playing some programming games. If there is a lot of focus on the game story, little focus is likely left for learning programming. This requirement is consequently considered to have low priority.

The game should include the motivational elements of competition and/or cooperation (SR07)

Priority: *High*
Difficulty: *Medium*

From the literature it is seen that competition and cooperation are effective ways of motivating through games (Shim, Kwon, and Lee 2017; Berland et al. 2011). Several teachers claim that they make use of competitions in their classes, and that the students feel motivated by it. A learning goal of the ECP is to be able to explain program code to classmates. This can be obtained by the use of competition and cooperation, where students are encouraged to share their knowledge and exchange code, in order to perform in a game. It may even be possible to combine the two, by cooperating in order to compete against other teams.

One of the main reasons for using a game to teach programming, is to engage the students who are not already eager to learn how to program. Consequently, it is crucial that the game focuses on motivational elements that work for the majority of students. A concern with competitions is that those who do not perform well may end up giving up, and that those who perform well are those who are already engaged in programming. If the game should include competition-elements, the way of including them must be carefully chosen, in a way where students do not get discouraged if they do not perform well. Bearing this in mind, the priority level of this requirement is high, while it is considered to be of medium difficulty.

The game should use physical elements (SR09/TR10)

Priority: *Low*
Difficulty: *High*

Physical elements can also be used as a means for motivation, and it can help students obtain a better understanding of programming, as described in Chapter 2. Two of the learning goals of the ECP include: explain how computers and programs work, and control or simulate physical objects. Physical objects can be helpful in offering body syntonicity to help decrease the abstraction level of programming, in addition to offering immediate feedback.

From the interviews it is seen that physical elements are already heavily used in the ECP, through tools like Lego Mindstorms and micro:bit. The teachers do not seem to have trouble employing these tools, and they claim that the use is motivating and successful. It can hence be argued that these learning goals are already well covered, while other learning goals such as mastering a textual programming language, or being able to explain a programming concept, may need more coverage and require more motivational focus. This requirement is therefore given a low priority. The difficulty level is high, seeing that the authors do not have any experience with working with hardware, and requiring purchase of hardware can make the game hard to promote.

The game should encourage two-fold learning in the teaching setting it is used in (SR04)

Priority: *Medium*
Difficulty: *Medium*

An effective teaching approach seen in the reviewed literature, is the two-fold approach. There are several programming games that employ two-fold learning, by introducing the concepts needed to solve the following tasks. However, in the workshop conducted in the specialization project, the authors observed that students skipped past instructions, because they found the instructions boring to read and were eager to proceed in the games.

A solution to this is that the game can be used as a tool to apply knowledge that is already taught by the teacher, i.e. that the teacher introduces a concept before it is practiced in the game. In that case, it must be clear that the game is to be used as the second part of a two-fold learning approach, for instance by providing a teacher guide with the game. A constraint is then that the game must be used in a classroom setting, as it is assumed that the players already have some knowledge about the programming concepts of the game, and they should not be required to acquire this knowledge on their own.

The game should use a textual programming language (SR02/TR07)

Priority: *High*
Difficulty: *Medium*

A requirement that comes from both the specialization project and the interviews is that the game should use a textual programming language. There already exist a lot of well functioning tools for teaching block-based programming, for instance Scratch. As such, it does not seem to be need for yet another tool that teaches programming in this way. The literature suggests that students find the transition from block-based to text-based programming hard as they struggle with remembering specific syntax. This is a problem that the authors might be able to address with their game.

The difficulty of this requirement is set to medium as it requires the game to include some sort of editor, so that students can write code inside the game. The editor then needs to be able to communicate with the game itself.

The game should use a programming language that is suitable for beginners, by having a simple and clean syntax (SR03)

Priority: *High*
Difficulty: *High*

This requirement is dependent on the previous requirement, as the selected programming language has to be a textual language. A struggle with introducing programming with textual languages is that the syntax can confuse learners, since it can be hard to separate between language-specific elements and elements connected to programming in general. Several languages heavily used in the industry have a verbose syntax that is not intuitive to use, and thus the choice of introduction language is crucial. It is seen that languages suitable for beginners are those with a simple and clean syntax, as described in Section 2.2.3. Python is an example of such a language, as it is designed with beginners in mind. This language is used in introductory programming courses at NTNU, and several of the teachers of the ECP claim to have used it or to be familiar with it.

By using Python as the programming language, there are some difficulties that follow as the game is going to be web-based, and as such probably not written in Python. In order to run the written code directly in the web-browser and get immediate feedback, it has to be sent to an external server for execution. The server must then send the output back to the web page. Hence, the difficulty is set to high.

The game should offer immediate feedback to the player (SR01/TR11)

Priority: *High*
Difficulty: *Medium*

As described in Chapter 2, immediate feedback is an effective element for teaching programming to children. It has been shown to increase understanding of programming, and it can help with troubleshooting code, since the consequences of code changes become apparent. Hence, it targets learning goals of the ECP: *explain how programs work and troubleshoot programs.*

The teacher-interviewees claim that one of the reasons for choosing to use Scratch and physical elements like micro:bit and Arduino in their lectures, is the element of immediate feedback. As mentioned, physical elements may not need additional focus in the ECP. The secondary school students participating in the workshop stated that they liked getting immediate feedback when performing actions in the games of CodeCombat and Grasshopper. As such, it is a priority to include immediate feedback in the game. By including an editor in the game, there will already be some sort of output when the students run their code. The authors should customize this output to make it more understandable for LSS students. Hence the difficulty is set to medium.

The game should teach introductory programming (TR06)

Priority: *High*
Difficulty: *Low*

Some of the interviewed teachers want the game to be used for introduction to programming concepts. The reason is that introduction lectures are often theoretical, which several students find boring. Students also tend to perceive the concepts as abstract when learning them for the first time. As stated by one of the teachers: *"Games can make the theory more fun, and they get a visual example on how concepts, such as loops, work"*. When being introduced to something for the first time, it may be hard to grasp or remember. Repetition can in such cases be helpful, and a programming game for applying theory can be used for this. If the students know that the theory is to be applied in a game later on, they may even be encouraged to pay good attention during the theory lesson.

The game should have a teacher guide (TR05)

Priority: *High*
Difficulty: *Medium*

During the interviews it was stated that it can be hard and scary to use games in lectures. The reason was that they were not that familiar with games in general, and therefore felt like they had to embark on a new territory. By creating a teacher guide for the game it might be more likely that teachers who are not into games wants to try it out. The guide can help teachers when integrating the game into their lectures, and possibly reduce the time required to learn the game. The guide can also list the learning goals of the ECP that are covered by the game. From the authors' perspective there are both up- and downsides in creating a teacher guidance. It will most likely make it easier to present the game as an education tool to be used in class, but it will also be time consuming to create such a guide.

The game should have a teacher mode where they can monitor the progress of their students (TR01)

Priority: *High*
Difficulty: *High*

This is a requirement that a lot of the teachers find important with a programming game, as it makes it easier to observe the students' results without being intrusive. Additionally, by implementing a teacher mode in the game it will make it more adaptable for lectures. As mentioned earlier, a desire of some teachers was an evaluation tool. A teacher mode cannot be directly translated into an evaluation tool, but by showing analytics on how many tasks the students have managed to solve, the difficulty level, and how they have solved them, the teachers are given an indicator for what their students master well and not so well. This can be used for evaluation, but maybe more importantly for following up on students, and offering support on the topics they struggle with.

A teacher mode can offer analytics during game-play, after game-play or both. During game-play a teacher can offer help to a student who has been struggling with a task for a long time. Analytics after game-play is more helpful for evaluations and can show trends and progress, as the data can be compared between playing sessions.

The game should offer students a clear perception of what they are about to learn (TR02)

Priority: *Medium*
Difficulty: *High*

From the workshop conducted during the specialization project (Holst and Magnussen 2018) it was found that after the students had played the programming games CodeCombat and Grasshopper, they did not manage to answer questions about the programming concepts introduced in the games. As such it may seem that the games struggled with giving a clear perception of what they were teaching the students. This is also recognised by some of the teachers who have used CodeCombat in their classes. Therefore, in order to use the game to teach the programming concepts it must give a clear perception of which concept the student is about to learn.

This requirement is set to medium on difficulty because it requires the authors to come up with a solution for giving a clear perception of the programming concepts. As the programming games that already exist seem to struggle with this part, it will most likely be a challenging problem to solve, and hard to test.

The game should encourage problem-solving (TR03)

Priority: *Medium*
Difficulty: *High*

A problem stated by several teachers is that the students are not trained as problem-solvers. They give up easily, lose focus, or hard-code solutions instead of trying to find the *smart* solution. Problem-solving is crucial for programming, and the game should as such encourage it in some way.

Another of the requirements for the game is to use elements of competition and cooperation for motivational purposes. These elements may also help encourage problem-solving. If there is a competition, the students may not feel they have time or opportunity to ask the teacher for help, but instead end up trying harder to solve the task themselves. The game can also inspire cooperation between players in order to perform, and this cooperation may help them think more creatively, and help them arrive at a solution with a confident feeling of have accomplished something themselves.

The game should be an evaluation tool (TR08)

Priority: *Low*
Difficulty: *High*

One of the biggest challenges for the teachers, as stated in Chapter 3, is the big variation in the students' programming skills. As a result, the students of the ECP work with a lot of different tools and projects, varying from block-programming with Scratch to creating a game in Python. In addition, the students are able to choose the ECP all three years of LSS, while the curriculum and learning goals do not change. Some teachers therefore struggle with evaluating their students. A suggestion from one teacher was to have a game that gives certifications in different programming concepts, by for example having a set of assignments that the student has to solve in order to get a certification.

This requirement has a high difficulty level. The authors do not have the competence to know how to evaluate students, and the preferred evaluation method of teachers is likely to differ. It also requires defining a great amount of tasks, in order to avoid the possibility of passing on tasks to classmates who have not played the game yet.

The game should have instructions in Norwegian (TR09)

Priority: *High*
Difficulty: *Low*

Discovered by both some of the teachers, and during the game playing session from the specialization project (Holst and Magnussen 2018), programming assignments written in English create language barriers for some students. Instead of struggling with the assignment itself, they struggle with understanding the assignment, which is unfortunate for the impression the students get of programming. Poor English skills should not affect the possibility and motivation for learning programming.

The game should be separated into levels, where each level introduces a programming concept (SR05)

Priority: *High*
Difficulty: *Medium*

The secondary school children the authors interviewed during the specialization project, claimed that they preferred to solve small tasks instead of bigger ones, so they could focus on one thing at a time. By letting a student choose from a pool of small tasks to solve, she can progress at her own pace. However, the tasks she solves may then end up not meeting all of the learning goals. There should as such be some restrictions for the task choices.

Since the mastering of programming concepts is a learning goal of the ECP, the game could be separated into levels, where each level of the game can be structured around a programming concept. A level can further have its own set of tasks to choose from, with different difficulty levels. This way, there is less risk of not covering the learning goals.

The game should have tasks that advance in difficulty for each level (SR06)

Priority: *High*
Difficulty: *Medium*

This requirement depends on the previous requirement as it assumes that the game is separated into levels. In order to engage all students regardless of programming skills, tasks should have different difficulty levels. By gradually increasing the difficulty of the tasks, the players must improve their skills as they face new challenges, and they may find the game interesting to play multiple times as the game progresses. A concern is then to make sure the difficulty level is suitable for those who find programming hard, without boring those who master programming well. The students discussed that they liked the possibility of skipping ahead if they found the tasks very easy to solve. Another possibility is to not make the game progress linearly with the same tasks in the same order, but that the player can choose tasks from a pool where the difficulty level of tasks is evident.

4.1.1 Overview of Requirements

An overview of all of the requirements, ordered by their priority and difficulty rate, are given in the following Table 4.1. Each requirement is given a new ID (HRXX) as they now constitute a full list of high-level requirements for the game.

ID	Priority	Difficulty	Requirement
HR01	High	High	The game should have a teacher mode where they can monitor the progress of their students.
HR02	High	High	The game should use a programming language suitable for beginners.
HR03	High	Medium	The game should use a textual programming language.
HR04	High	Medium	The game should be integrated into the Elective Course of Programming.
HR05	High	Medium	The game should include the motivational elements of competition and/or cooperation.
HR06	High	Medium	The game should offer immediate feedback to the player.
HR07	High	Medium	The game should have a teacher guide.
HR08	High	Medium	The game should have tasks that advance in difficulty for each level.
HR09	High	Medium	The game should be separated into levels, where each level introduces a programming concept.
HR10	High	Low	The game should be web-based so that it is easily accessible.
HR11	High	Low	The game should teach introductory programming.
HR12	High	Low	The game should have instructions in Norwegian.
HR13	Medium	High	The game should offer students a clear perception of what they are about to learn.
HR14	Medium	High	The game should encourage problem-solving.
HR15	Medium	Medium	The game should encourage two-fold learning in the teaching setting it is used in.
HR16	Low	High	The game should use physical elements.
HR17	Low	High	The game should be an evaluation tool.
HR18	Low	High	The game's tasks should have a motivational purpose, and not just focus on learning programming.

Table 4.1: Finalized list of high-level requirements.

Chapter 5

Proposed Game Concepts

A total of eighteen requirements were gathered from the specialization project and the teacher interviews. Based on these, three game concepts were proposed, as described in Section 5.1. The different game concepts incorporate game elements of the requirements in different ways. In an attempt to evaluate these elements, the authors participated in the *Lær Kidsa Koding* conference, as presented in Section 5.2. Here they held a short presentation about their project, before collecting opinions through a survey from teachers of technology courses in lower secondary school. Based on dimensions drawn from the conference survey, the three concepts were compared and evaluated, before one concept was chosen for implementation and further evaluation.

5.1 Game Concepts

Based on the requirements presented in Chapter 4, the authors proposed three different game concepts. As the concepts were derived from the same list of requirements, they have several elements in common. These are presented in Section 5.1.1. The differences between the three game concepts are elements that do not directly originate from the requirements, like individual versus team competition, the game's purpose, and graphics.

5.1.1 Common Game Elements

Small Independent Tasks

The game should have small independent tasks that the students must solve to get closer to a goal. These tasks should be structured into different topics, so that each level of the game can introduce a new, more difficult topic. This is based on two of the high priority requirements, *HR08: The game should have tasks that advance in difficulty for each level*, and *HR09: The game should be separated into levels, where each level introduces a programming concept*. The structuring of tasks into topics also supports *HR13: The game should offer students a clear perceptions of what they are about to learn*, as they know the topic of the tasks. This way, the game encourages two-fold learning (HR15), which is seen to be an effective teaching approach (Mayer 2013; Kaučič and Asič 2011; Tangney et al. 2010; Majgaard 2015), since the teacher then knows what to go through before a session of the game.

In the game concepts the task difficulty is mapped to colors, so that the students are able to choose tasks based on difficulty. The color red represents hard tasks, yellow represents medium tasks, while green represents easy tasks.

Competition

Most games are based on some sort of competition, where the player either competes against other players, or against the computer. Since one of the high-level requirements is to include competition or cooperation, the authors decide to include the elements of competition. As stated in Section 2.3, competitions can help engage children to learn a topic in order to perform in a competition (Tomcsányiová 2013). Competition is also a natural element in a multiplayer game, and since graphics may not be a big part of the game, the competition-element can act as the main motivational factor. With small independent tasks, there are several ways to include competition. The students can compete individually by trying to solve as many tasks as possible, which is the goal of game concept 1, or the students can compete as teams to solve tasks.

Hints

The game concepts have a feature called *Hint*. This is related to each individual task, and the main purpose of it is to encourage students to help each other, or to avoid students getting stuck on a task. If a player struggles with a task and clicks on the *Hint* button, two different options will appear. One of them allows the student to see the solution code written in block code, while the other allows the student to ask another player for help. To promote usage of the latter, this option should include bonus points that both students receive if they ask for or give help to others. The latter additionally encourages cooperation, which is seen to engage students to learn a topic (Shim, Kwon, and Lee 2017). The former can help encourage problem solving (HR14), as hints provided by the game can help students in the right direction for solving a task, without handing out the solution.

Graphical User Interface

All of the game concepts have a graphical user interface that is split into two parts. The left part includes an editor with a *Run*-button and a console. This is were the students write their code and see the output of it. The game concept itself is presented in the right part of the user interface, so this part differs between the three concepts.

Editor and Console

Since a requirement for the game (HR03) is that it should use a written programming language, all game concepts include an editor for writing code. The chosen language for all games is Python, as it is seen to be widely used in the ECP, and since it is a language recognized in the literature to be suitable for beginners (Pears et al. 2007). This is also covered by HR02. It should be possible to run the code and immediately get the output in a console, in order to offer immediate feedback to the player, as is stated by HR06. Additionally, the completion of a task should trigger some visual feedback to the player.

Teacher View with Analytics

All of the concepts are meant to have a teacher view, as of HR01, where the teacher can create games, and is offered analytics during and after game-play. Game analytics can easily be

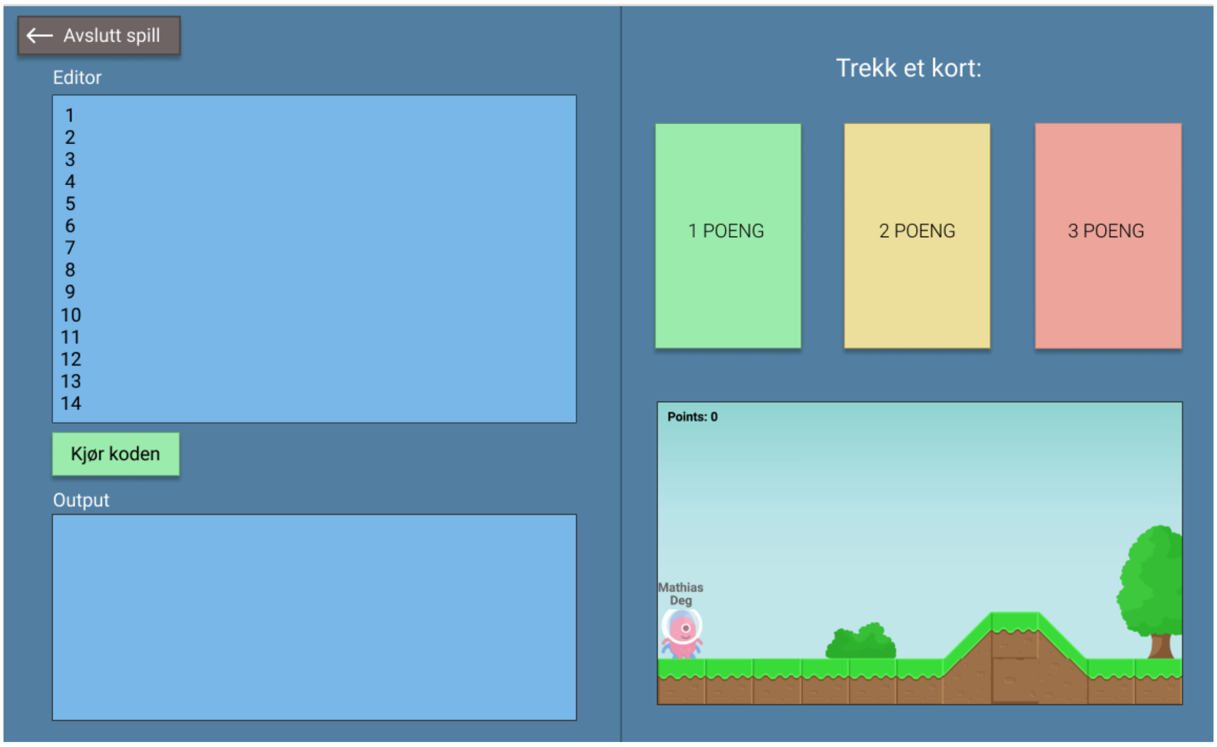


Figure 5.1: Game concept 1: Initial state

included by logging student code, time spent on a task, scores, and so on in a database. The teacher view should offer an overview of each student, which is automatically updated during game-play. The teacher should also be able to view analytics after the game through this view, and possibly see trends in the data over time.

The game should come with a predefined set of tasks, to facilitate employment of the game. A plug-and-play game calls for a low threshold for employment. However, to accommodate more teachers, there should also be functionality for defining new tasks through the teacher view. This way, tasks can be tailored to the level of each specific ECP class. It also introduces the possibility to use the game several times, even when the students have solved all of the predefined tasks that come with the game.

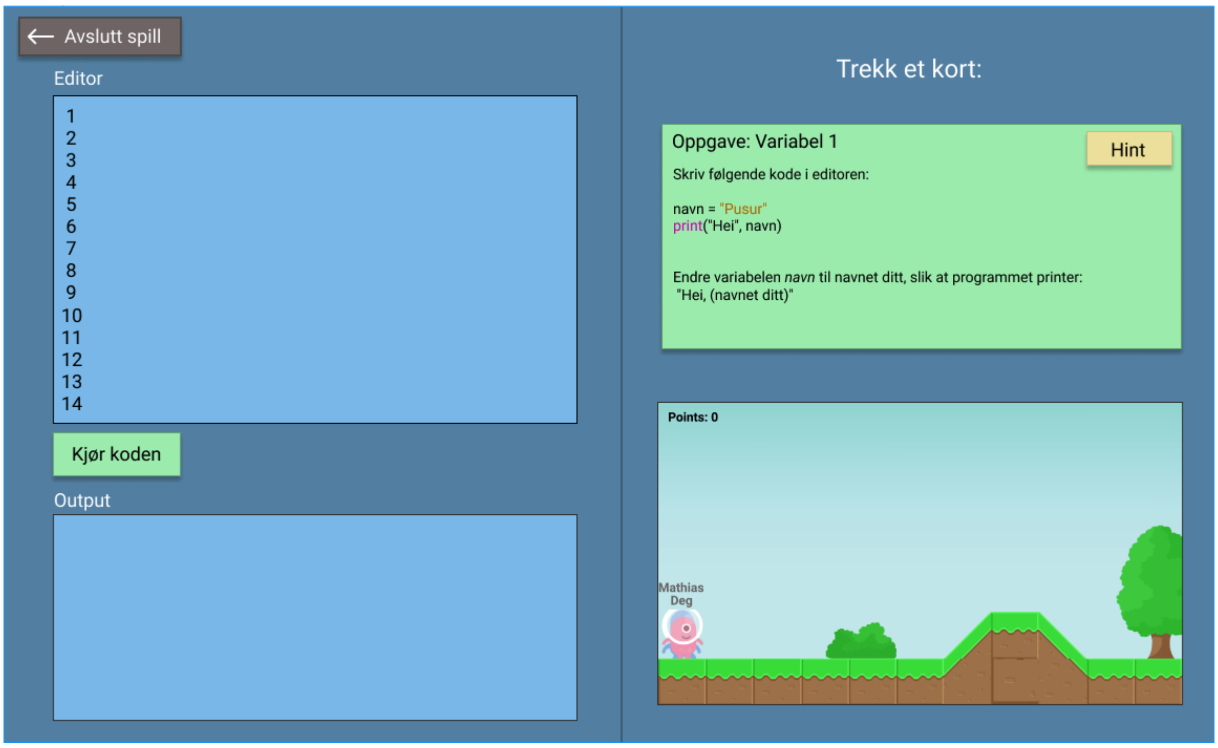
HR07 states that the game should come with a teacher guide. The teacher view should be designed to be sufficiently intuitive to use in order to not require a teacher guide. Should such a guide be included, it would be natural to place this in the teacher view.

5.1.2 Game Concept 1

The first game concept offers an individual competition where the player earns points by solving tasks. As shown in Figure 5.1, it consists of a platform game, complemented by three task-cards. The cards are indicated with the amount of points they reward, depending on their difficulty. The platform game view shows a 2D-world, where the players are visualized as different characters. As a player earns points, she moves rightwards corresponding to the amount of points.

The player starts by choosing one of the three cards. When a card is chosen, it is flipped and expanded, as shown in Figure 5.2. If the player struggles with a task, she has the possibility to get a hint by pressing the *Hint*-button. The first player to reach the finish line wins the game.

The intention of this concept is to promote motivation and excitement by involving compe-



Oppgave: Variabel 1

Skriv følgende kode i editoren:

```
navn = "Pusur"
print("Hei", navn)
```

Endre variabelen *navn* til navnet ditt, slik at programmet printer: "Hei, (navnet ditt)"

Figure 5.2: Game concept 1: Card selection state

tition and fun graphics. It is formed as a single-player game, because by letting students write code themselves, everyone will obtain more hands-on experience. The graphics are based on the concept of a horse race, where it is easy to see the overview of the ranking of each player, in order to engage the players and to arouse their competitive instinct.

5.1.3 Game Concept 2

The second game concept offers competition in teams where the players of each team solve tasks individually, but together work towards a main goal. The goal is to guess what the picture hidden behind a board shows. The board consists of several cards, which can be seen in Figure 5.3. Each card represents a task, and by clicking on one of them, the card will be flipped to display its task, as shown in Figure 5.4. By solving a task, its card will be removed from the board, and a piece of the picture will appear.

Each team shares the same board. This means that they cannot choose the same tasks. However, if a player selects a task that is too hard, she can close the card, and it will become available again for other team members. The team size for this game is thought to be around 2 - 4 players, considering the size of the board, and thus the amount of tasks each player must solve to be able to guess what is underneath. A small team size also makes it easier to locate the students in groups in the classroom, so that they can discuss both tasks and what the picture shows.

This game concept promotes motivation by using competitive and cooperative elements. The competitive element is driven by the main goal: The first team to guess the picture wins the game. The cooperative element is driven by having teams. Even though the players solve the tasks individually, they contribute to the same board, and therefore win or loose as a team.

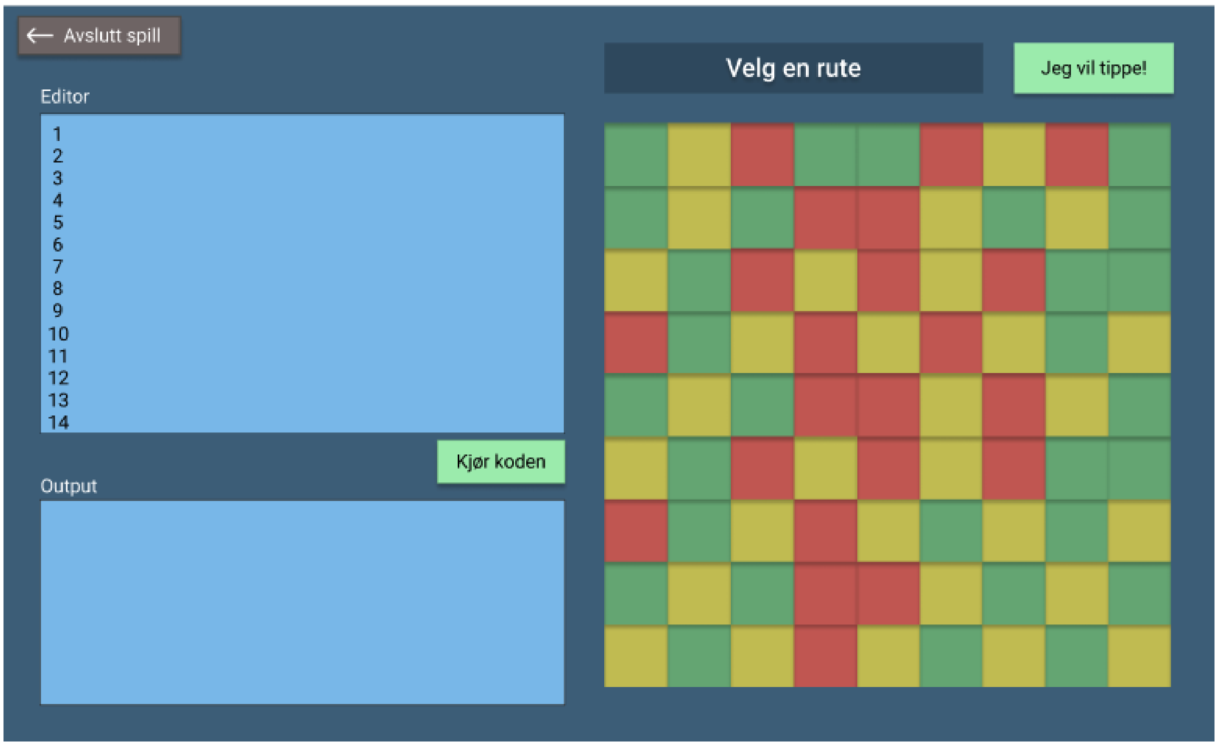
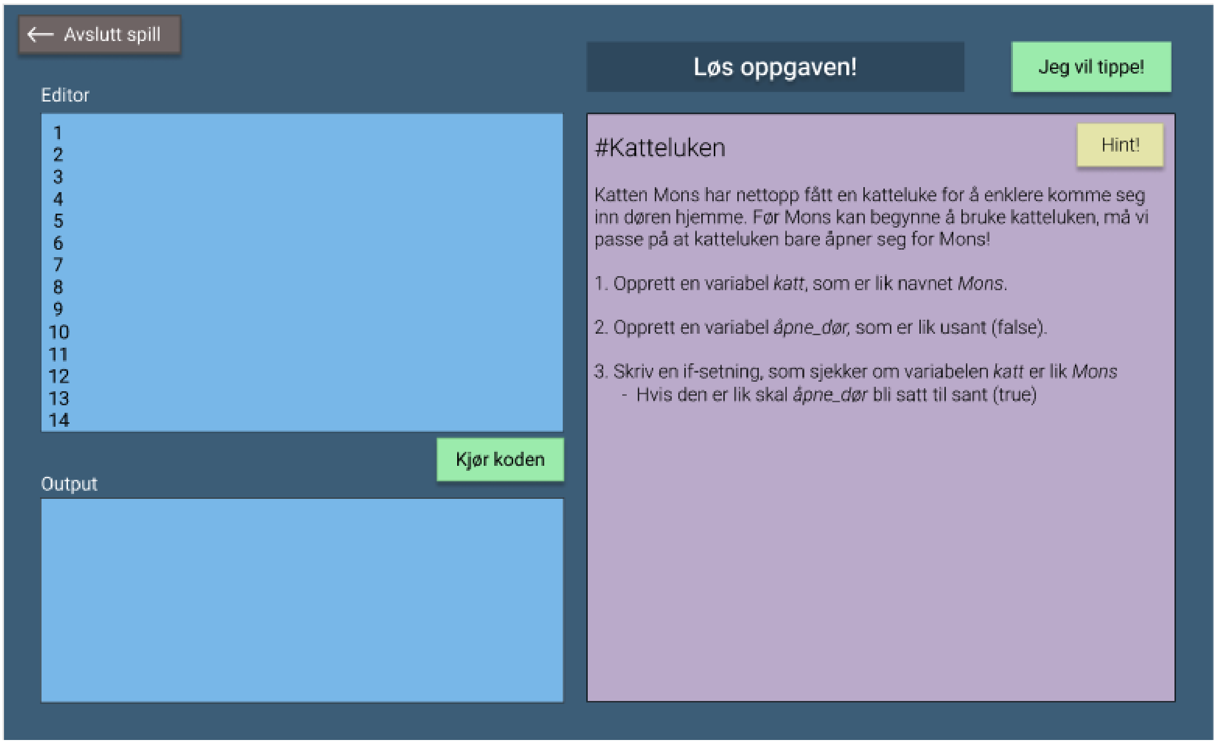


Figure 5.3: Game concept 2: Initial state



#Katteluken

Katten Mons har nettopp fått en katteluke for å enklere komme seg inn døren hjemme. Før Mons kan begynne å bruke katteluken, må vi passe på at katteluken bare åpner seg for Mons!

1. Opprett en variabel *katt*, som er lik navnet *Mons*.

2. Opprett en variabel *åpne_dør*, som er lik usant (false).

3. Skriv en if-setning, som sjekker om variabelen *katt* er lik *Mons*
   - Hvis den er lik skal *åpne_dør* bli satt til sant (true)

Figure 5.4: Game concept 2: Card selection state

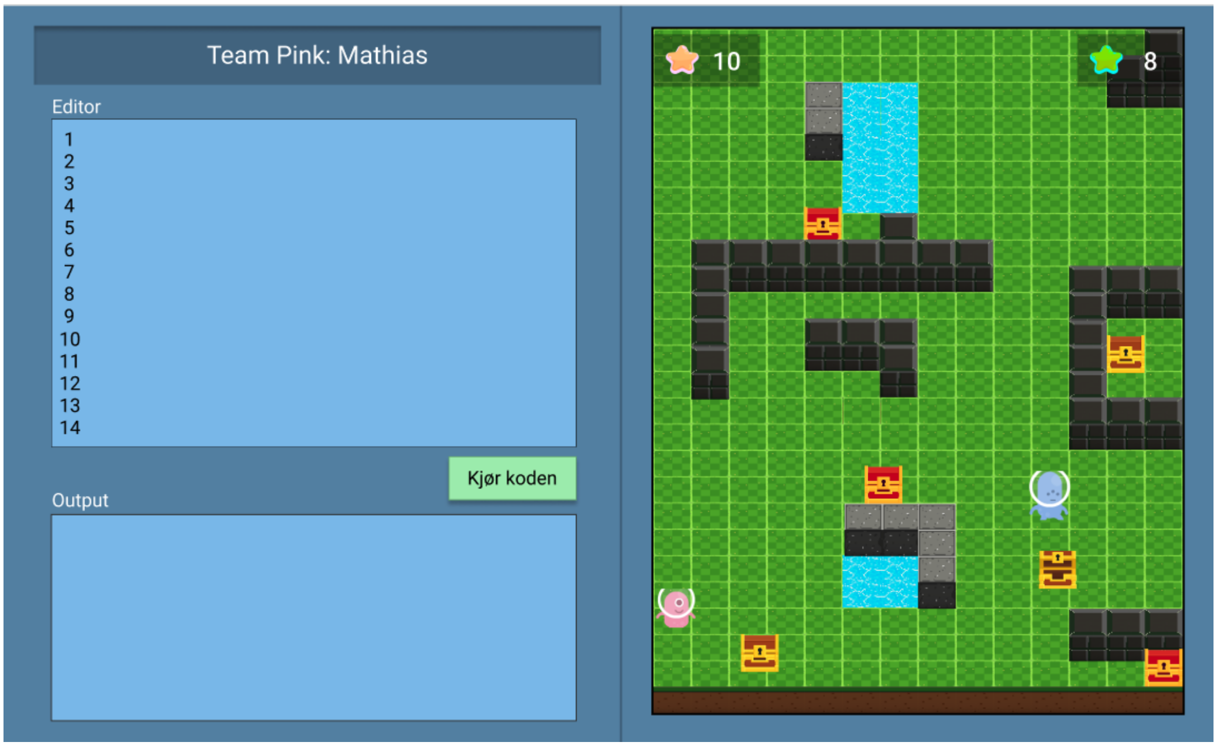


Figure 5.5: Game concept 3: Initial state

5.1.4 Game Concept 3

The third game has a team-based competition where the players of each team work individually, but together work towards a main goal. In this game, the main goal is to earn points by opening chests that are placed around in a fictive world. The team that has the most points when all chests are opened, wins the game. The differences between this concept and game concept 2 are the graphics and the goal.

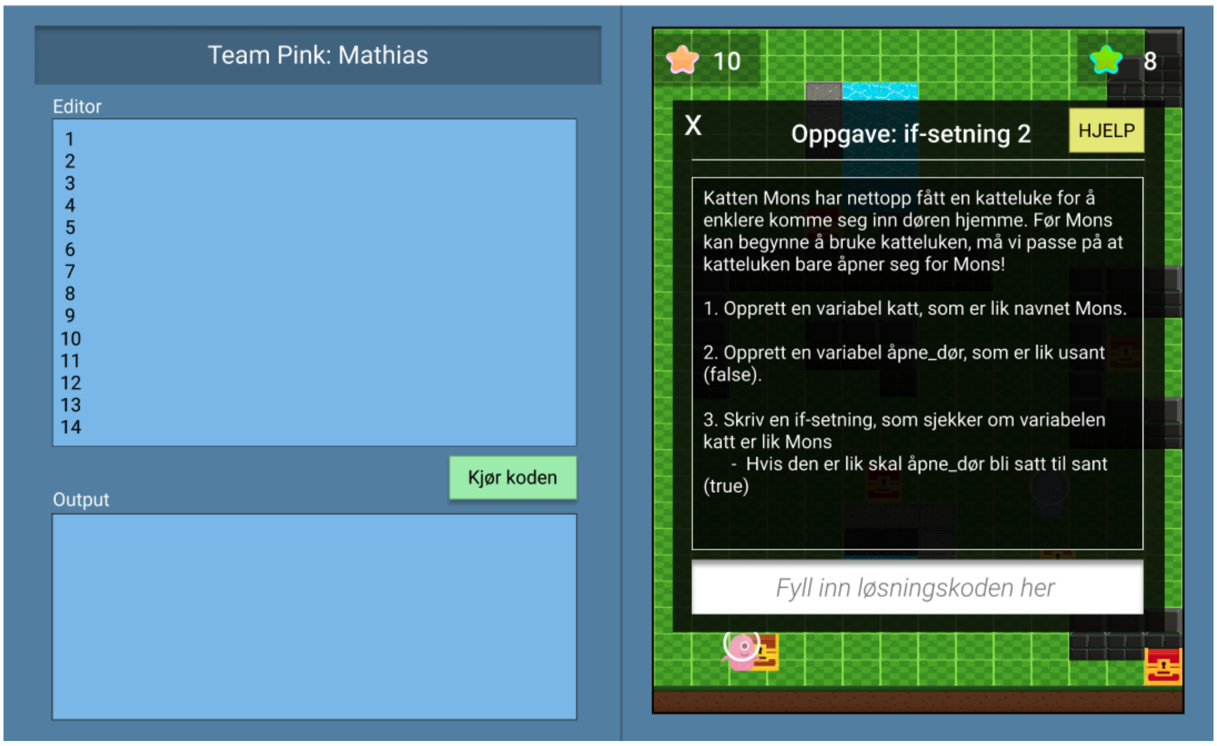
As shown in Figure 5.5, this game concept includes more graphics than the other two. It is a 2D platform game, where the graphical user interface only shows a part of a bigger board. The player moves her character by using her keyboard, and the view will move corresponding to the players' character. All of the players, regardless of team, play on the same board. The board is filled with chests of different colors, and in order to open a chest and get the points, the player has to approach a chest and solve a programming task. The colors of the chests represent the difficulty of the tasks they contain.

This game concept is meant to promote engagement by having rich graphics. It additionally includes competition through the goal of collecting the most points. The players are divided into teams to promote cooperation. As an example, the players can ask players on their team for help by clicking on the *Hint*-button. They also win or loose as a team.

5.2 Feedback for Selecting a Game Concept

In order to evaluate the different game concepts, the authors participated in the *Lær Kidsa Koding* conference. This was hosted by *Lær Kidsa Koding*[1] (LKK) in cooperation with the

[1]https://kidsakoder.no/



Oppgave: if-setning 2

Katten Mons har nettopp fått en katteluke for å enklere komme seg inn døren hjemme. Før Mons kan begynne å bruke katteluken, må vi passe på at katteluken bare åpner seg for Mons!

1. Opprett en variabel katt, som er lik navnet Mons.

2. Opprett en variabel åpne_dør, som er lik usant (false).

3. Skriv en if-setning, som sjekker om variabelen katt er lik Mons
   - Hvis den er lik skal åpne_dør bli satt til sant (true)

Fyll inn løsningskoden her

Figure 5.6: Game concept 3: Card selection state

Department of Computer Science and Informatics at NTNU, and Bebras[2] (Koding 2019). The participants were Norwegian lower secondary school teachers seeking to learn more about technology in teaching, some of whom teach the ECP. A five-minute talk was held by the authors to present their master project and their findings so far. Participants were provided with a printed survey covering how different game elements can affect motivation and learning outcome. Additionally, there were questions about attitude towards educational games, and whether they have used games in teaching earlier, to get more significant quantitative results to topics already covered by the interviews.

5.2.1 Survey Questions

Some of the survey questions were articulated in order to get more significant answers to questions already asked under the teacher interviews. Hence, the three first survey questions were identical to questions from the interviews, where two of them now were in the format of a 7-point Likert Scale.

- Have you ever used games in the lectures, in that case which games?
- A game can be used to increase knowledge in programming. (7-point Likert Scale)
- A game can be used to increase engagement for programming. (7-point Likert Scale)

The main intention of performing the survey was however to evaluate the elements of the proposed game concepts, which were based on requirements for the game. Some of these elements were directed at supporting teachers, and some at increasing motivation or knowledge

[2]`https://bebras.no/`

about programming. The game elements were extracted and categorized under these three contexts. A question was added for each context, asking to prioritize game elements between 1 and 6 in this context. There was also an ”other”-field for each context in case the respondents had elements they want to add.

- What type of support would you want in a game for teaching programming? (Prioritize 1-6)
- Which elements do you think are useful to gain knowledge about programming? (Prioritize 1-6)
- Which elements do you think are useful to increase motivation for programming? (Prioritize 1-6)

In particular, the game elements below were due for further evaluation, as they either differed between the game concepts, or had different ways of being approached. The original survey can be seen in Appendix D.

Teacher View with Analytics

This element could use some clarifications, as there are several ways of monitoring progress of students. For instance, the game could provide analytics about the performance of their students while they are playing, giving them an indication about who needs help, or how they approach different problems. Analytics can also be provided after the game is finished, showing how much time a student has used to solve a task, how she has solved it, how many tasks she has solved, and so on. This may be useful for evaluation in the course as well, which is part of requirement HR17. Consequently, game elements added to the teacher support-context are:

- Get analytics about the students' results during the game.
- Get analytics about the students' results after the game.

Small Independent Tasks

This element originates form the wishes of students participating in the workshop as part of the specialization project. As such it is not certain how the teachers prefer tasks to be separated or categorized. Is choosing tasks based on topic or difficulty level considered most important? Seeing that teachers of the ECP have a differing level of programming competence, their opinions on this matter are likely to differ. Some teachers may also want to define their own tasks, in order to customize the game to the level of their students, while others may prefer using predefined tasks. The game elements below are added to the teacher support-context.

- Choose tasks in the game based on topic.
- Choose tasks in the game based on difficulty level.
- Being able to define new tasks in the game.

Competition and Cooperation

These elements originate from requirement HR05, which is highly prioritized. It would be useful to find out if competition or cooperation is most valued, or if a combination is preferred. Additionally, the element of cooperation raises the question of group sizes - should the students cooperate in pairs or in groups of several students? The following game elements are added to the contexts of motivation and knowledge gain.

- Cooperation in teams
- Cooperation in pairs
- Competition
- The opportunity to help each other
- The opportunity to get hints if one gets stuck

The Purpose of the Game

The requirement *HR18: The tasks should have a motivational purpose, and not just focus on learning programming* is already covered by the questions about competition and cooperation. However, graphics can also be a way of increasing the motivational purpose of the game. This game element was uttered to be popular among the students participating in the co-design workshop. As such, *graphics* is also included as a game element in the contexts of motivation and knowledge gain.

5.2.2 Results

23 answers were collected through the survey. In the following sections, each question is presented with its results. For the last three questions, several participants marked which element(s) they found important instead of prioritizing them. It may have been because the survey instructions were not clear, or that the respondents rushed through the surveys, as there were a lot of other things that required their attention in the conference. Two respondents skipped some of the last three questions. These also responded that they had not used educational games before, so maybe they simply did not know what game elements they found important. As a consequence, not all answers sum up to 23.

The results of the last three questions were processed in two different ways, according to how they were answered. The results were then analyzed looking at the two different data representations as a whole.

Question 1: Have you ever used games in the lectures, in that case which games?

Five of the teachers had used one or several games in their lectures. Most of the games only had one occurrence, except for CodeCombat which occurred twice. Combining this with the results from the interviews with teachers of the ECP (Chapter 3), gives a total of 30.6% teachers who have used games in their lectures - 11 out of 36.

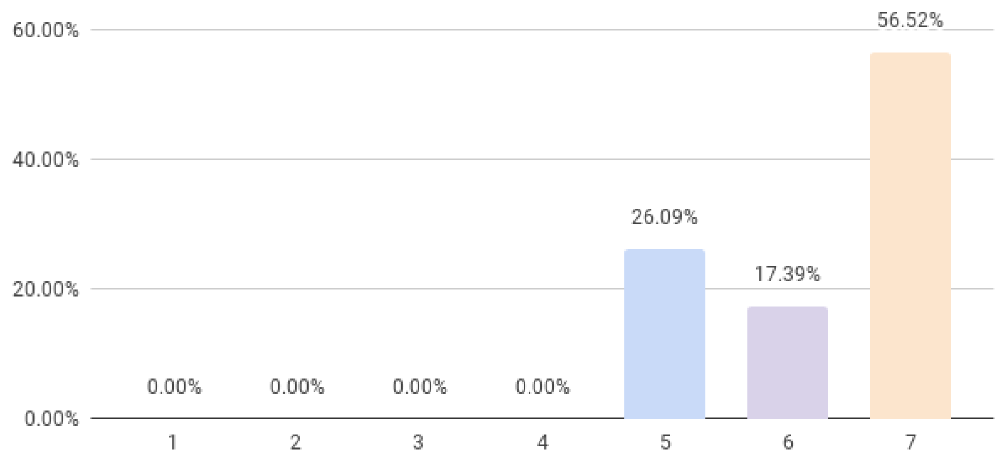


Figure 5.7: Results from question 2: *A game can be used to increase knowledge about programming.*

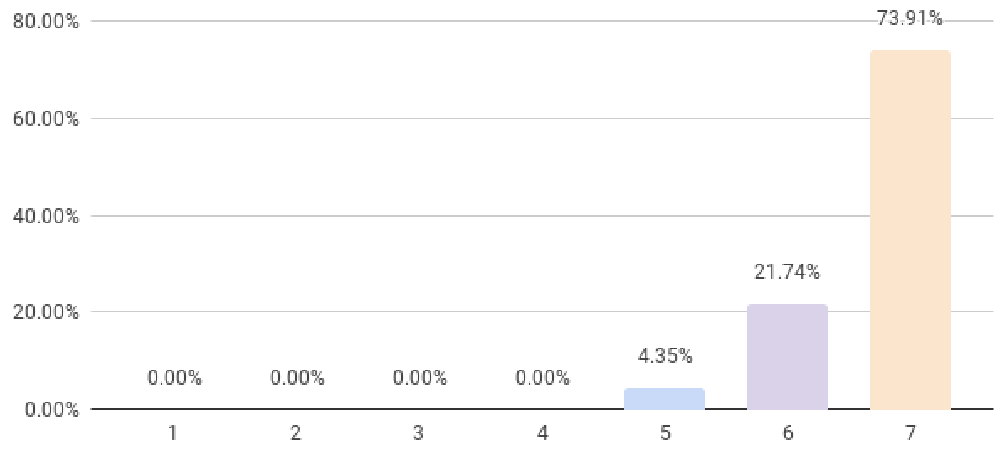


Figure 5.8: Results from question 3: *A game can be used to increase engagement for programming.*

Question 2: A game can be used to increase knowledge about programming.

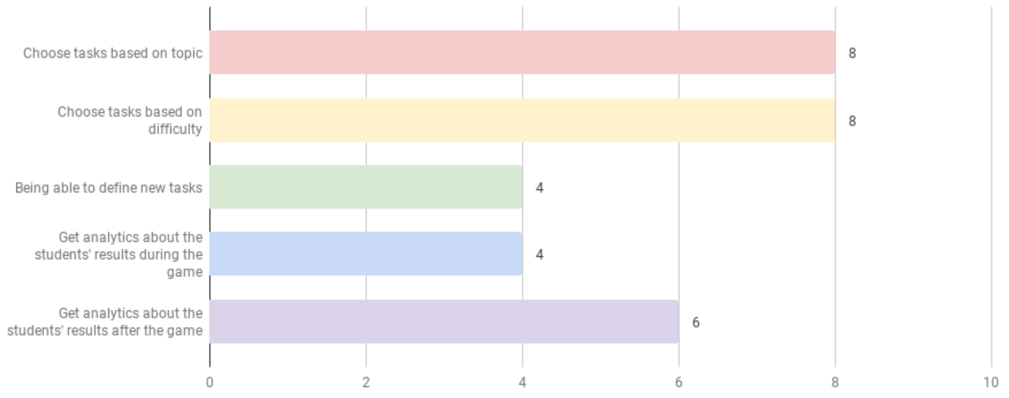
Figure 5.7 shows the results of question 2. All respondents answered between 5 and 7, agreeing that a game can increase knowledge in programming. This coincides with the attitudes of the interviewees.

Question 3: A game can be used to increase engagement for programming.

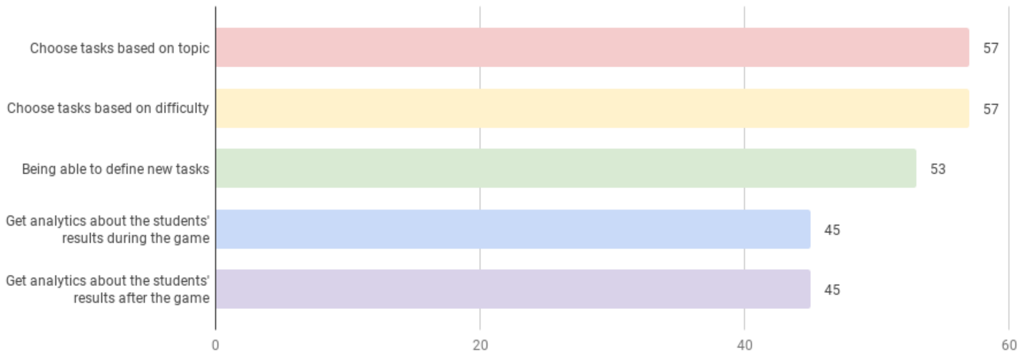
The results of question 3, as seen in Figure 5.8, show that the teachers very much agree that a game can be used to increase motivation for programming. Only one teacher answered 5 on the Likert-scale, while the rest answer 6 and 7 (Strongly agree).

Question 4: What kind of support do you want in a game that teaches programming?

The results are split into two different graphs due to different ways of answering. Figure 5.9a shows the results from teachers who used crossing instead for ranking, while Figure 5.9b shows the ranked answers. For the crossing there is a total of 9 answers. The ranked answers has a total



(a) Crossing results



(b) Ranking results

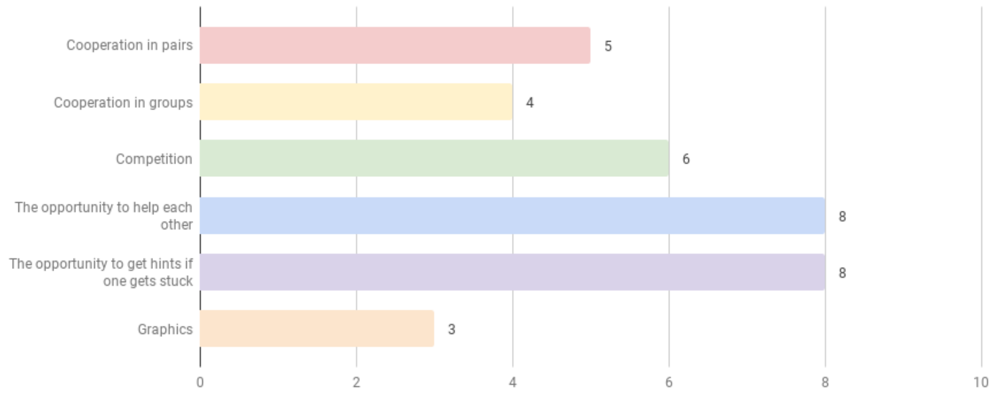
Figure 5.9: Results from question 4: *What kind of support do you want in a game that teaches programming?*

of 13 answers. Both of the graphs give a clear view of what kind of support the teachers want the most, namely *Choose tasks based on topic* and *Choose tasks based on difficulty*. This may be connected to the teachers' concern with the diversity of the students' skills in programming, as stated in Section 3.4.3.

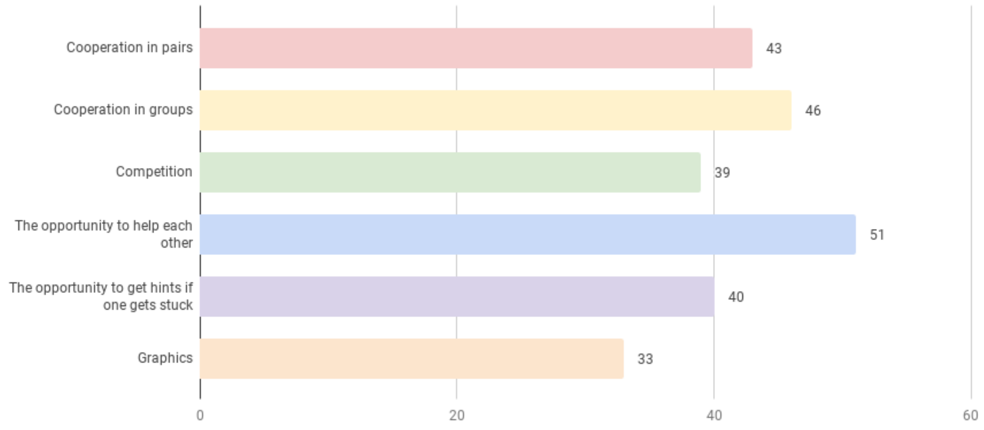
The difference between the highest and lowest ranked element are only 12 points. This means that the teachers ranked the elements very differently, and even if the support regarding analytics has the lowest rank, it is still important for many of the teachers. From the interviews it was also found that some teachers struggle with the evaluation of their students, in which analytics might be able to assist them.

Question 5: Which elements do you think are useful to gain knowledge in programming?

Figure 5.10a shows the crossing answers, with a total of 10 answers. Figure 5.10b shows the ranked answers, with a total of 12 answers. From both graphs graphics are seen to be least important for the teachers when it comes to gaining knowledge in programming. This may be because they think it is the educational part that actually teaches the students and should weigh more, hence graphics come out the weakest amongst the six elements. However, the specialization project (Holst and Magnussen 2018) found that graphics are very essential for the students.



(a) Crossing results



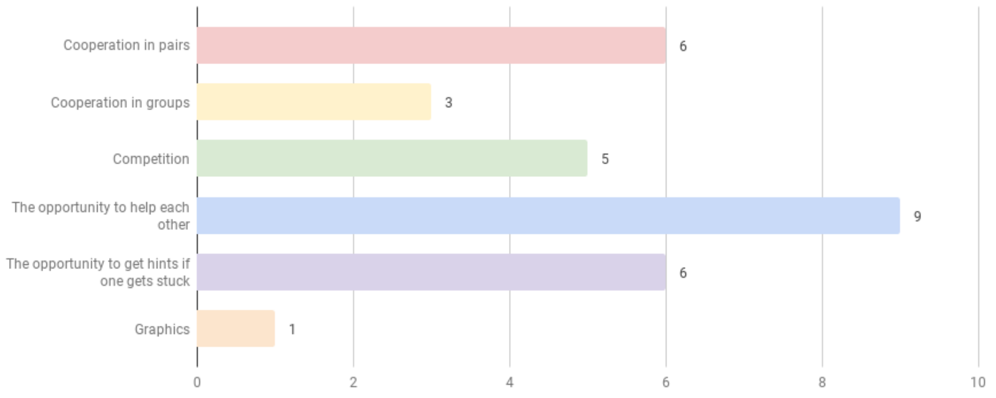
(b) Ranking results

Figure 5.10: Results from question 5: *Which elements do you think are useful to gain knowledge in programming?*

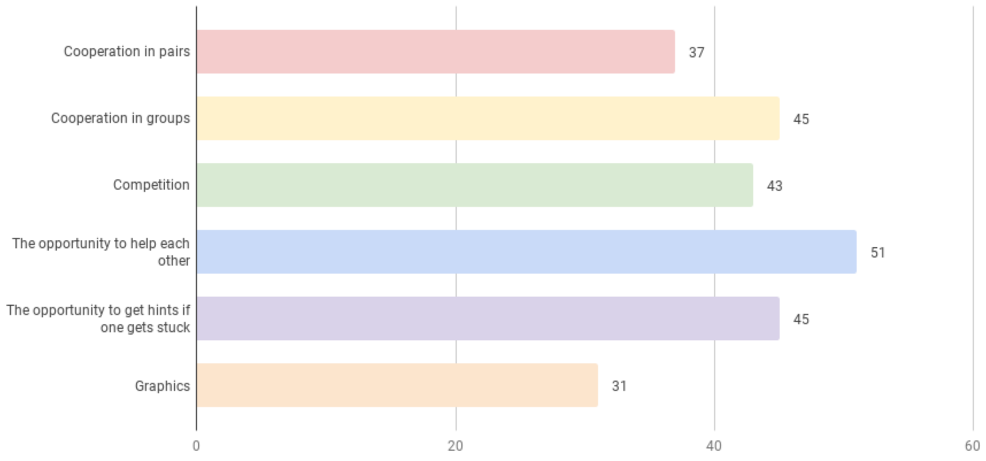
Common for the answers is that the element *The opportunity to help each other* is the most important for gaining programming knowledge. During the interviews (Chapter 3) it was found that many of the teachers did not have any programming background, which can make them think that the students learn as much from each other as from them. Some teachers also stated that the students were at very different levels, and that a lot of time was spent helping students with different problems, hence it would be easier if the game encouraged the students to help each other.

Question 6: Which elements do you think are useful to increase engagement for programming?

Figure 5.10a shows the crossing answers, with a total of 10 answers. Figure 5.10b shows the ranked answers, with 12 answers in total. When it comes to engaging the students in programming, the element *The opportunity to help each other* weighs the most and *Graphics* weighs least, the same results as of question 5. Combining the results also indicates that the rest of the answers are even in importance. This may suggest that the respondents find it difficult to range



(a) Crossing results



(b) Ranking results

Figure 5.11: Results from question 6: *Which elements do you think are useful to increase engagement for programming?*

or select elements because all of them are quite important, or because they are not familiar with educational games, hence choosing elements for such game was not easy.

5.2.3 Discussion of Survey Results

Altogether it is clear that the teachers have a positive attitude towards educational games, which most likely means that teachers will be interested in a new game customized for the ECP. However, when choosing a game concept several elements must be taken into account. A lot of the elements that teachers did not weigh highly, are elements that the students find very important. Examples are graphics and competition. From the authors' perspective, less graphics means easier game development. Considering the time constraints for this project, it is important to find a balance between making the game possible to develop, and making the graphics good enough to engage students.

The teachers are also concerned with the selection of tasks, hence the tasks are important to consider when choosing a game concept. Since the results from both question 5 and 6 reflect that *The opportunity to help each other* is important, the game should in some way incorporate this element. The rest of the elements are fairly equally weighted, and should all be considered in the process of choosing a concept.

5.3 Discussion

In order to select a concept to proceed with, they were compared and evaluated according to results from the LKK survey. The population size of the survey was not large. The concepts were as such also evaluated according to how they meet the game requirements, their inclusion of elements seen as important by teachers and students, as well as whether the authors would be able to implement them in the course of two months.

The game concepts were created with the list of requirements in mind, but that does not mean that the list was followed slavishly. Requirements that were not specifically kept in mind are those with low priority and high difficulty - HR16-18. The use of physical elements (HR16) is not fulfilled by any of the concepts. As mentioned in Chapter 4.1, there already exist several resources that work well, and it would be hard to implement considering the experience of the authors and the time constraints. HR17 involves using the game as an evaluation tool. None of the game concepts are evaluation tools in themselves, but they can all serve as tools aiding the teacher when evaluating their students, through the analytics of the teacher view. HR18 states that the game should have a purpose that goes beyond learning programming. This requirement has the objective of ”disguising” the programming tasks in something fun, and involves that the main goal of the game is something else than just solving tasks correctly. The concepts 1 and 3 reward players with points when solving tasks, but the main goal of these concepts is still simply to solve programming tasks. Concept 2 on the other hand, has the objective of guessing what is displayed on a picture in addition to solving tasks.

Most of the game elements that were evaluated by the teachers are included in all three game concepts. Those that are not are cooperation in teams and graphics. Cooperation was seen to be important both according to students in the specialization project workshop, and amongst teacher from interviews and LKK-survey, and it is a vital part of game concepts 2 and 3. Graphics were claimed to be important to the students, but not that important to the teachers. All game concepts focus on graphics to some degree, but concept 3 stands out with a lot of graphical focus. The main view of this game is namely a 2D platform game where the player can navigate with the arrow keys. However, rich graphics call for a lot of work and processing power. There exist several game engines that can alleviate the work-amount, but the authors are not certain that this game concept would be manageable to implement in two months. They were however certain that the concepts 1 and 2 would be possible to implement using mainly resources they were already familiar with. Not needing to learn a lot of new technology could save time for implementation.

A comparison of the game concepts is seen in Table 5.1. The games are compared according to the dimensions that differ between them. Seeing that game concept 2 meets the requirements of all dimensions, this concept was chosen to proceed with.

	Purpose Beyond Programming	Cooperation in Teams	Focus on Graphics	Manageable to Implement
Game Concept 1	No	No	Some	Yes
Game Concept 2	Yes	Yes	Some	Yes
Game Concept 3	No	Yes	A lot	Doubtful

Table 5.1: Comparison of the game concepts.

Chapter 6

The Game: Code Flip

This chapter presents a description of the initial version of the game Code Flip, which was developed based on game concept 2, described in Chapter 5. Some changes were made to the original concept during the development, as specified below. A technical description of the game can be found in Chapter 7. The following Chapters 8 and 9 describe how the second and third versions of Code Flip were developed, according to feedback from a pilot evaluation and an expert evaluation. The third version of the game was evaluated in a class of the Elective Course of Programming at a lower secondary school, as described in Chapter 10. This version is the final one of this thesis, but new versions may be developed in the future. A demo of the final version of Code Flip can be seen at `https://youtu.be/kevnpYWTbmU`. The demo is created as an introduction of the game, but may also be used as a teacher guide.

Code Flip is a web-based game targeting the ECP, where students compete in teams and solve programming tasks from a board, in order to uncover a picture hidden behind the board. The goal of the game is to guess what the picture displays. The programming in the game is performed by the students through an integrated editor accepting Python code. The game comes with a set of predefined tasks based on learning content seen to be widely used in the ECP. The programming tasks are highly independent of the game concept, and there is hence room for teachers to define new tasks, in order to accommodate teachers with varying programming competence. Through a teacher view, teachers can see analytics about the performance of their students in the game.

6.1 Learning Goals

Code Flip is specifically designed to be used in the Elective Course of Programming. The learning goals of the ECP are divided into two modules - *Modeling* and *Coding*, as can be seen in Table 2.2. From interviews with the teachers, it was seen that the challenges faced by teachers were connected to the *Coding*-module, while the *Modeling*-module is already well covered by their learning activities. As such, the game aims at covering the learning goals of the *Coding*-module.

According to a study of the ECP (Corneliussen and Tveranger 2018), students struggle with understanding how programming principles apply for textual programming languages. This was corroborated through the teacher interviews of this thesis. As such the learning goals of *Be able to program in the textual language Python* and *Use basic concepts of programming, such as variables, loops, if/else, and functions* are heavily weighted in the game. Another challenge seen to be faced by teachers of the interviews, is that students give up quickly when they face problems, as they are not trained as problem-solvers and find troubleshooting hard. Thus, a learning goal is *Use error messages to troubleshoot code*. The last learning goal of the *Coding*-module, involves knowledge transfer - being able to generalize solutions to problems. This was also seen to be a big challenge for several of the teachers. Below follows a description of how Code Flip aims to cover the different learning goals.

Be able to program in the textual language Python

An editor is included in the game for writing and running Python code, in order to cover this learning goal. The game offers hints to students solving tasks to help with syntax errors. The teacher should hold an introduction to Python in advance, so that the students are somewhat familiar with Python syntax when they play the game. Since it was seen that all the interviewed teachers already hold an introduction to a textual language, and most of them do this in Python, teachers do not have to change how they teach the course in order to use the game in their lectures.

Use basic concepts of programming, such as variables, loops, if/else, and functions

This is included through separating the programming tasks of the game into different topics. Each task instructs to use one or several programming concepts, and provides explanations of the concepts to students who struggle with a task.

Use error messages to troubleshoot code

In order to help the students with troubleshooting code, the game offers error messages in Norwegian and with a simple and direct language. They are also given similar messages when they fail to solve a task correctly. The error messages explain the mistakes, and some of them additionally include hints pushing the students in the right direction. Students will not face such error messages in other coding situations, but the idea is that they should become accustomed to paying attention to them and using them, and thus end up using them more as they write code in the future.

Be able to transfer solutions to new problems

This is not handled by the game concept in itself, but it is ensured through the predefined task sets. The sets have several tasks that are similar, so that students can use previous solutions. There are also tasks where students are instructed to correct or adjust a code skeleton.

6.2 Game Content

Game concept 2 was designed with a board of 81 cards. This would require the authors to create 81 tasks for each set of tasks. Instead, it was decided to create a 4x4 board, consisting of 16 tasks, to decrease workload that was not crucial for the results of the project. This change was likely to make the players guess the picture faster, since solved tasks uncover bigger pieces of the picture. However, this should not affect the main evaluation of the game concept, and it will be straight-forward to change this in the future.

```
    {"task1": {
        "code_requirements": ["return"],
        "default_code":
            "tall = [1, 2, 3, 4, 5]
             def summen(tall):
                #din kode her
             summen(tall)",
        "difficulty": 1,
        "error_hints": ["Har du husket å returnere summen?", ""
           , "Summerer du alle tallene?"],
        "output_requirement": "",
        "test": "assert summen([1,2,3]) == 6",
        "text": "Fullfør funksjonen summen() slik at den
           summerer tallene i listen tall, og returnerer summen
           .",
        "title": "Funksjoner 5"
        }
    }
```

Figure 6.1: Task example showing the structure of tasks in the database with JSON.

6.2.1 Programming Tasks

When defining tasks for the game, the following details need to be provided: *title*, *difficulty*, *text*, *default_code* (optional), *tests*, *code_requirements*, *output_requirements*, and *error_hints*. The fields *tests*, *output_requirements* and *code_requirements* are responsible for validating the solution of a task, submitted by a student. They should check for correct output, as well as detect hard-coded solutions. An explanation of the different fields is shown in Table 6.1, while an example of how a task is structured in the database can be seen in Listing 6.1.

Two sets of tasks were created together with the first version of Code Flip. These were named *Functions* and *Easy Mix*. *Functions* contains tasks involving the use of functions, while *Easy Mix* covers more basic programming concepts, such as variables, loops and if/else statements. The task sets can be seen in Appendix E. The programming tasks are inspired by the Python-content of Lær Kidsa Koding (LKK)[1], CodeCademy[2], and code.org[3]. This is a resource seen to be used by many teachers of the ECP during the teacher interviews, and the difficulty level of the tasks should therefore be appropriate. The content should also be familiar to the students testing the game during evaluations. The tasks of LKK are divided into the levels *Introduction*, *Beginner*, and *Experienced*. The three different difficulty-levels of Code Flip - green, yellow and red - reflect the LKK levels.

The initial plan for the game was to let teachers be able to define their own tasks, in addition to choosing from a predefined set of tasks. This feature was not prioritized in the initial version of the game, as some predefined sets of tasks should be enough to evaluate the game in a classroom setting. There is already support in the game for adding tasks directly to the database, but a user interface would be needed for teachers to be able to do this themselves. Thorough

[1]https://oppgaver.kidsakoder.no/python
[2]https://codecademy.com/
[3]https://code.org/

Field	Format	Purpose
`title`	Text field	Contains the title of the task.
`text`	Text field	Contains instructions about the task. May also contain explanation of concepts that are required to solve the task.
`difficulty`	Integer between 1 and 3	Contains the difficulty level of the task, where 1 is easy and 3 is hard. These are mapped to the colors green (1), yellow (2), and red (3) in the game.
`default_code`	Code skeleton	Specified for tasks where students are supposed to manipulate a provided code segment.
`code_requirements`	List of strings	String(s) that should be contained in the submitted code if it correctly solves the task.
`output_requirement`	Text field	String that should appear in the output of the submitted code when it is run, if it correctly solves the task.
`tests`	Assertion(s)	Test code used for verifying whether the variables and functions of the submitted code serve the correct purpose for solving the task.
`error_hints`	List of three strings	The hints that are displayed when a students submits code that does not correctly solve the task. If the submitted code does not contain all of the code requirements, the first element of the list is displayed, with hints about what is missing. The second list-element is shown if the code output does not contain the output requirement. If the assertions of the tests-field fail, the third element of the list is displayed.

Table 6.1: Explanation of the fields of the task structure, including their format and purpose.

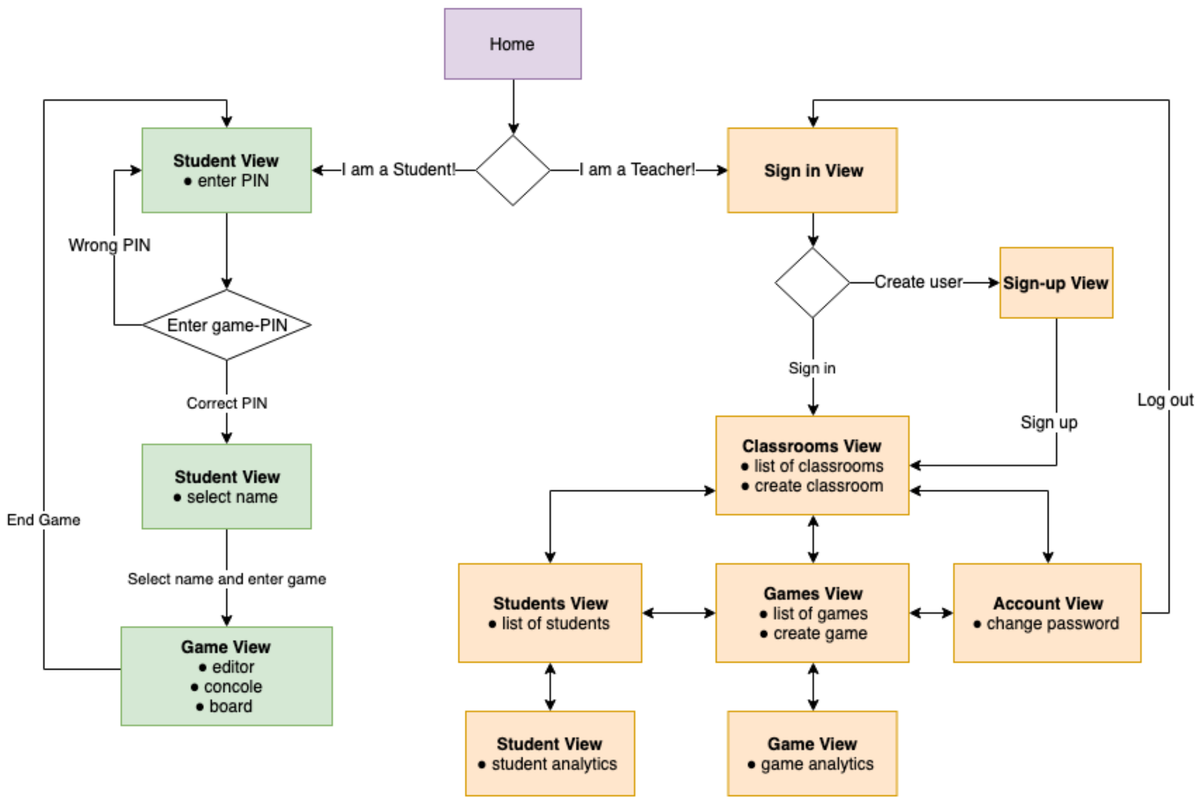


Figure 6.2: Graph showing the navigation paths of Code Flip.

validation and rules would be required for the UI, for ensuring the definition of correct and exhaustive tests for each task. This feature would be useful to include in a future version of the game.

6.2.2 Pictures

The pictures of the game are stored in 16 pieces, one for each task card of the board. When a task is solved, its card is swapped with its corresponding picture piece. The reason for not storing the picture in one piece and hiding it behind the board, is to avoid the ability to download the whole picture to see what it displays without having to solve any tasks. The pictures provided with the game have been manually sliced and uploaded to the picture storage. However, if a user interface for adding tasks to the game is created, functionality for adding new pictures should also be included.

6.3 Game Description

The game is formed as a web application, which is divided into two parts - student and teacher. The actual game resides in the student part, while the teacher part has functionality for creating games and seeing analytics about the performance of students. Figure 6.2 gives an overall representation of the navigation paths of Code Flip. The home page of the web application is shown in Figure 6.3. It displays two buttons, *"I am a student!"* and *"I am a teacher!"*, leading to the two different parts of the application.

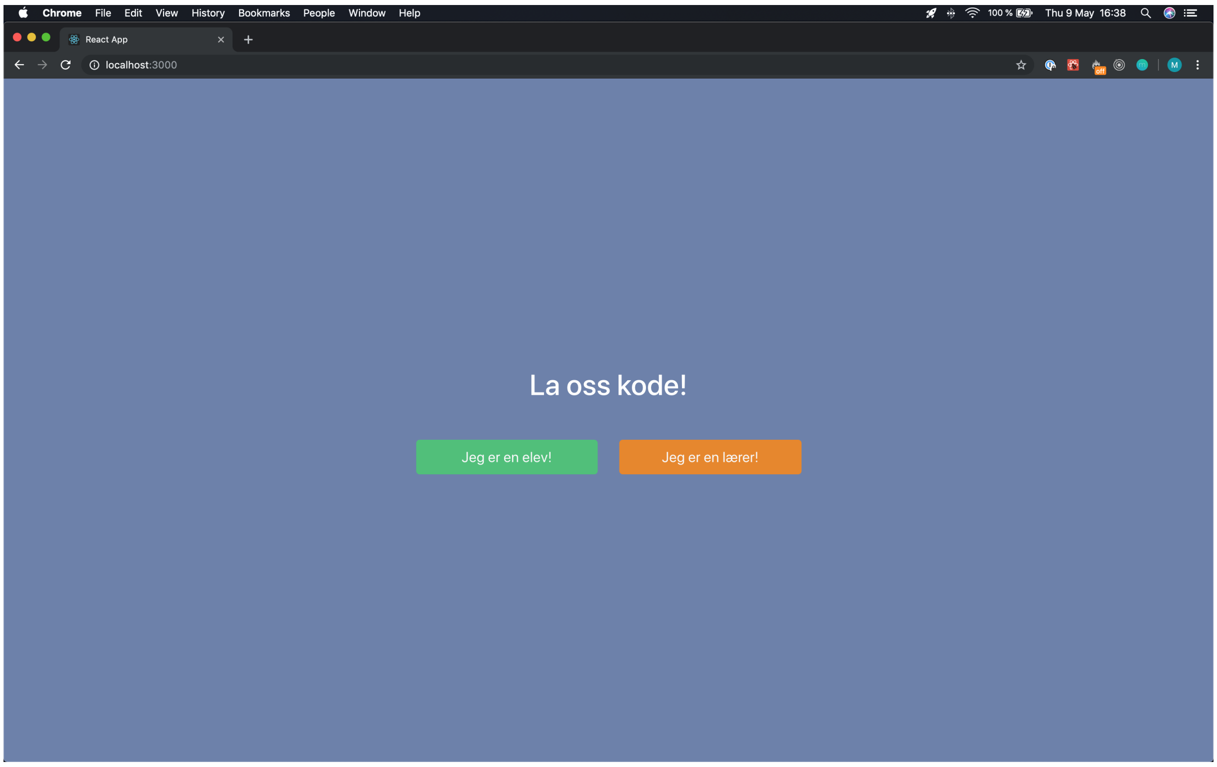


Figure 6.3: Home page of the game.

6.3.1 Joining the Game as a Student

The student part of the web application contains the actual game. It has two different views - one is for joining a game, and the other is the actual game. Since privacy and GDPR was a concern with the teachers, see Chapter 3, the authors decide to not require any login for the student part, and make the way of joining a game similar to that of Kahoot![4]. This means that the students only need a game-PIN to play the game.

Figures 6.4 and 6.5 show the procedure for joining a game. The player is presented with a form for submitting a game-PIN. If a valid PIN is entered, a drop-down menu is displayed, containing the names of each student of a class. These names have been provided in advance by the teacher. As a player selects her name and hits *Enter*, she has managed to join the game. Her name will then disappear from the drop-down, so that other players cannot choose the same name.

6.3.2 Playing the Game

After a player has entered a game, she encounters the game view. This view has two states, depending on whether the teacher has started the game or not. Common for both game states is that the player has the option to exit the game, through an *Exit game*-button in the top-right corner. An unstarted game does not contain any game elements. It only displays an explanatory text - *"Waiting for the game to start"*.

[4]`https://kahoot.com/`

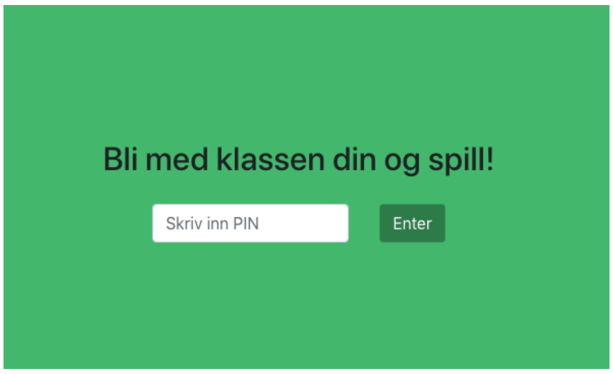


Figure 6.4: Joining game: enter game-PIN.

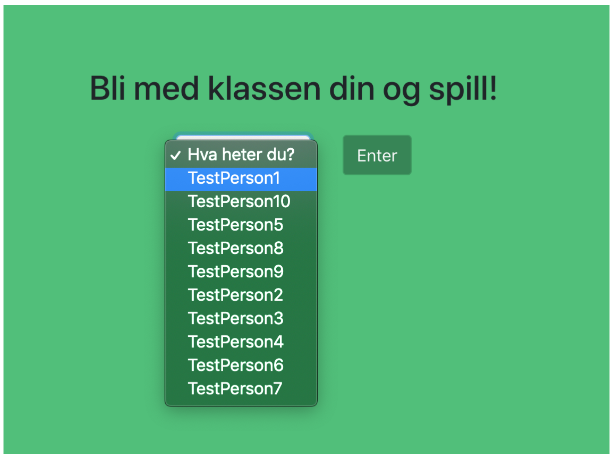


Figure 6.5: Joining game: select name.

The Playable Elements

As soon as the teacher starts the game, the playable game elements are made visible, as shown in Figure 6.6. These elements are an editor and a game board. The game board consists of sixteen cards covering a hidden picture, where each card contains a programming task that the student needs to solve to remove its card from the board. The task of each card is not visible until the card is opened, which is done by clicking on it. Figure 6.7 shows an open task-card. It covers the whole game board, and displays the task name, description and a button for closing the card. If the task contains a code skeleton, this is displayed in the editor.

Teams

The players of the game are separated into teams that are predefined by the teacher. Team members play on separate computers, but the game board for all members of the same team is identical, and changes made to the board by one player are also displayed on the board of her team members. This means that if a player selects a task, her team-mates will not be able to select the same task, as it becomes disabled. A disabled card has a dark grey color, and is not clickable.

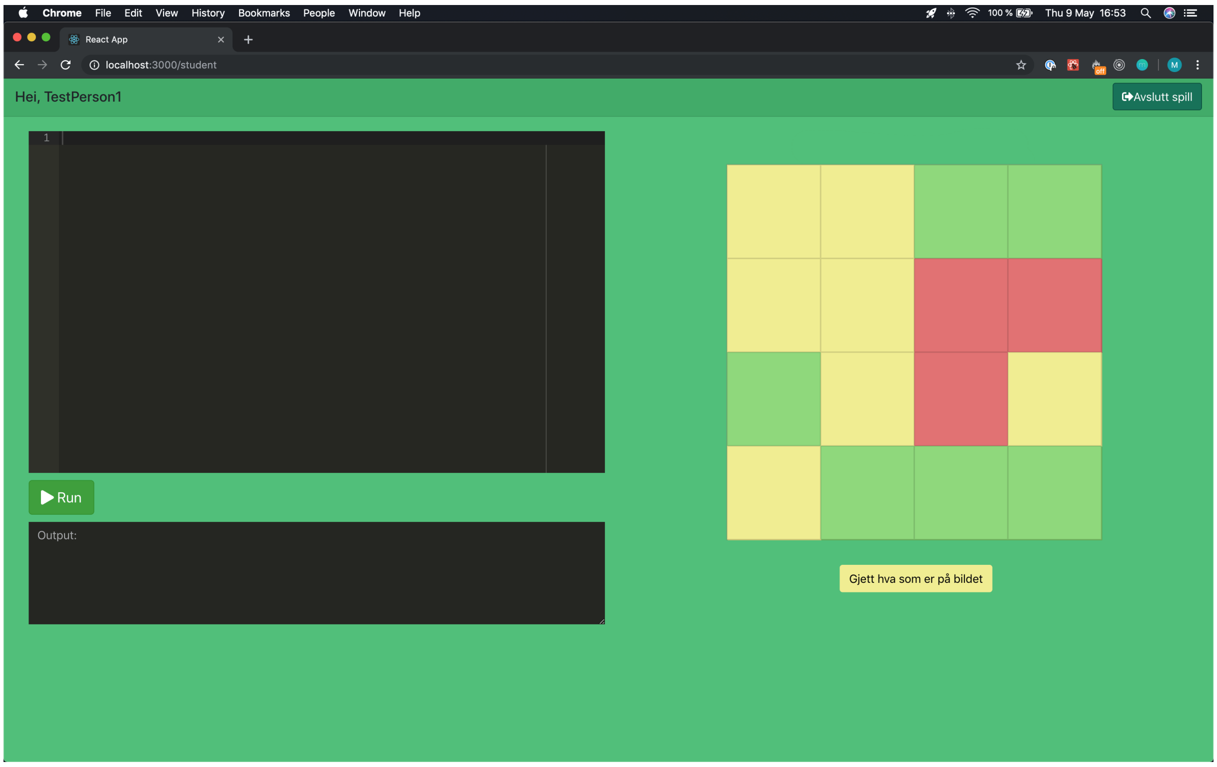

Chrome File Edit View History Bookmarks People Window Help
React App
localhost:3000/student
Hei, TestPerson1
Avslutt spill
Run
Output:
Gjett hva som er på bildet

Figure 6.6: Active game view showing the different game elements - editor on the left, board on the right, and the guess-button under the board.

Solving Tasks

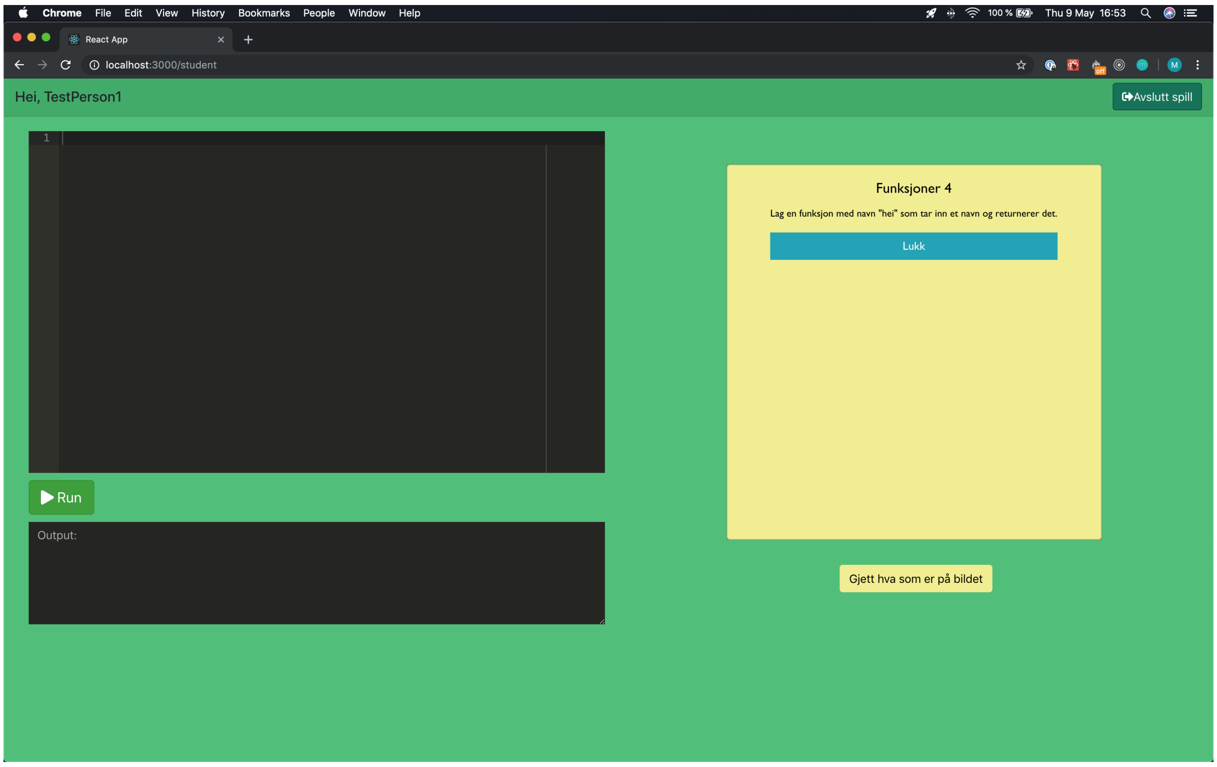
When a player has opened a task-card, she can solve the task by writing code in the editor. To submit an attempt, she presses the *Run*-button to run the code. The code is then sent to the Python evaluation service, where it will be executed together with the predefined tests to check whether the solution is correct. If a test fails, or the code causes an error or exception, the player is presented with a modal containing an explanatory error message. The error message is either provided directly by Python, translated to Norwegian in the evaluation service, or a hint that is predefined for that task.

An example of such a hint is shown in Figure 6.8. In the example, the player is to create a function called **name**, which takes a name and returns it. The player submits code with a function that instead returns a string, and therefore gets the error message *"Does your function return the name it takes as parameter?"*.

If all tests succeed and the code causes no errors, a modal is displayed with the text *"Good work! You solved the task :D"*, see Figure 6.9. When closing the modal, the board appears again, but now the task-card has been removed from the board, and a part of the hidden picture is visible. This is shown in Figure 6.10.

Guessing the Picture

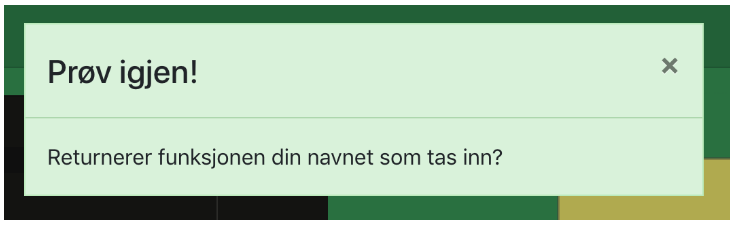
Behind the board is a hidden picture, which is gradually uncovered as the students solve tasks. As already mentioned, the same board is displayed on the screens of all members of a team. This means that when a player correctly solves a task, uncovering a part of the hidden picture, all team-members will see this change on their screen.



Funksjoner 4

Lag en funksjon med navn "hei" som tar inn et navn og returnerer det.

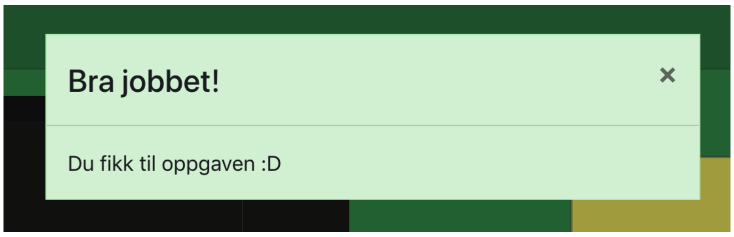
Figure 6.7: Player has selected a task-card.



Prøv igjen!

Returnerer funksjonen din navnet som tas inn?

Figure 6.8: Player has solved a task wrongly.



Bra jobbet!

Du fikk til oppgaven :D

Figure 6.9: Player has correctly solved a task.

A button is placed below the board with the text *"Guess what is on the picture"*. When a player thinks she knows what the picture shows, she clicks on this button, and a modal with a form for guessing the picture is displayed. If the guess is correct, the team wins the game. A modal appears on the screen of all players, stating the winner team and the picture solution.

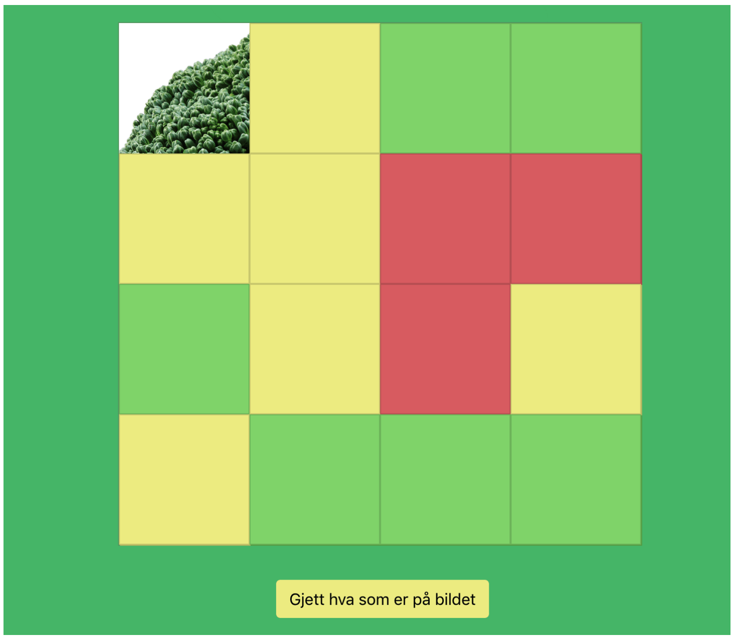


Figure 6.10: A piece of the picture has been uncovered.

Helping Each Other

In the game concepts chapter (Chapter 5.1), all proposed game concepts are described with functionality for helping each other through the game. The authors think it is cumbersome to force students to help each other through the game, when they can easily cooperate without this functionality. The students can instead be placed together with their team in the classroom while playing. This allows them to easily discuss both how to solve tasks, and what is displayed on the hidden picture. It is therefore decided to wait with this functionality, and rather include it later on if the evaluations show a need for it.

6.3.3 Logging in to the Teacher Part

To enter the teacher part of the web application, authentication is required. This means that a user has to be created when entering for the first time. If a teacher already has a user, she can log in through the login-view, as in Figure 6.11. After logging in, the account view appears, with functionality for changing password. In the future, this could include more account details, and the ability to delete the account. The teacher view has a navigation bar displaying the things a teacher can manage through the view: account, games, classrooms, students, and a logout button.

6.3.4 Creating Classrooms and Games

Before the game can be played, a teacher has to create a classroom and a game connected to that classroom.

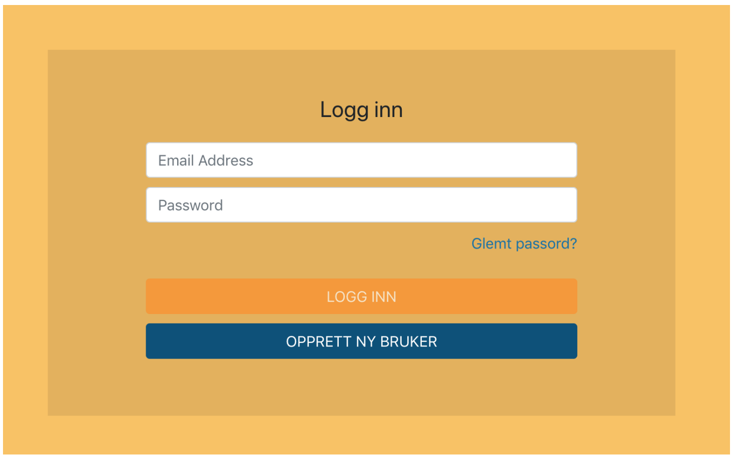


Figure 6.11: Logging in as a teacher.

Managing Classrooms

In the view for managing classrooms, a teacher can create and view her classrooms, as shown in Figure 6.12. Classrooms are created with a name, and a list of all the student names of a class, so that the correct names show up in the drop down menu when students enter a game. The reason for using a predefined list of names instead of letting the students type their own names, is that game analytics can be connected to a student across games, without requiring authentication for joining. The names need to be unique across all of the teacher's classrooms. This is further explained in Section 6.3.7.

When a classroom is created, it appears in the classrooms list on the left. If the teacher clicks on a classroom in the list, a modal showing the student names of the corresponding classroom appears, as is seen in Figure 6.13. The modal also includes a button for deleting the classroom.

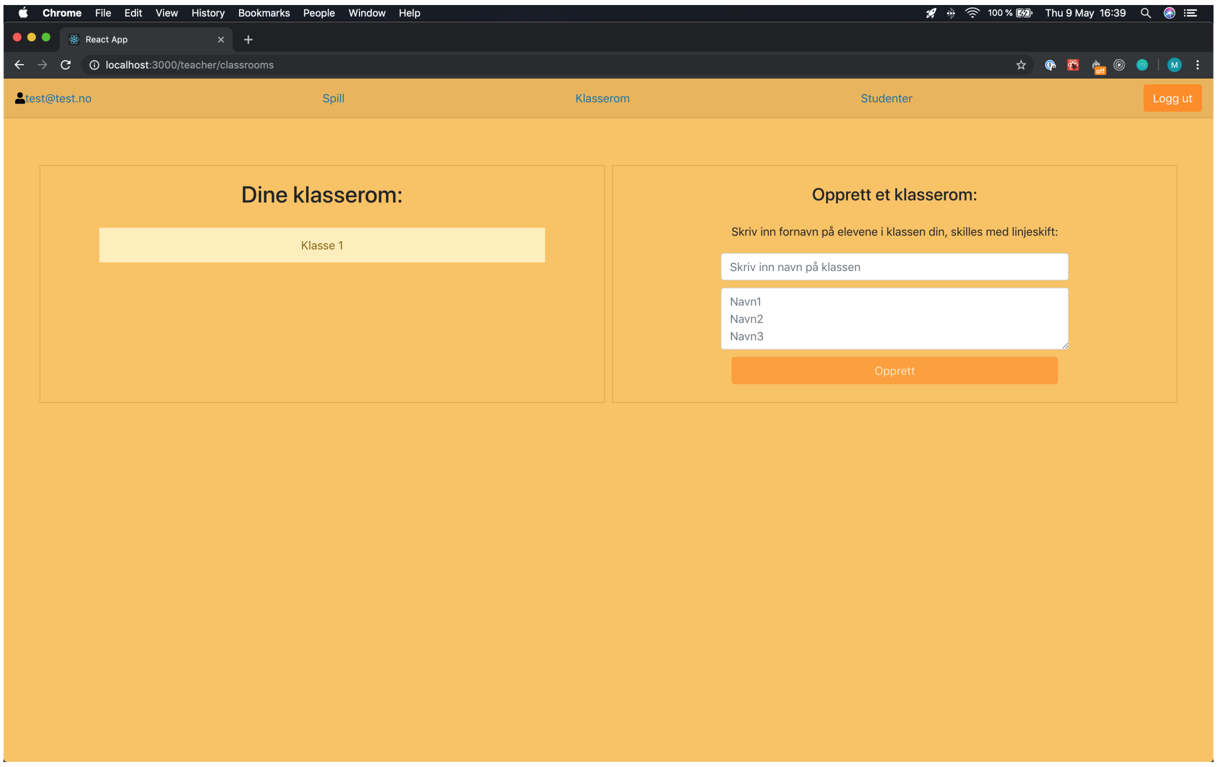
Managing Games

The view for managing games allows for creation of new games, given that a classroom already exists for that user. It contains a list of a teacher's existing games, and a form for creating a new game, as is shown in Figure 6.14. The form contains three options: selecting the classroom to use for the game, selecting the amount of teams, and selecting the secret picture. When a teacher has selected a classroom and amount of teams, a grid with the student names will appear. An explanatory text states: *"Divide into teams by clicking on a player, so that the color of the box changes"*. Figure 6.15 shows the state when a teacher is in the process of grouping the students in teams for creating a game.

When clicking on the *Create game*-button, the new game is created, and it appears in the game-list. The games in the list are mapped with colors corresponding to the game's status. Yellow represents a game that is ready to play, blue represents a game that is being played, while green represents a finished game.

6.3.5 Starting a Game

If the teacher clicks on a game that is ready to be played, she enters the game view. This view shows the game-PIN, and a button for starting the game. Before the game has started, the view displays the names of students as they enter the game, in boxes with colors corresponding to



Chrome File Edit View History Bookmarks People Window Help
Thu 9 May 16:39
React App
localhost:3000/teacher/classrooms
test@test.no
Spill
Klasserom
Studenter
Logg ut

Dine klasserom:

Klasse 1

Opprett et klasserom:

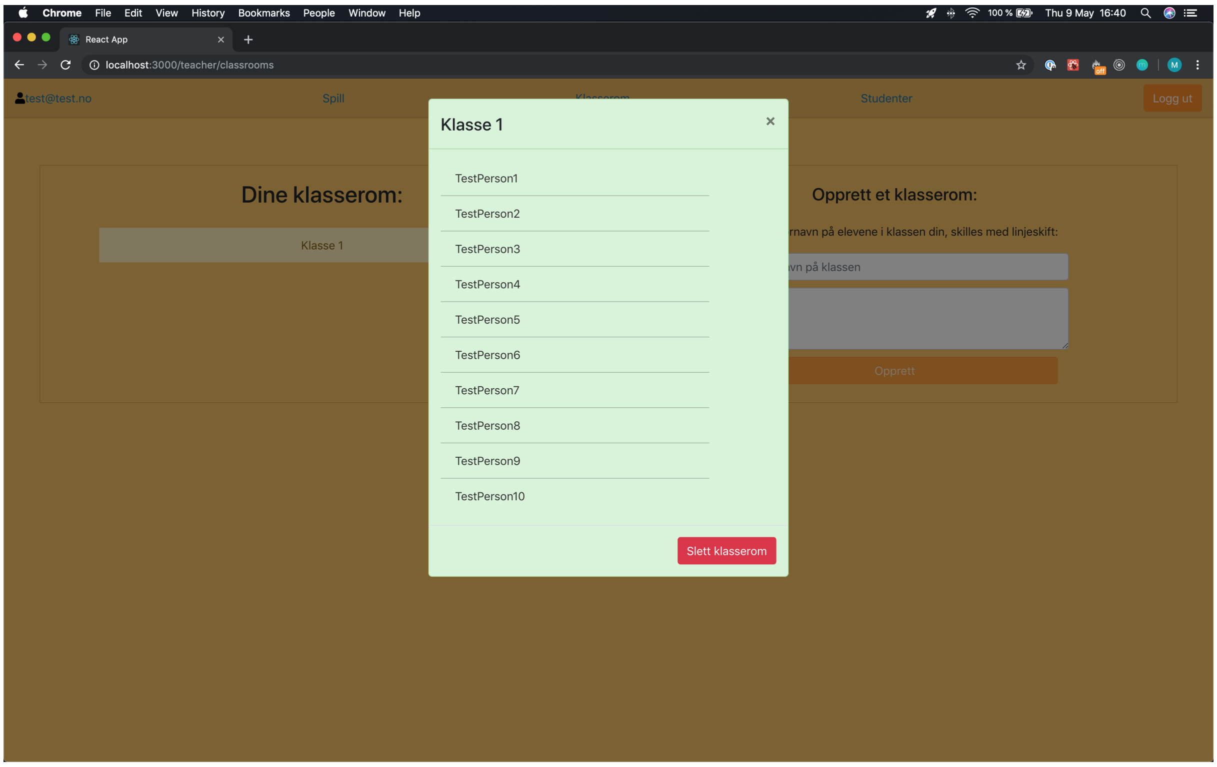
Skriv inn fornavn på elevene i klassen din, skilles med linjeskift:

Skriv inn navn på klassen

Navn1
Navn2
Navn3

Opprett

Figure 6.12: Managing classrooms: list of the teacher's existing classrooms, and form for creating a new classroom.



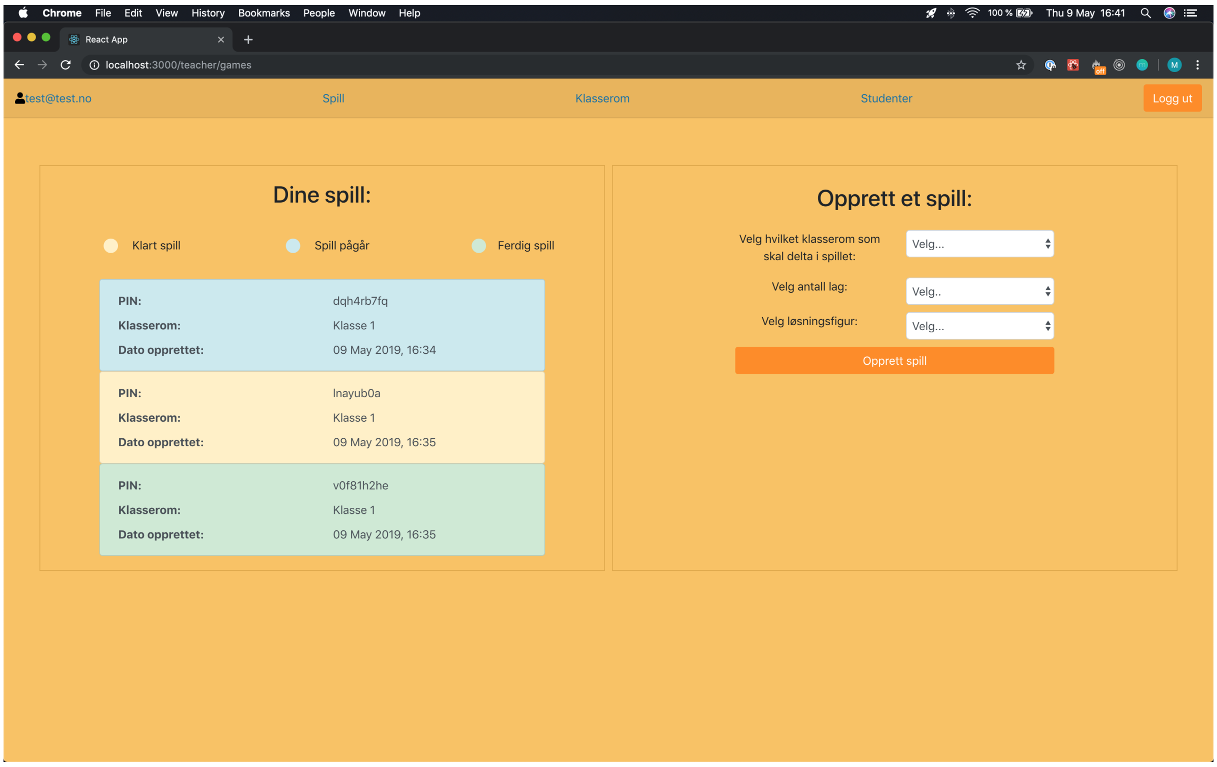
Chrome File Edit View History Bookmarks People Window Help
Thu 9 May 16:40
React App
localhost:3000/teacher/classrooms

Klasse 1

TestPerson1
TestPerson2
TestPerson3
TestPerson4
TestPerson5
TestPerson6
TestPerson7
TestPerson8
TestPerson9
TestPerson10

Slett klasserom

Figure 6.13: Managing classrooms: Detailed view of a specific classroom.


Chrome File Edit View History Bookmarks People Window Help
React App
localhost:3000/teacher/games

test@test.no Spill Klasserom Studenter Logg ut

Dine spill:

● Klart spill ● Spill pågår ● Ferdig spill

PIN:	dqh4rb7fq
Klasserom:	Klasse 1
Dato opprettet:	09 May 2019, 16:34
PIN:	lnayub0a
Klasserom:	Klasse 1
Dato opprettet:	09 May 2019, 16:35
PIN:	v0f81h2he
Klasserom:	Klasse 1
Dato opprettet:	09 May 2019, 16:35

Opprett et spill:

Velg hvilket klasserom som skal delta i spillet:	Velg...
Velg antall lag:	Velg..
Velg løsningsfigur:	Velg...

Opprett spill

Figure 6.14: Managing games: List of the teacher's existing games, and form for creating a new game.

their teams. This is shown in Figure 6.16. When all players have entered the game, the teacher can start the game by clicking the *Start game*-button. The boxes of each player will now include a list of tasks, which is empty to begin with. As soon as the players start choosing tasks from the board, they will appear in the list, as shown in Figure 6.17.

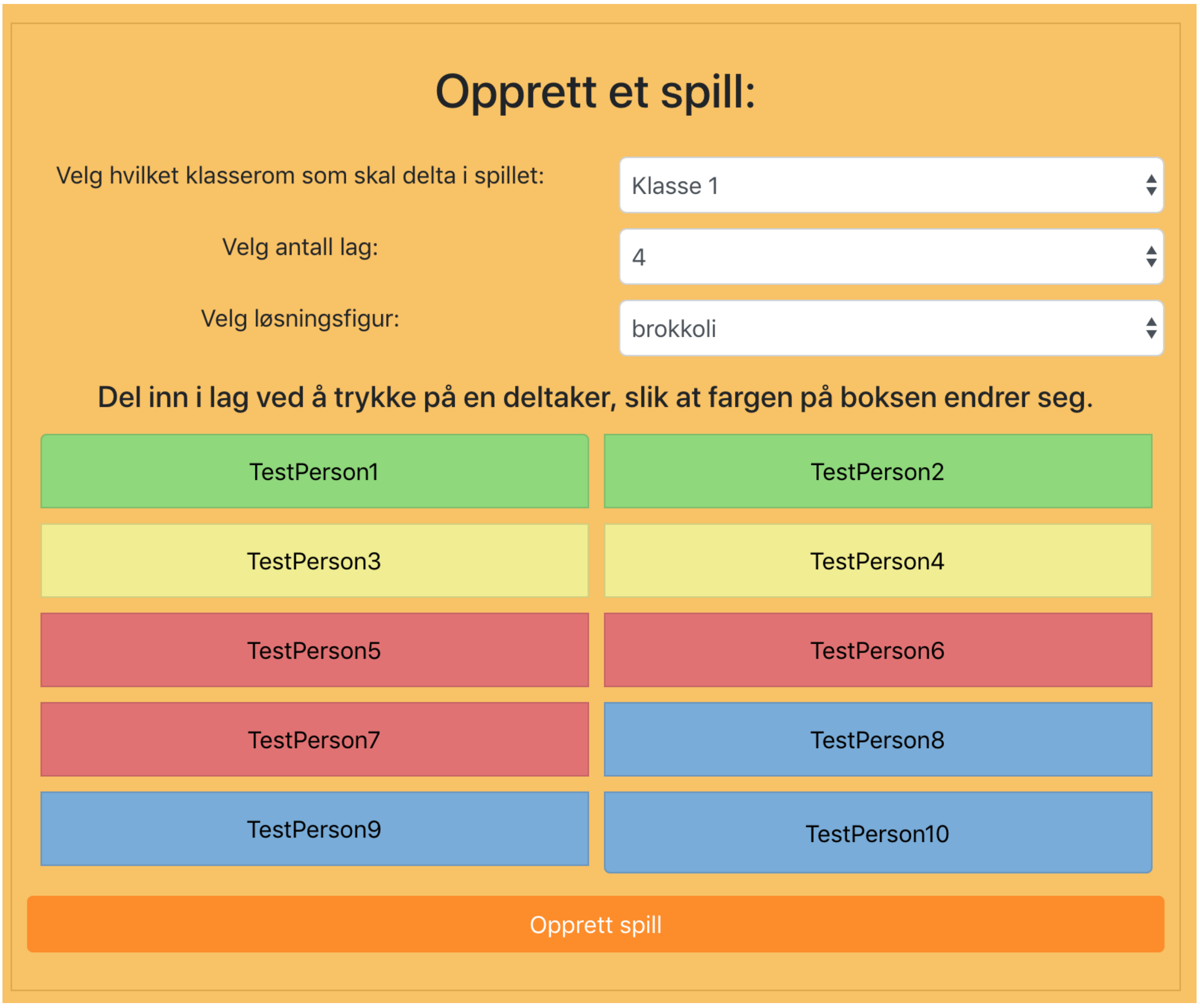
6.3.6 Analytics During Game-Play

The box of each student containing a list of tasks, is how the teacher can view analytics during game-play. Each task in the task-list is presented with its title, difficulty and a time-stamp. Next to each time-stamp, is either a check mark or a clock icon. The check mark icon symbolizes solved tasks, hence the time-stamp next to it shows the time spent solving the task. The clock icon symbolizes tasks that are in progress, and hence the corresponding time-stamp displays the time of starting the task.

By clicking on tasks, a modal with details about the task appears. As shown in Figure 6.18, these details include the task title, difficulty-level, time spent, and the student's solution code. If a student has not been able to solve the task, the last code she tried to execute is saved in the database. This means that if a player opens a task, and exits without trying to solve it, the solution code displayed to the teacher view will be empty.

6.3.7 Analytics After Game-Play

By navigating to *Students* in the navigation bar, the teacher can see an overview of all her classrooms, and the students that are part of it, as seen in Figure 6.19. When clicking on a box of a student, a detailed view of the tasks the student has solved is displayed. Figure 6.20 shows



Opprett et spill:

Velg hvilket klasserom som skal delta i spillet: Klasse 1

Velg antall lag: 4

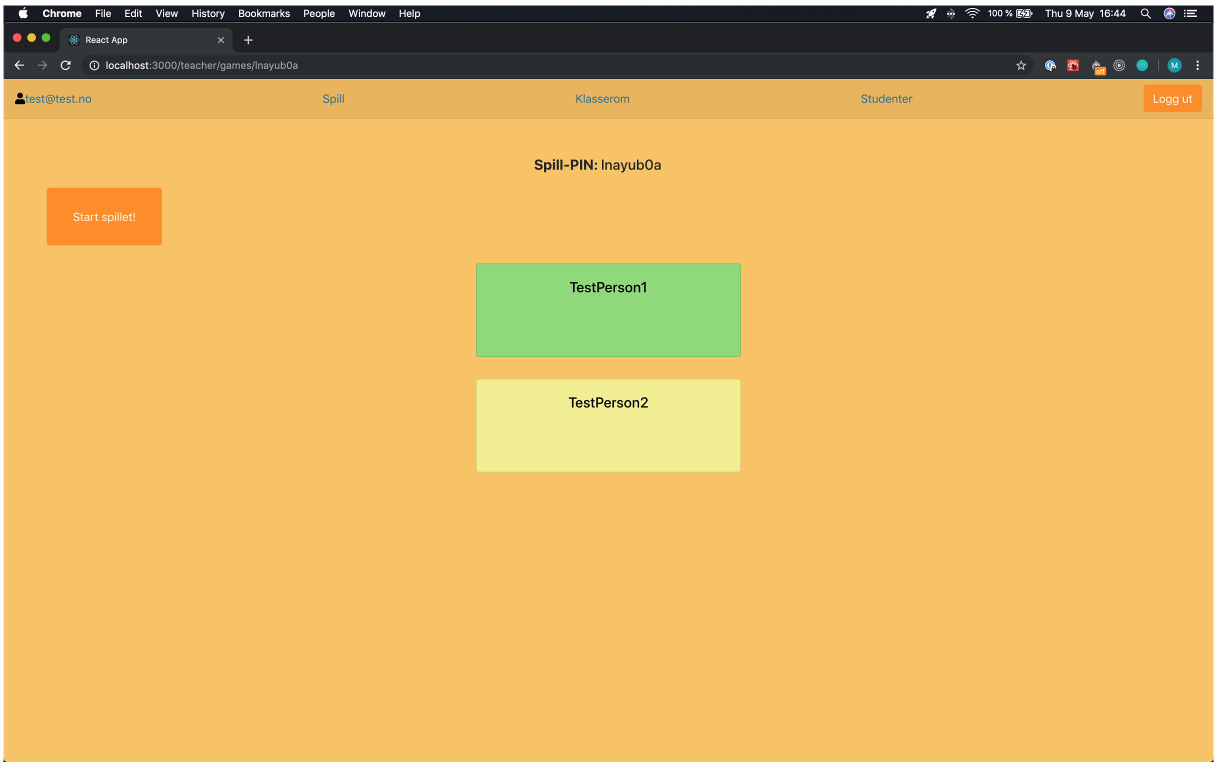
Velg løsningsfigur: brokkoli

Del inn i lag ved å trykke på en deltaker, slik at fargen på boksen endrer seg.

🟩 TestPerson1	🟩 TestPerson2
🟨 TestPerson3	🟨 TestPerson4
🟥 TestPerson5	🟥 TestPerson6
🟥 TestPerson7	🟦 TestPerson8
🟦 TestPerson9	🟦 TestPerson10

Figure 6.15: Managing games: The process of creating a game.

this. This view is the reason for requiring that the teacher defines unique names across all her classrooms. The web application goes through all of the teacher's games, and fetches the tasks connected to the name of the student. The solutions and attempts of each task is then presented with the task-title, level of difficulty, game date, time spent solving the task, task description, and the Python code written by the student.

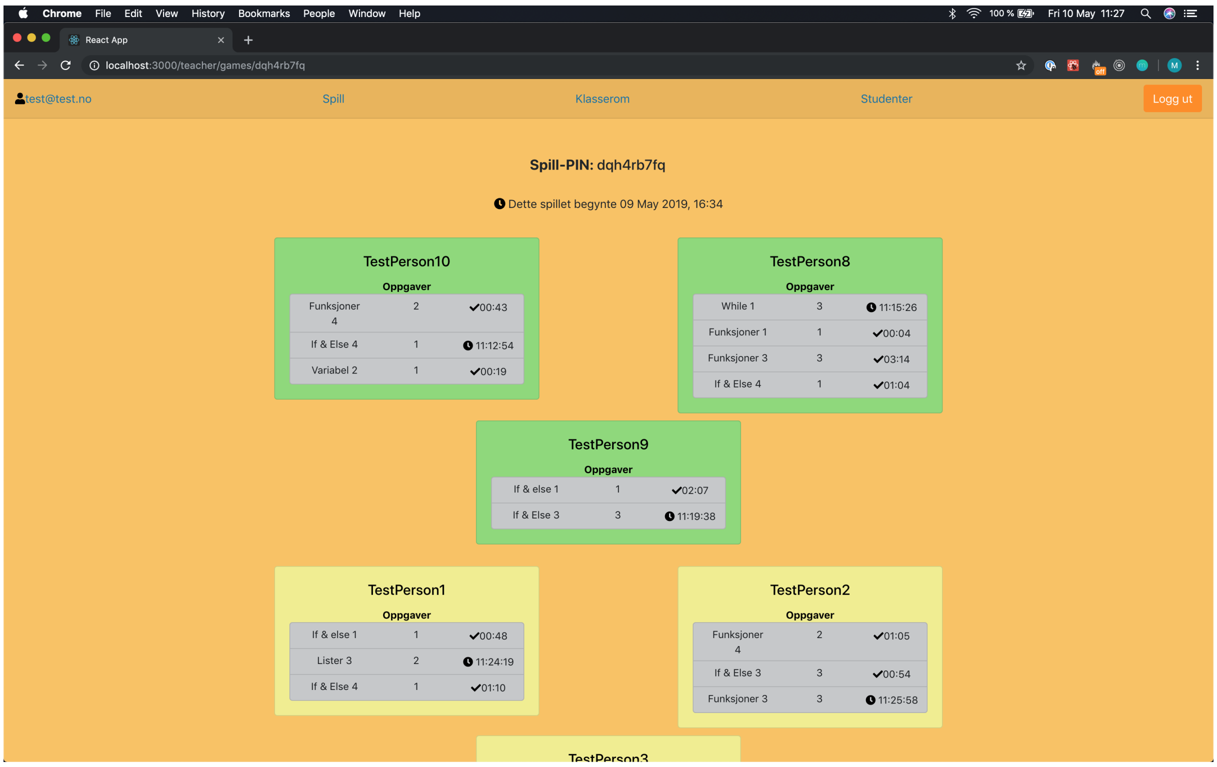

Chrome File Edit View History Bookmarks People Window Help
Thu 9 May 16:44
React App
localhost:3000/teacher/games/lnayub0a
test@test.no
Spill
Klasserom
Studenter
Logg ut
Start spillet!

Spill-PIN: lnayub0a

TestPerson1

TestPerson2

Figure 6.16: Game view: Unstarted game when two players have joined.


Chrome File Edit View History Bookmarks People Window Help
Fri 10 May 11:27
React App
localhost:3000/teacher/games/dqh4rb7fq
test@test.no
Spill
Klasserom
Studenter
Logg ut

Spill-PIN: dqh4rb7fq

Dette spillet begynte 09 May 2019, 16:34

TestPerson10

Oppgaver

Funksjoner 4	2	✓00:43
If & Else 4	1	11:12:54
Variabel 2	1	✓00:19

TestPerson8

Oppgaver

While 1	3	11:15:26
Funksjoner 1	1	✓00:04
Funksjoner 3	3	✓03:14
If & Else 4	1	✓01:04

TestPerson9

Oppgaver

If & else 1	1	✓02:07
If & Else 3	3	11:19:38

TestPerson1

Oppgaver

If & else 1	1	✓00:48
Lister 3	2	11:24:19
If & Else 4	1	✓01:10

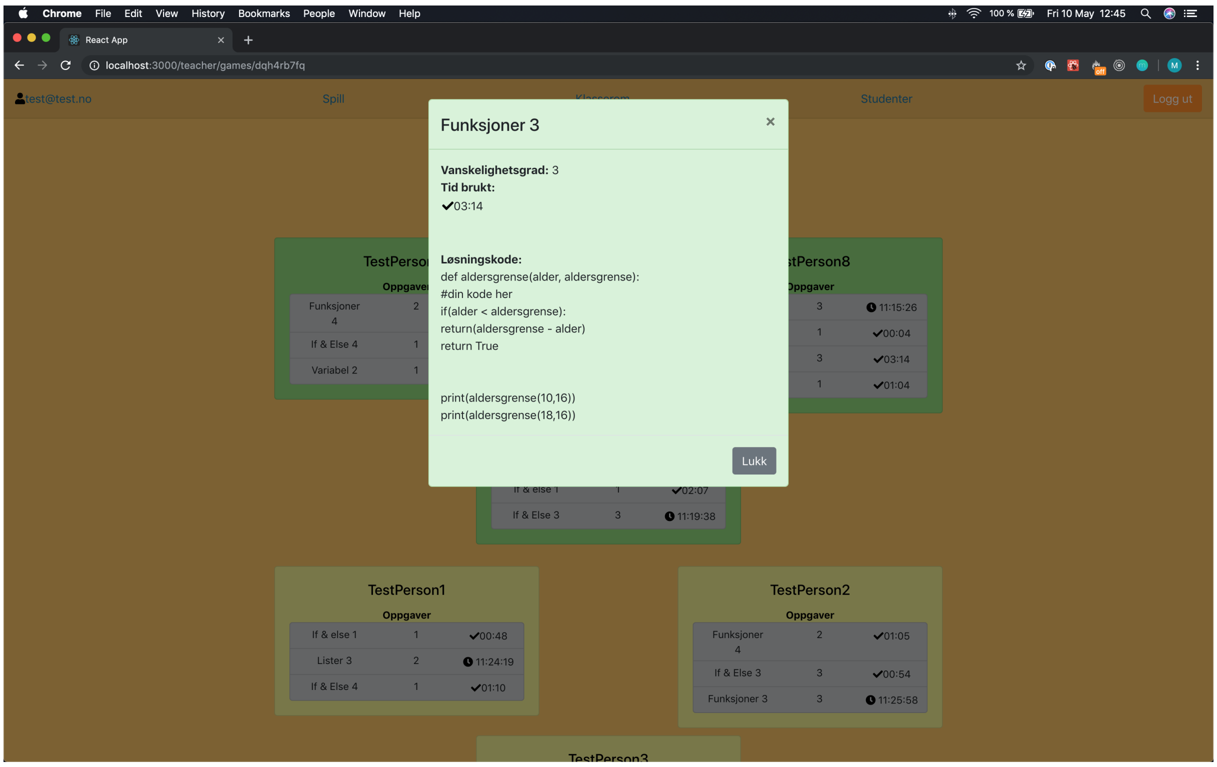
TestPerson2

Oppgaver

Funksjoner 4	2	✓01:05
If & Else 3	3	✓00:54
Funksjoner 3	3	11:25:58

TestPerson3

Figure 6.17: Game view: Students have started to solve tasks.



Funksjoner 3

Vanskelighetsgrad: 3
Tid brukt:
✓03:14

Løsningskode:

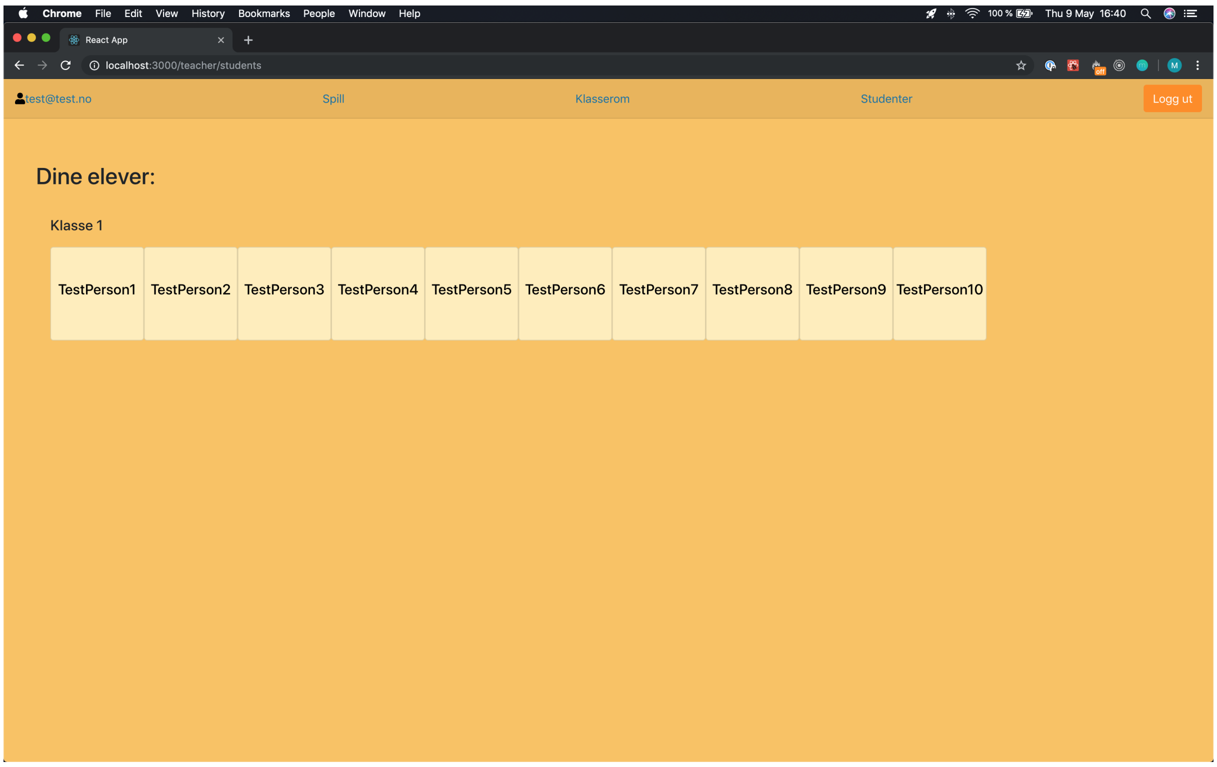
```
def aldersgrense(alder, aldersgrense):
#din kode her
if(alder < aldersgrense):
return(aldersgrense - alder)
return True


print(aldersgrense(10,16))
print(aldersgrense(18,16))
```

TestPerson1		
If & else 1	1	✓00:48
Lister 3	2	11:24:19
If & Else 4	1	✓01:10

TestPerson2		
Funksjoner 4	2	✓01:05
If & Else 3	3	✓00:54
Funksjoner 3	3	11:25:58

Figure 6.18: Game view: Inspecting the task of a student.

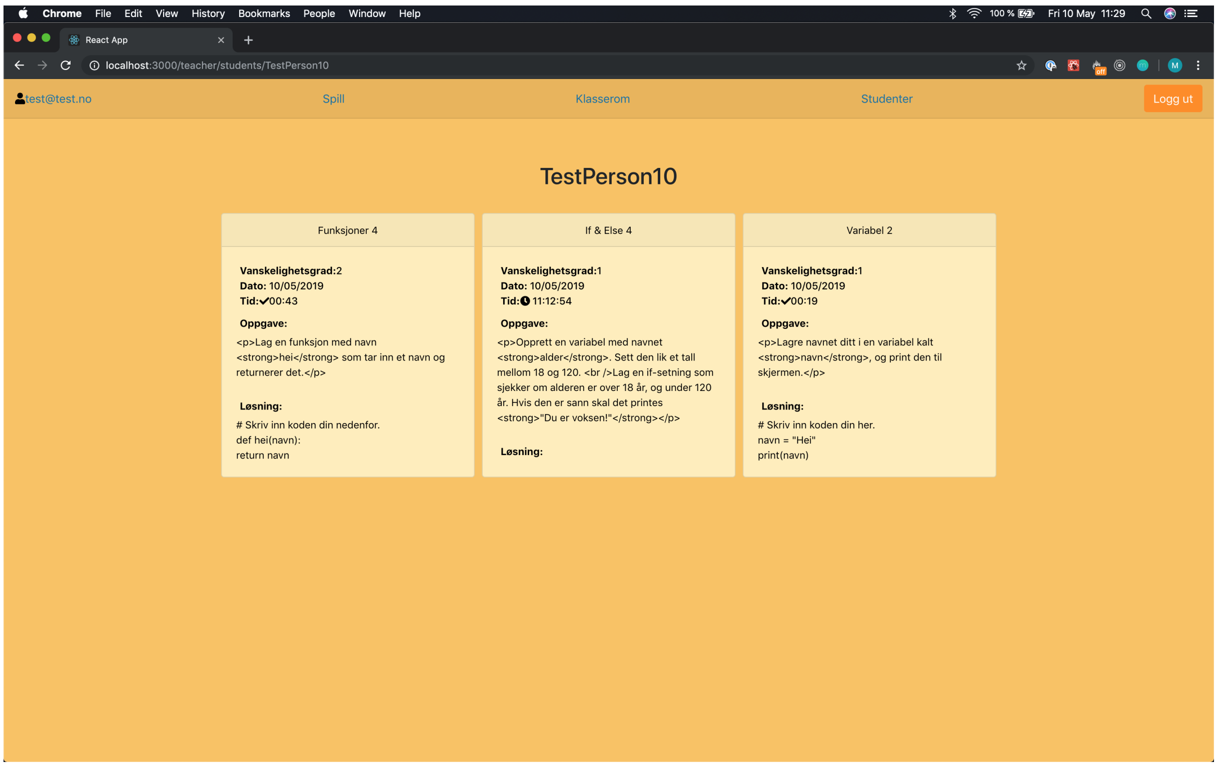


Dine elever:

Klasse 1

TestPerson1 TestPerson2 TestPerson3 TestPerson4 TestPerson5 TestPerson6 TestPerson7 TestPerson8 TestPerson9 TestPerson10

Figure 6.19: Overview of the teachers classrooms and its students.


Chrome File Edit View History Bookmarks People Window Help
Fri 10 May 11:29
React App
localhost:3000/teacher/students/TestPerson10
test@test.no
Spill
Klasserom
Studenter
Logg ut

TestPerson10

Funksjoner 4

Vanskelighetsgrad:2
Dato: 10/05/2019
Tid:✓00:43

Oppgave:

<p>Lag en funksjon med navn <strong>hei</strong> som tar inn et navn og returnerer det.</p>

Løsning:

```
# Skriv inn koden din nedenfor.
def hei(navn):
return navn
```

If & Else 4

Vanskelighetsgrad:1
Dato: 10/05/2019
Tid:🕓 11:12:54

Oppgave:

<p>Opprett en variabel med navnet <strong>alder</strong>. Sett den lik et tall mellom 18 og 120. <br />Lag en if-setning som sjekker om alderen er over 18 år, og under 120 år. Hvis den er sann skal det printes <strong>"Du er voksen!"</strong></p>

Løsning:

Variabel 2

Vanskelighetsgrad:1
Dato: 10/05/2019
Tid:✓00:19

Oppgave:

<p>Lagre navnet ditt i en variabel kalt <strong>navn</strong>, og print den til skjermen.</p>

Løsning:

```
# Skriv inn koden din her.
navn = "Hei"
print(navn)
```

Figure 6.20: Overview of all tasks solved or attempted by a specific student.

Chapter 7

Technical Description

This chapter gives a technical description of Code Flip. Before diving into the implementation, a foundation had to be laid for the design. The chapter therefore starts by describing how the architecture was chosen, and the technologies used for the implementation. Finally, the last section covers the game development, including methodology and implementation.

7.1 Game Architecture

When designing the architecture of the game, the authors emphasized that it should be lightweight and easy to maintain, and that it should allow for use of modern technology. This is both due to the time-constraints for the project, and that new technology that is widely used in the industry offers a lot of helpful documentation and resources. Such technologies are also constantly maintained and improved in order to offer the best to its users, and compete with similar technologies on the market. The following sections describe the architecture choices.

7.1.1 Server-less

With the arise of cloud providers, came a new type of architecture called server-less (Bashir 2018). A server-less architecture lets application developers fully focus on client side logic, which makes this architecture fitting for implementing a game in a short amount of time. For everything else, the developers can rely solely on a combination of third-party services. Cloud providers dynamically manage all allocation and provisioning of servers, and are responsible for the security and performance of the services that they offer. Developers only pay for the resources that are used by their application. This pay-as-you-go scheme suits this project well, as many cloud providers offer free plans to those who do not need a lot of resources.

Figure 7.1 shows the server-less architecture of the game, and how it communicates with different third-party services for authentication, real-time data storing, storing of static images and a micro-service for evaluating the Python code submitted by players of the game. The direction of the arrows indicate the direction in which the communication is initiated. The communication between the game and the real-time database can be initiated from both sides - user actions updates the database, and data changes updates the game UI.

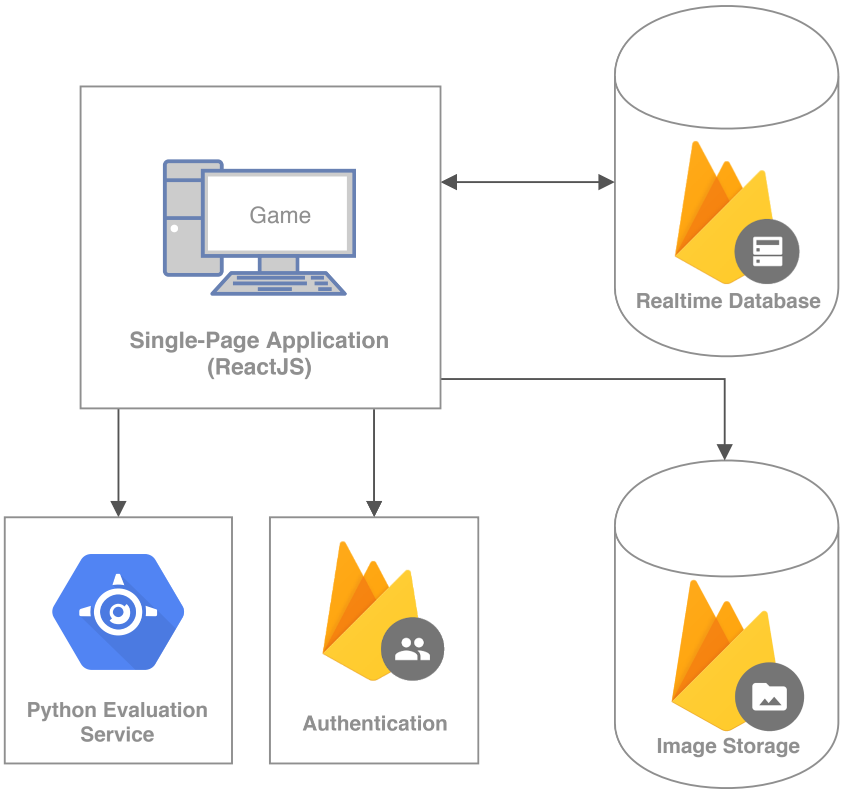


Figure 7.1: Deployment diagram showing the server-less architecture of the game.

7.1.2 Single-page

For the client-side of the application, the authors decided to go with a single-page architecture. Once again, their reason for this is that it is simple to use and maintain, and that the most widely used web libraries of 2019 - React[1], Angular[2], Vue[3] - are built on this architecture (Neagoie 2018). Single-page applications are web-sites that automatically update their content in response to navigation actions or data changes (Sherman 2018). Only the parts of the web-site that are related to the change are updated, instead of reloading the whole page to display new content. This way, a single-page application ensures a smooth user experience, which is crucial for a multi-player game like this one.

The authors decided to not require any authentication for players of the game, in order to ensure a low threshold for playing and to address the teachers concern of privacy and GDPR. For the teacher view of the game, authentication should be required to access data about the students, and to be able to create games. The application is therefore structured around these two parts - student and teacher. Each of these parts consists of several components, to divide the game logic into smaller parts. Figure 7.2 shows how the different components of the game communicate with each other, the Python evaluation service, and with third-party services.

[1]`https://reactjs.org/`
[2]`https://angular.io/`
[3]`https://vuejs.org/`

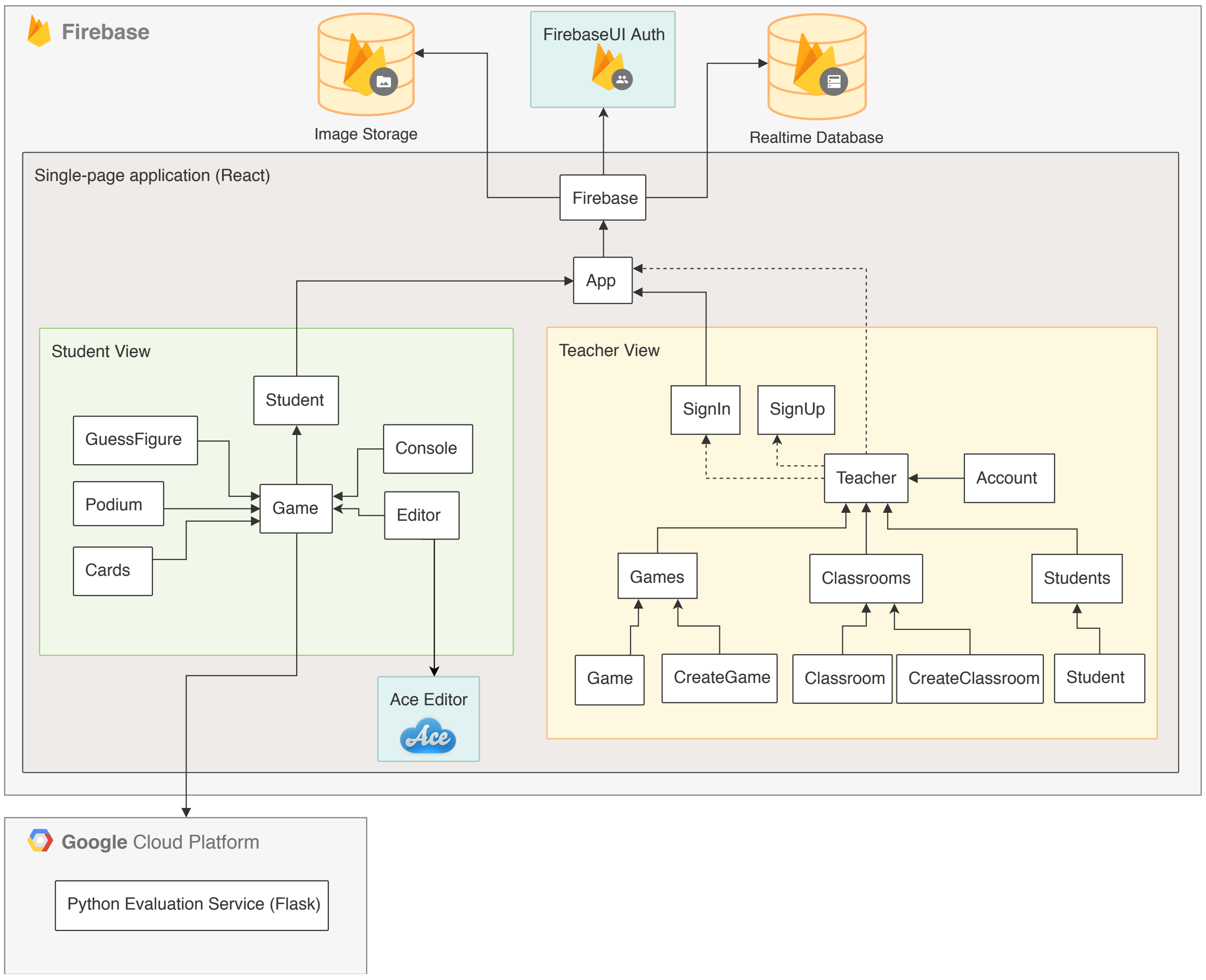


Figure 7.2: Component diagram showing how the components of the single-page application communicate with each other, and with other services.

7.1.3 Micro-service

When allowing for user input through a website, the input should normally be thoroughly validated and sanitized in order to avoid different types of injection attacks (Meucci and Muller 2019). For this game however, the user is supposed to submit executable Python code, and this input can therefore not be fully sanitized. Other measures must be taken in order to avoid that the code can harm file system or database, or exhaust resources of the game.

An effective way of protecting the game from the users' code, is to isolate the execution service from the rest of the application (Jackson 2017). This can for instance be done through using a sandbox or Jupyter notebooks. For this project, the simplest solution is to design the service as a micro-service, which suits this case well since it is supposed to do only one thing.

A micro-service architecture is a service-oriented architecture where an application consists of many small and independent services that communicate with each other (Martinez 2017). The services should be loosely coupled, even with its own data and own deployment, since they are to work independently of each other. In this case, the game in itself is not based on the micro-service architecture, but the evaluation service is.

7.2 Technology

This section describes the technologies used to implement the game and the evaluation service, according to the chosen architectures.

7.2.1 The Game

As mentioned, the game is developed as a single-page application, and the chosen technologies for this are React and Firebase.

React

Since one of the authors already had some experience with React, and it is amongst the most popular JavaScript libraries (Neagoie 2018), the authors decided to use this. React[4] is a JavaScript library for building user interfaces, maintained by Facebook. It is a component-based language, with a single-page architecture. Facebook provides simple tutorials for getting started, as well as up-to-date documentation that is easy to understand, which helped as one of the authors had not worked with the React library prior to this project. It is also heavily used in the industry, so there is a wide community and exist a vast amount of online resources for support.

Ace Editor

The Ace Editor[5] is an open-sourced code editor designed for being integrated into websites. It is used by web-services such as AWS, Khan Academy, and the Cloud9 IDE. It can be used as a component in React by importing a package called React-Ace. Through offering customization of features like syntax highlighting for specific programming languages, editor themes, and automatic indentation, the editor can be customized to appear similar to the code editors most commonly used with Python.

[4]`https://reactjs.org/`
[5]`https://ace.c9.io/`

Firebase: Realtime Database, Cloud Storage, Authentication

The selected provider of most of the required services was Firebase[6], a cloud platform owned by Google. Other platforms such as Amazon Web Services and Microsoft Azure were considered, however Firebase's core product is a realtime database which is ideal for a multiplayer game. Firebase also includes Authentication, Storage and Hosting services in its free pricing plan.

Firebase Realtime Database[7] is a cloud-hosted database, where data is synced across all clients in real-time. Data is stored in the JavaScript Object Notation (JSON). To make GET-requests to the Realtime Database, one creates an instance of it, and can then choose to retrieve data once, or to listen to changes to the element specified in the GET-request. This way, a JavaScript-element can be automatically updated when data changes.

Firebase Cloud Storage[8] is chosen for storing of static content like images, while the Firebase Authentication is used for authenticating users of the teacher view of the game.

7.2.2 The Evaluation Service

The technologies chosen for the Python evaluation micro-service, are Flask and Google App Engine.

Flask

Flask[9] is a web microframework for Python with the intention of being easy to set up and use, which is also why it was chosen. It is used for creating the API-endpoint for running code sent to the evaluation service. The framework comes with a built-in server, but this is only employed for local testing of the evaluation service as it does not scale well.

Python exec()

In order to execute Python code, the builtin function exec()[10] is used. The function simply executes the code it is provided, and it can therefore be harmful to the file-system if for instance the ”os”-module is imported. However, it is possible to restrict the availability of methods and variables through exec() in order to protect a system against harmful actions. What exec() does not handle is infinite loops. These exhaust a lot of processing power, and can make an application slow.

Google Cloud Platform: App Engine

In order to host the Python evaluation service, Google App Engine is used, as this is more compatible with the Flask framework than Firebase. As Firebase is owned by Google, the products of Google Cloud Platform and Firebase go well together. However, seeing that the evaluation service is designed as a micro-service, the chosen hosting platform does not have anything to do with the game. Flask provides its own documentation for hosting with Google App Engine, which facilitates the setup. An extra advantage of hosting the evaluation service

[6] `https://firebase.google.com/`
[7] `https://firebase.google.com/docs/database`
[8] `https://firebase.google.com/docs/storage`
[9] `http://flask.pocoo.org/`
[10] `https://programiz.com/python-programming/methods/built-in/exec`

on Google App Engine, is that it kills any resource stuck in an infinite loop after 60 seconds. Since it also handles scaling of the application, there will always be an available replica of the service.

7.3 Development

Seeing that a fully functioning prototype of Code Flip is the main contribution of this work, two months were set aside for developing it. This section describes the methodology used for the game development, as well as giving a brief description of how the initial version was implemented.

7.3.1 Methodology

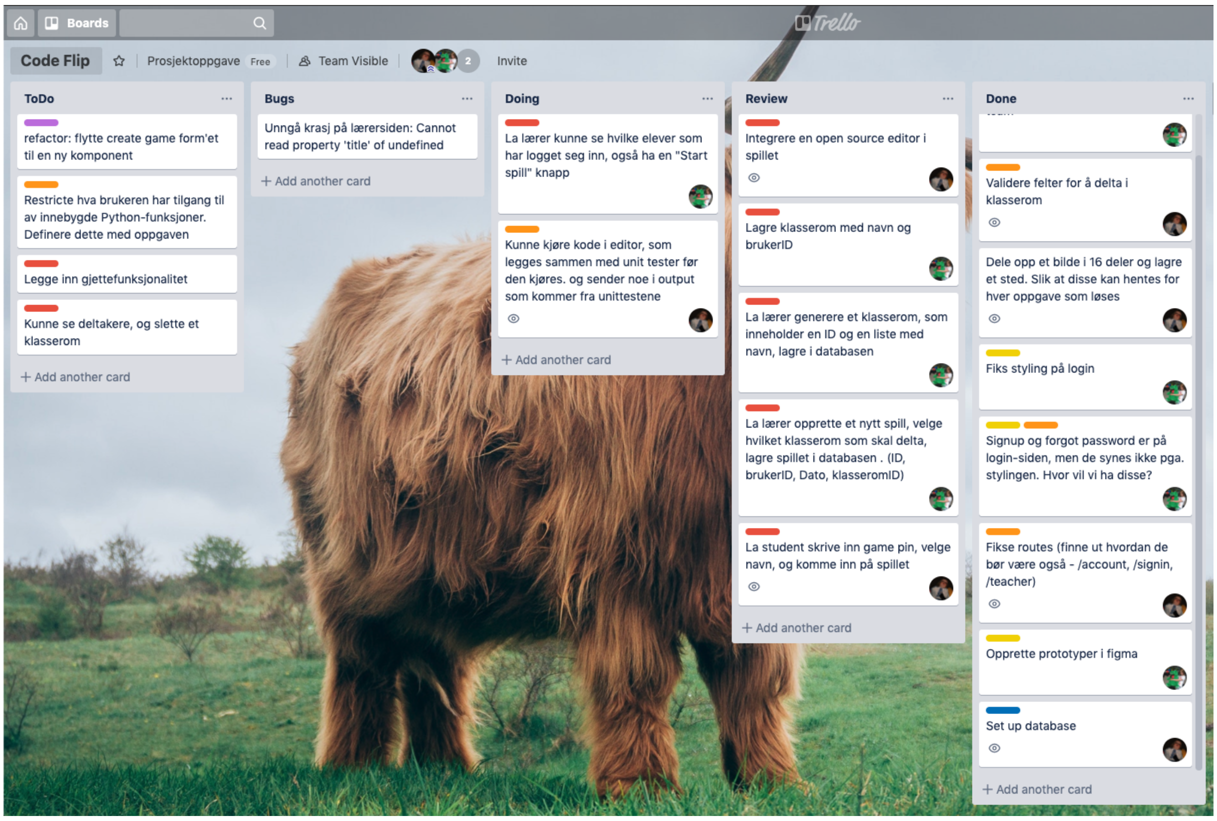
A list of requirements had already been formulated and refined several times to address the stakeholders of the game. However, the requirements are high-level, and needed to be divided into smaller and more specific tasks for implementation. The game was therefore developed in an agile way, in order to refine the tasks continuously, and assess whether the game still meets the requirements. Seeing that Code Flip was created by the initiative of the authors, it was natural to do this weekly assessment without including stakeholders.

Tools used to aid the development are Trello and git. Trello[11] visualizes a Kanban board, used to schedule and allocate tasks. Git was used for version control through an open repository on Github[12]. One or several times each week, the authors discussed the status of the development, and chose the features to focus on. The way these features should be implemented was discussed, before they were split up into smaller tasks and added to the ”ToDo”-list of the Trello board. Each author chose freely among the available tasks, and the tasks they chose were moved to the ”Doing”-list, and assigned to their Trello-user. For each task, a branch was created with git. When a task was done, it was placed in the ”Review”-list, and a pull request was created on Github, requiring the review of the other author. When reviewing the other's code, code quality and proper function of the implemented feature were emphasized. Prettier[13] was used for the JavaScript code to ensure consistent code formatting according to JavaScript standards. A task was moved from ”Review” to ”Done” when the other author had reviewed and merged it with the master branch. Figure 7.3 shows a screenshot of the Trello-board with the different lists indicating the status of tasks.

7.3.2 Implementation

The implementation of the web-application was initiated by setting up a React template, connecting it to a new real-time database in Firebase, and implementing Firebase's Authentication service for the app. A React template was easily created by using Facebook's tutorial for getting started with React[14]. Before implementing the Authentication service, the application was divided into two main components, teacher and student, since the Authentication service should

[11]`https://trello.com`
[12]`https://github.com/annikh/horseRace`
[13]`https://prettier.io/`
[14]`https://facebook.github.io/create-react-app/docs/getting-started`



ToDo

- refactor: flytte create game form'et til en ny komponent
- Restricte hva brukeren har tilgang til av innebygde Python-funksjoner. Definere dette med oppgaven
- Legge inn gjettefunksjonalitet
- Kunne se deltakere, og slette et klasserom
- + Add another card

Bugs

- Unngå krasj på lærersiden: Cannot read property 'title' of undefined
- + Add another card

Doing

- La lærer kunne se hvilke elever som har logget seg inn, også ha en "Start spill" knapp
- Kunne kjøre kode i editor, som legges sammen med unit tester før den kjøres. og sender noe i output som kommer fra unittestene
- + Add another card

Review

- Integrere en open source editor i spillet
- Lagre klasserom med navn og brukerID
- La lærer generere et klasserom, som inneholder en ID og en liste med navn, lagre i databasen
- La lærer opprette et nytt spill, velge hvilket klasserom som skal delta, lagre spillet i databasen . (ID, brukerID, Dato, klasseromID)
- La student skrive inn game pin, velge navn, og komme inn på spillet
- + Add another card

Done

- Validere felter for å delta i klasserom
- Dele opp et bilde i 16 deler og lagre et sted. Slik at disse kan hentes for hver oppgave som løses
- Fiks styling på login
- Signup og forgot password er på login-siden, men de synes ikke pga. stylingen. Hvor vil vi ha disse?
- Fikse routes (finne ut hvordan de bør være også - /account, /signin, /teacher)
- Opprette prototyper i figma
- Set up database
- + Add another card

Figure 7.3: Screenshot of a state of the Trello board

only be part of the teacher part of the application. As part of implementing authentication came the creation of components for sign up, sign in, sign out and changing of password.

Now that the web-application was connected with both a database and an authentication service, the main implementation of the game could start. Dummy data was added to the database to support testing of the components during development. The next main steps for the implementation were as follows:

1. Implement forms so that the teacher can create classrooms and games
2. Implement functionality for the students to join games (By using a game-PIN and selecting a name)
3. Implement a board with cards
4. Integrate editor and add a console
5. Create tasks in the database for the board

As soon as the board, editor and console were in place, it was time to implement the evaluation service for executing Python code. As stated, this was done by using the web framework Flask. Then, the feature containing the main objective of the game - guessing the picture - was implemented. In order to store the pictures in Firebase, the authors set up a Firebase Storage, where the pictures are uploaded. Lastly, the teacher analytics were implemented.

Chapter 8

Pilot Evaluation

This chapter presents a pilot evaluation of the first version of Code Flip, covering the student part of the game. The chapter begins by describing the purpose of the evaluation, the participants and the process, before presenting and discussing the results.

8.1 Purpose

The pilot evaluation was held to evaluate the usability of the game, and to rule out errors. It was also useful for assessing the evaluation process the authors planned to use during the main evaluation of the game in a classroom setting. A great concern for the usability of the game, is the error messages displayed to the user while trying to solve tasks. If these are not understandable, the player is likely to get confused or frustrated, and the messages will not serve their purpose of supporting the learning outcome. Since programming tasks usually have several possible solutions, the pilot evaluation was a great possibility for evaluating the error messages, so the participants were asked to pay extra attention to these. The participants were also asked to assess the quality of the tasks themselves.

8.2 Participants

The participants of the pilot evaluation were six fellow Computer Science students, acting as students who were to play the game. The authors acted as facilitators. All participants were recruited voluntarily. As they studied computer science, they already had an interest for programming, and sufficient knowledge to solve the tasks easily. They were older than the main target group, hence their opinion of the difficulty of the tasks and the engagement for the game were not directly relevant, but this was neither the purpose of the pilot evaluation.

8.3 Process

In advance of the pilot evaluation the facilitators created four games, where the participants were divided into pairs. All games used the same set of tasks, but the solution picture was different for each game. The participants were asked to bring their own computers. The evaluation was estimated to last for about an hour, including a short introduction of the master project and the game, as well as a focus group session at the end. Between playing each game, the participants got a five-minute break. Data collection was done through observations and through taking notes, both during game-play and a focus group.

All of the students were provided with a note containing game-PINs for four different games, a blank sheet of paper and a pen for taking notes. They were told to raise their hand to let the facilitators know if there was something they did not understand. If they noticed something strange about the game, they were to write it down. The games were played in the order they were listed on the provided note. They started by choosing *"I am a student"* on the home page, and entered with the first game-PIN. When all of the students had entered a game, the facilitators started it. When one team managed to guess the picture, the game was finished and the participants got a 5-minute break before the next game started. After having played through four games, they all participated in a focus group.

8.3.1 Focus Group

A focus group was held after the pilot evaluation to gather qualitative impressions about the game. Since most of the participants already knew each other, an open discussion was likely to work efficiently. The participants were familiar with user tests and well aware of the term usability, hence they could be asked quite open questions. The questions were inspired by the System Usability Scale (Brooke 1996), and were as follows:

- How would you describe your overall experience with the game?
- What did you like the most about the game? Why?
- What did you like the least? Why?
- Did you encounter any difficulties?
- Do you have any suggestions for improvement?

8.4 Results

This section presents the results from the pilot evaluation. The results are divided into two parts - observations during the game-play and feedback from the focus group.

8.4.1 Observations

The facilitators observed while the participants played the game. The observations were mainly positive, as the participants were seen to have few problems and were very eager to solve the tasks. It was evident that all of the teams wanted to guess the picture first to win the game. However, some obstacles occurred:

Unable to run the code

One of the participants was unable to run the code for the first task. The reason for this was that the participant closed the task before trying to run the code. This set the editor in an inactive state, without a *selected task*, hence it would not run.

Disrupted during a task

When the first team guessed the correct motive of the picture, the game automatically ended for all players. One of the participants stated during the game-play that it was frustrating to be interrupted in the middle of solving a task.

Easy to guess the pictures

Some of the pictures used for the games were very easy to guess. This resulted in the participants only solving a couple of tasks on the board before the game was finished.

8.4.2 Feedback

To give an overall view of the feedback that would require changes to the game, it is presented in Table 8.1. All the feedback is listed with an ID with the letter PF, pilot feedback, and a number. The participants also gave positive feedback. They thought the game was very clean and intuitive, and that the game concept was engaging. Their competitive instincts were activated by the competition of guessing pictures, and they became completely focused on solving tasks and winning the game.

ID	Feedback
PF01	Too complex game-PIN. Suggestion: Only use numbers.
PF02	No error message when trying to enter the game without selecting a name.
PF03	Does not see the guess button at first. Maybe because it has the same color as medium tasks.
PF04	The task instructions are a bit messy. Suggestion: Use code-highlighting or something to set certain words to bold or italic.
PF05	Get feedback if the code runs an infinite loop. Now nothing happens.
PF06	Some tasks are not precise enough. Struggled with some tasks where it did not state that it should call the function.
PF07	The pictures should be more difficult to guess.
PF08	Multiple game modes. Suggestion: letting the teacher select goals. Another goal can be to first finish all the tasks on the board.
PF09	Get the results after the game is finished. For example top 3 players on a podium.
PF10	Some of the tasks accept hard-coded solutions.

Table 8.1: Feedback from the pilot evaluation.

8.5 Discussion and Changes

The goal of the pilot evaluation was to evaluate the usability and assess the quality of the tasks. Even if the participants had a highly technical background, and were not part of the main target group, their opinions are valuable in regard to task quality and usability of interfaces. Some of the participants also had suggestions for additional features to the game concept. As there were only six participants, the focus was on qualitative data, as it is more significant than quantitative when there is a small amount of participants. The results are divided into three sections related to the feedback from the participants and the goals of the pilot evaluation: Usability, content and concept.

8.5.1 Usability

The pilot evaluation indicates that the overall usability of Code Flip is good. The participants stated that the game was very intuitive, which made it easy to understand what to do. One of the observations is related to usability: *Unable to run the code*. This problem is decided not to be prioritized. The reason is that only one of the six participants had this problem, and the option would be that they either will be able to run random code without selecting a task, which is not a part of the main concept of the game, or to get a modal which says ”you need to open a task”. The authors decide to wait with this change for further evaluation.

For the question *Did you encounter any difficulties?*, neither of the participants had any comments other than those already observed. Nevertheless, five of the feedback comments (PF01, PF02, PF03, PF04, PF05) were related to the user experience of the game. They are however not seen as critical for the user experience. For example PF03 was only mentioned by one of the participants, and during a game it is likely that the *Guess*-button eventually is discovered or explained by another team member.

8.5.2 Content

The content of the game includes the data displayed in the interfaces, such as the task descriptions, error messages, and pictures. Most of the tasks were perceived as simple and understandable. However, as stated in PF06, some of the tasks were not precise enough. This was mainly due to the custom error messages. Some tasks require that the user calls a function after creating it, which some participants pointed out to not be specified in the task descriptions.

Results from both observations and feedback states that the pictures were too easy to guess. According to the original game concept of Chapter 5.1.3, the board should in fact have 81 cards so that each card only uncovers a tiny piece of the picture. As stated in Section 6.2, more cards would require creation of a much bigger amount of tasks, hence it was not prioritized at this point. This is however easy to change in the future, as more tasks should be defined before a possible release of the game. Another solution to the problem is to use pictures that require solving of several tasks to guess.

8.5.3 Concept

The concept of Code Flip was perceived as exciting. The participants stated that they enjoyed playing the game and thought it was a fun way to solve programming tasks. They stated that the competition of guessing the picture made it even more engaging, and that it encouraged them to try to win the game. However, as mentioned, the participants already had an interest for programming, so it is not granted that the main target group will experience the same.

Feedback PF08 and PF09 are related to the game concept, both of which suggest additional features for the game. One of the students stated that the game could have an additional goal, such as completing the whole board. This was based on the fact that they did not have time to solve many tasks before the game was finished. Another student wanted a scoreboard after the game was finished, instead of a modal stating the winner team. The authors see these features as interesting features for the game. By having several goals, more students are likely to feel that they succeed in the game. The introduction of the goal of solving all the tasks on the board, allows for students who are good at programming to perform well. By keeping the first goal of guessing the pictures, also those who are not as skilled in programming, are able to win. As stated in the problem elaboration (Chapter 2), *"more students can become engaged if it becomes less probable that the same students win every time"* (Shim, Kwon, and Lee 2017). These changes to the game will in addition make the students solve more tasks. By implementing a scoreboard, the advantage of earning points can be introduced as well, which can increase the engagement for the game even more.

One participant experienced frustration when the game suddenly ended in the middle of solving a task. It is therefore discussed whether the game should continue until the last team is finished with their board. This way, players will not get frustrated for being disrupted in the middle of a task. However, if a team struggles with solving the tasks, the other teams will be left for a long time without anything to do, which can be more frustrating for more players. The decision fell on ending the game when three teams are finished with their board, as a compromise between the two situations. If there are only two teams in the game, it will end when both teams are finished.

8.5.4 Changes

Most of the feedback will either require cosmetic changes or modifications of small details. Consequently, not much effort would be needed to address all of the feedback, and it was decided to cover everything. All of the changes made to the game are presented in Table 8.2.

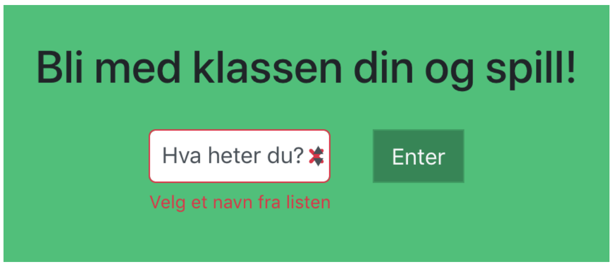
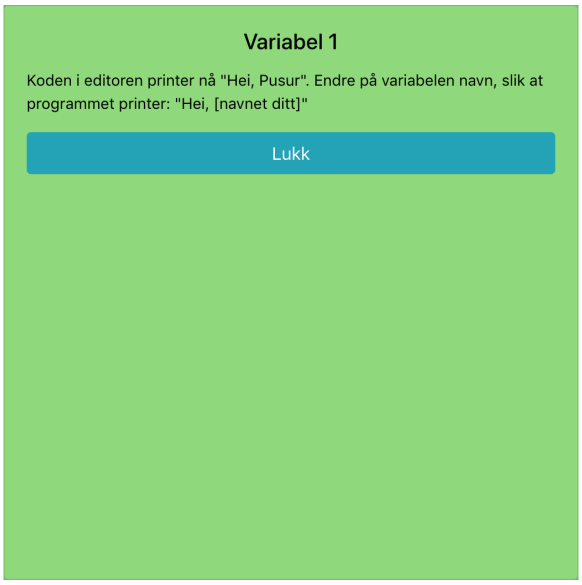


Figure 8.1: Solution of PF02.

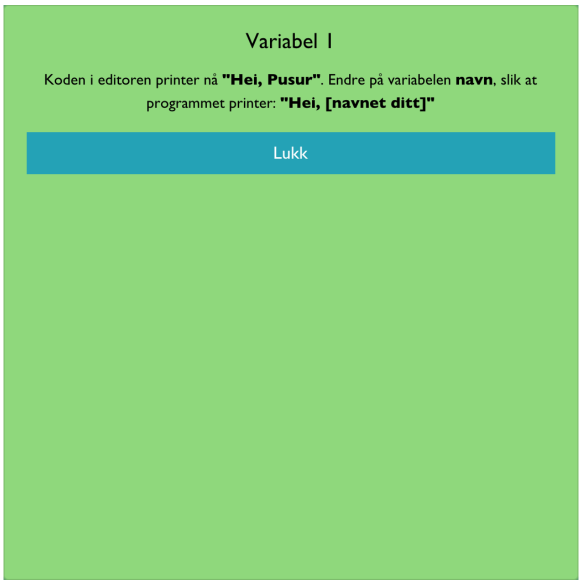
ID	Changes
PF01	The game-PIN is now a six digit number.
PF02	An error message is displayed if no name is selected when trying to enter a game, as shown in Figure 8.1.
PF03	The guess button is made bigger and with a color that is not used for any of the cards.
PF04	The task descriptions are written in HTML, and the code uses a html-to-react parser when presenting it. The old and new versions are shown in Figures 8.2a and 8.2b.
PF05	If the code runs for more than 3 seconds, a modal with a descriptive error message appears, as shown in Figure 8.3.
PF06	All tasks are assessed and reformulated if needed.
PF07	The pictures are replaced with new pictures that will be more difficult to guess without seeing big parts of them, such as close-up pictures of different motives.
PF08	The players now first have the goal of guessing the picture, then the goal of completing the board.
PF09	The teams now earn points by solving tasks. First three teams to guess the picture or solve the board also earn bonus points. The team with the most points wins the game. When the game is finished, the top three teams are displayed on a podium that becomes visible for all players, as seen in Figure 8.4.
PF10	The evaluations of the tasks are modified to ensure that they check the solution attempts strictly.

Table 8.2: Changes made after the pilot evaluation.


Variabel 1

Koden i editoren printer nå "Hei, Pusur". Endre på variabelen navn, slik at programmet printer: "Hei, [navnet ditt]"

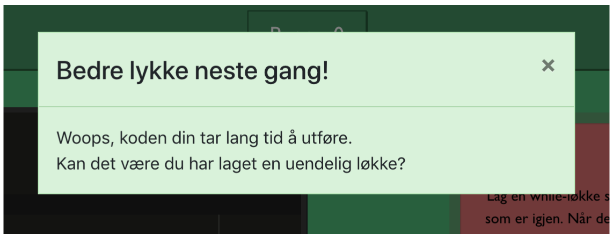
(a) Previous representation of task instructions.


Variabel I

Koden i editoren printer nå **"Hei, Pusur"**. Endre på variabelen **navn**, slik at programmet printer: **"Hei, [navnet ditt]"**

(b) New representation of task instructions.

Figure 8.2: Solution of PF04


Bedre lykke neste gang!

Woops, koden din tar lang tid å utføre.
Kan det være du har laget en uendelig løkke?

Figure 8.3: Solution of PF05.

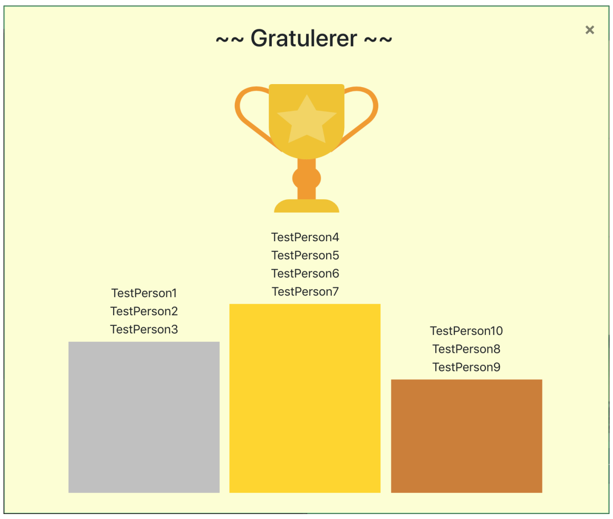


Figure 8.4: Solution of PR08 and PR09.

Chapter 9

Expert Evaluation

Before the main evaluation of Code Flip, the authors wanted to conduct a more thorough evaluation of the user experience, this time also for the teacher part of the game. As such they ran an expert evaluation to collect more qualitative data. This chapter describes the purpose, participants, and process of the expert evaluation, before presenting and discussing the results.

9.1 Purpose

The purpose of the expert evaluation was to get a professional, a person with a lot of experience in a certain field, to give feedback on the game. This evaluation would contribute to answering **RQ1.3**. A game expert was recruited to examine the user experience of the game, and a teacher expert to examine the user experience of the teacher part.

9.2 Participants

Both participants were recruited voluntarily. As before, the authors acted as facilitators. The teacher expert's background was from taking a master's degree in Natural Sciences with Teacher Education. The expert has worked as a substitute teacher in several lower secondary schools, had a temporary position as teacher for the elective course *Technology in practice* at one of the LSS in Trondheim, and worked as an e-teaching assistant for the Education Quality Division at NTNU, teaching the use of e-learning systems to university lecturers.

The game expert was a master student from the Department of Computer Science, with the specialization *Interaction Design and Game Technology*. This expert had experience with game development from courses at NTNU, and through developing several games for both mobile phones and computers. The game expert was also an eager gamer himself.

9.3 Process

The expert evaluation consisted of four main parts, each lasting thirty minutes. Even if the experts were to evaluate two different parts of the game, the evaluations were held together. This means that the teacher expert participated in all parts of the evaluation, while the game expert participated to the second and fourth part. This would allow for more analytics during game-play for the teacher, through following the analytics view while the game expert played the game.

In the first part, the teacher expert tested the teacher part of the game. Tasks were given for creating a user account, and for creating a game that could be played afterwards by the game expert. The second part was for testing the game-play. The game expert, as well as one of the facilitators, played the game, while the teacher was ordered to follow the game from the teacher part of the application. In the third part, the teacher tested and analyzed the *analytics after game-play*. The last part included two interviews with the experts, each of them held separately.

The experts were asked to bring their own computers. Both were given a blank sheet of paper and a pen for taking notes. They were told that if there was something they did not understand, they should raise their hand to let the facilitators know, and if they noticed something strange they should write it down. They were given a 5-minute break between each of the four parts. Qualitative data was collected through observations and interviews.

9.3.1 User tests

The experts participated in different user tests. One of the user tests was conducted simultaneously with both experts, while the rest was only held with the teacher expert. The user tests were planned with inspiration from Shneiderman and Plaisant (2010). In advance of the user test, the experts were informed that the test was voluntary, that it would be possible to break the user test at any time, that no help would be received during the test, and that it was the game that was being tested, not them. The experts were also encouraged to think out loud to explain actions and assumptions.

Creating the Game

The first part included a user test with the teacher expert. One of the facilitators observed while the other gave assignments to the expert. The test started at the home page view, as is shown in Figure 6.3. The assignments were as follows:

- Log in as a teacher
- Create a game with the following data:
  - **Students:** Game expert, Facilitator, Player 3
  - **Number of teams:** 2 (Team 1: Game expert and Facilitator, Team 2: Player 3)
  - **Picture:** Match

The game expert and the facilitator were put on the same team so that the game expert would experience how the game board changed when other team-members selected and solved tasks.

Playing the Game and Analyzing the Game Analytics

The game was now ready to be played. The game expert and one of the facilitators entered the game, and the teacher expert started the game. The teacher was asked to follow the game analytics while the game was being played. Since the game requires at least two or three teams to complete the board before it crowns the winners, some dummy data was stored in the database, so that this would happen when only one of the teams had finished.

Analyzing the Analytics after the Game is Finished

After the game was finished, the teacher expert got a new assignment: *"Imagine that your class has been playing several games over the last couple of weeks. Now you want to see how many tasks the Facilitator has solved in total"*. The expert was then asked to gather as much information as he could from the analytics view.

9.3.2 Interviews

The experts were interviewed individually. The game expert was asked questions about the game, while the teacher expert received questions about the teacher part. The interviews were semi-structured and inspired by the System Usability Scale (Brooke 1996). The two interview guides were as follows:

Interview guide for the game expert:

1. How would you describe your overall experience with Code Flip?
2. What did you like the most about Code Flip? Why?
3. What did you like the least? Why?
    - Did you encounter any difficulties?
4. Do you have any other thoughts about the game that have not been covered?

Interview guide for the teacher expert:

1. How would you describe your overall experience with the teacher part of Code Flip?
2. What did you like the most about the teacher view? Why?
3. What did you like the least? Why?
    - Did you encounter any difficulties?
4. Did you think the game analytics during game-play gave you valuable information?
    - Did you think the game analytics were easy to understand?
    - What kind of information did you get from the analytics? (Both during a game and after several games)
    - Were you missing any analytics in the game?
5. Do you have any other thoughts about the teacher part that have not been covered?

9.4 Results

The feedback from the interviews is shown in Tables (9.1 and 9.2) to give a better overview. The feedback ID for the game expert is GE, and TE for the teacher expert. Positive feedback was also given during the interviews. The game expert thought the game seemed to be a suitable game for beginners of programming, and that the goal of guessing the pictures gave the game more meaning in addition to solving programming tasks. The teacher expert found it easy to log in and create both classrooms and games, as was also observed during the user test. He also uttered that the dynamic data in the analytics view was very helpful with showing the students' progression during the game.

There were only a few notes from the observations of the user test and game-play. These were also mentioned by the experts during the interviews, hence they are presented together with the interview results. The small amount of notes suggests that the usability of the game is good. Both experts managed to perform the user tasks without any obstacles.

ID	Feedback
GE01	All buttons should do the same when hovered.
GE02	Be consistent with sharp and rounded corners.
GE03	When using the enter button on the keyboard for submitting a guess, the page reloads instead of submitting the guess.
GE04	Be able to run code even if the game is finished.
GE05	Be able to open a task even if it is solved, to see and discuss solutions with classmates.
GE06	Disabled tasks should give information about which player is working on it.
GE07	Overview of how the other teams are doing.
GE08	The pictures should be loaded faster.
GE09	Remove the guess button when the team has correctly guessed the picture.
GE10	Make the elements of the game appear to be more in the front of the screen.

Table 9.1: Feedback from the game expert

9.5 Discussion and Changes

The goal of the expert evaluation was to get a more thorough evaluation of the user experience for both parts of the game. Since the two parts were evaluated by one expert each, only qualitative data was gathered. The results were therefore analyzed qualitatively, keeping in mind that they came from only one expert. Results are categorized in: Usability, content, concept and styling.

ID	Feedback
TE01	Hard to distinguish between the blue and green color for the different game statuses.
TE02	The *Start Game*-button should have some information about what it is for. (The teacher did not understand that the game should not be activated before all players have joined).
TE03	Tiresome to calculate how much time one student has used on a task, given the start time. (The teacher did at first not understand that the text behind the clock symbol was a timestamp).
TE04	Did not understand the number representing difficulty for each task.
TE05	The overview of game-play analytics would probably become messy, if for instance 30 students are playing at the same time.
TE06	Being able to see how many tasks that are left.
TE07	See the analytics for a specific student by clicking on their name in the classroom-view (Figure 6.13), instead of having a students-view (Figure 6.19).
TE08	When viewing a student's task during a game it only shows the solution and not the task's instructions.

Table 9.2: Feedback from the teacher expert.

9.5.1 Usability

The usability of Code Flip was perceived as good. Both of the experts performed the user test assignments correctly and showed confidence when maneuvering in the game. Nevertheless, both experts had some comments regarding the usability (GE03, GE08, TE02, TE03, TE07).

The game expert had two concerns regarding usability. The first was that some of the modals did not accept submission by using the enter button on the keyboard (GE03). The other issue (GE08) was that when a task was completed, the picture was loaded slowly.

The teacher expert mentioned that the functionality of the *Start game*-button was unclear (TE02). It was not obvious that if the teacher activated the game, the students would start solving the tasks at different times depending on when they enter the game. Another issue was that when a student opened a task (without completing the task), the analytics only showed the start time for the task. This means that if the teacher wants to know how much time a student has used on a task, it would have to be deducted from the timestamp. The third usability issue the teacher expert expressed was being able to maneuver to the *after-game-analytics* view for a student from the classroom view instead of having a separate tab for students in the navigation bar.

The authors decided to prioritize GE03, TE02 and TE03. TE07 was not prioritized because this was a rather big change for the navigation in the web application, and it was decided to do further evaluation of this. GE08 was neither prioritized. Each piece of the image is now loaded when a task is solved, hence the user will notice a small latency. A solution would be to load all of the parts from the beginning, and get them from the local storage when needed. However, if all parts are loaded from the beginning, the user is able to obtain all images by inspecting the page, even though the task is not solved yet. Besides, a small latency for loading images does not directly affect the concept of the game.

9.5.2 Content

The game expert did not have any comments about the content of the game, which suggests that the changes from the pilot test were successful. The teacher expert mentioned one issue. During game-play, when the teacher opened a task by a student, it only showed the title, difficulty-level, time used and the solution, and not the task description. Since the teacher was not familiar with the tasks, which will be the case for most end-users of the game, the title alone is not enough to understand what a student has solved. This was seen an important issue to prioritize.

9.5.3 Concept

For the game concept, the game expert had several suggestions for additional features that would include more gamification. However, most of them were big features that would require a lot of implementation time. Since new features were not part of the purpose of the evaluation, these were not prioritized, but they can be interesting to consider in future versions of the game. The features are covered by issues GE04, GE05, GE06 and GE07.

The teacher stated that it would be beneficial to see how many tasks that are left for each team (TE06). When a lot of students are playing, it would be difficult to get a good overview of how the teams are doing. Implementation of more team analytics are therefore prioritized for change.

9.5.4 Styling

A lot of the feedback from the expert evaluation would require changes of the game's styling. Since styling is easy to manage, all the feedback was addressed. This includes GE01, GE02, GE09, GE10, TE01, T04 and T05.

9.5.5 Changes

The changes covering the prioritized feedback are shown in Table 9.3.

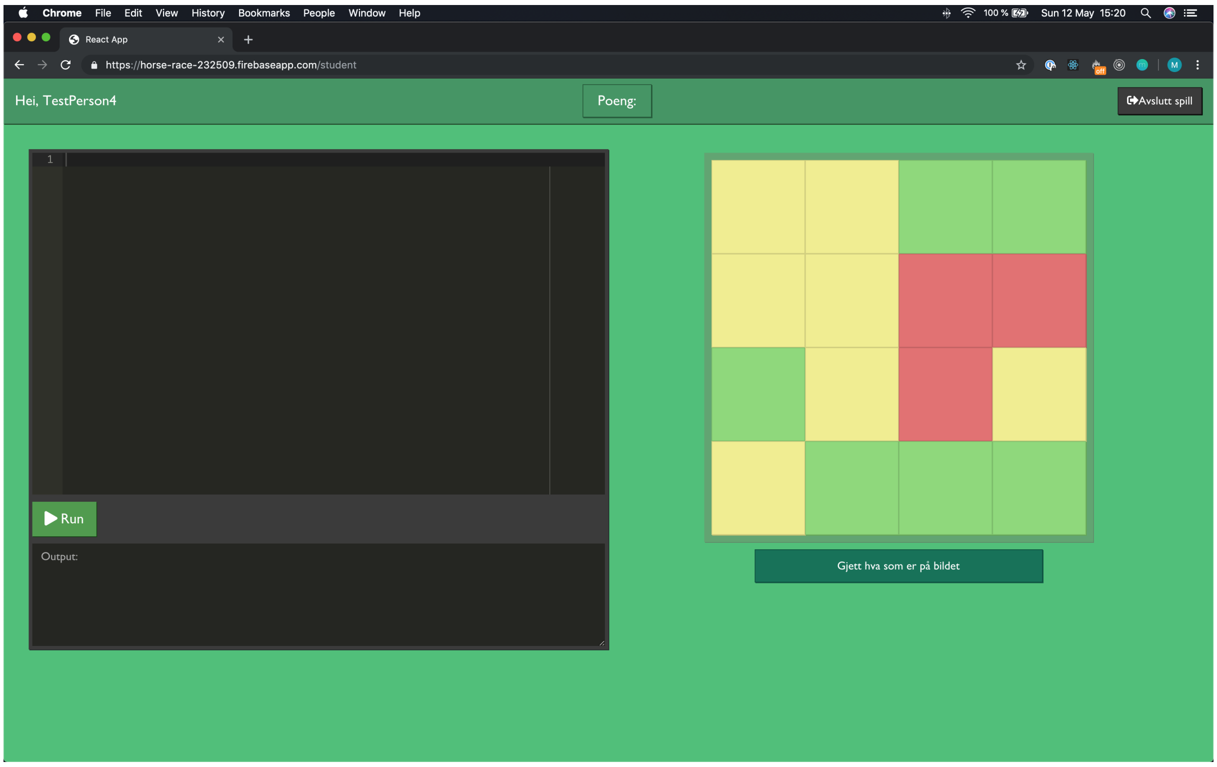
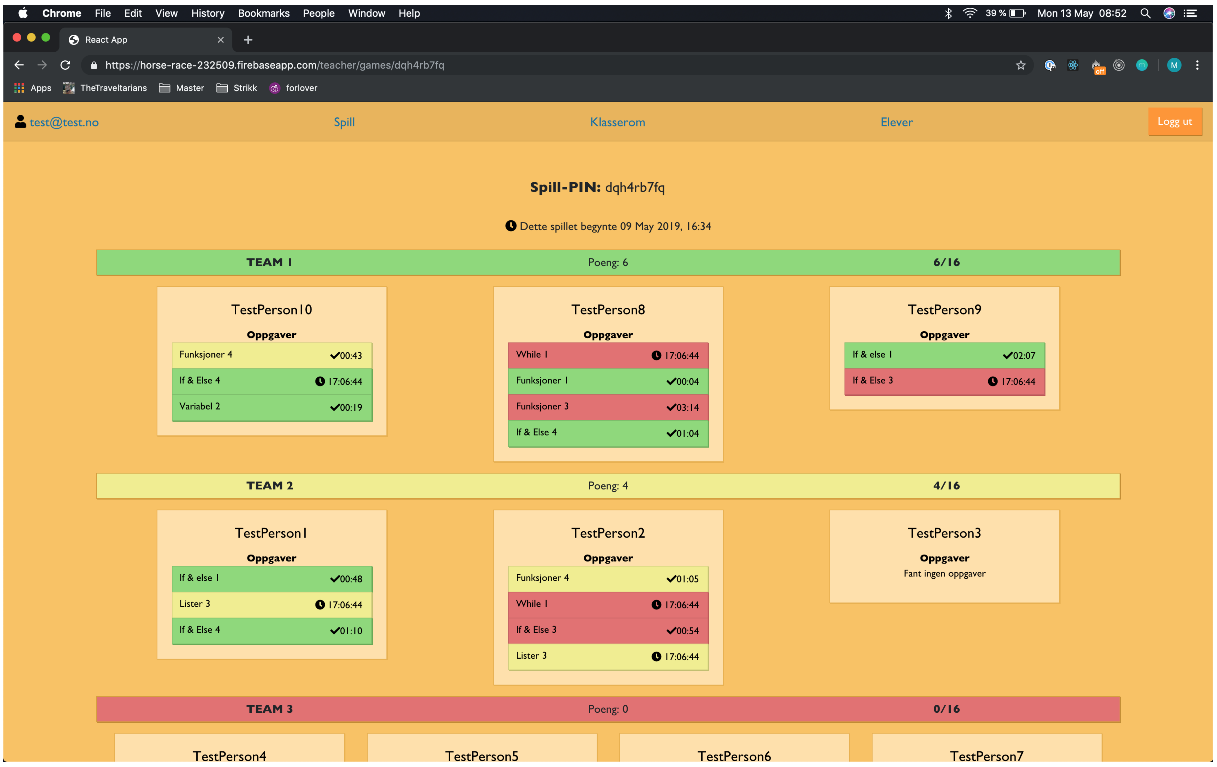

Chrome File Edit View History Bookmarks People Window Help
Sun 12 May 15:20
React App
https://horse-race-232509.firebaseapp.com/student
Hei, TestPerson4
Poeng:
Avslutt spill
Run
Output:
Gjett hva som er på bildet

Figure 9.1: Solution of GE10


Chrome File Edit View History Bookmarks People Window Help
Mon 13 May 08:52
React App
https://horse-race-232509.firebaseapp.com/teacher/games/dqh4rb7fq
Apps TheTraveltarians Master Strikk forlover
test@test.no
Spill
Klasserom
Elever
Logg ut

Spill-PIN: dqh4rb7fq

Dette spillet begynte 09 May 2019, 16:34

TEAM 1 Poeng: 6 **6/16**

TestPerson10
Oppgaver

Funksjoner 4	✓00:43
If & Else 4	17:06:44
Variabel 2	✓00:19

TestPerson8
Oppgaver

While 1	17:06:44
Funksjoner 1	✓00:04
Funksjoner 3	✓03:14
If & Else 4	✓01:04

TestPerson9
Oppgaver

If & else 1	✓02:07
If & Else 3	17:06:44

TEAM 2 Poeng: 4 **4/16**

TestPerson1
Oppgaver

If & else 1	✓00:48
Lister 3	17:06:44
If & Else 4	✓01:10

TestPerson2
Oppgaver

Funksjoner 4	✓01:05
While 1	17:06:44
If & Else 3	✓00:54
Lister 3	17:06:44

TestPerson3
Oppgaver
Fant ingen oppgaver

TEAM 3 Poeng: 0 **0/16**

TestPerson4 TestPerson5 TestPerson6 TestPerson7

Figure 9.2: Solution of TE04, TE05 and TE06

ID	Change
GE01	All buttons now have the same hovering effects.
GE02	All elements are changed to have sharp corners.
GE03	All forms accept submitting through the enter-button on the keyboard.
GE09	The guess button is removed after the team has guessed correctly.
GE10	The new style of the game view can be seen in Figure 9.1. Drop shadows and frames have been added to the elements.
TE01	The colors for game status have been modified.
TE02	The *Start Game*-button now has informational text beneath, stating that the teacher should wait for the students to enter the game before activating the it.
TE03	By installing the react component ”react-moment” with npm, it was easy to include a timer that calculates the duration from a certain timestamp. It also updates automatically every [x] seconds. This was set to 15 seconds.
TE04	The difficulty of tasks is now represented by colors, as shown in Figure 9.2. (Easy = green, medium = yellow, and hard = red).
TE05	To make the analytics view more clear, the teams are separated into different rows with a header stating which team it represents, as can be seen in Figure 9.2. The space between the student cards, as well as the width of the cards, are decreased.
TE06	As shown in Figure 9.2, each team’s header includes an overview of the amount of tasks solved by that team. In addition, the amount of points is added to the header.
TE08	The selected task’s description is displayed in the overview of a student’s task.

Table 9.3: Changes made after the expert evaluation

Chapter 10

Main Evaluation

Code Flip had now been through two evaluations, and parts of the game had been altered accordingly in order to improve its usability, content and concept. It was therefore time for evaluations in the context the game was created for, which were to be held in four different lower secondary school classes at two different schools.

This chapter describes the main evaluations of Code Flip. Both the student and teacher parts of the game were evaluated, and hence the results are described and discussed separately in the context of their target groups.

10.1 Purpose

The purpose of the main evaluation was to evaluate the game in a classroom setting with the correct target group, in an attempt to answer **RQ1.3**. The primary focus of the evaluation was to find out whether the game succeeds in motivating students for programming, and whether the teachers find the game analytics and the game concept useful for the Elective Course of Programming. These matters are strongly connected to the concept and usability of the game. The content of the game, i.e. the programming tasks and pictures, were emphasized on a secondary level. Tasks and pictures can easily be modified, and are thus not directly related to the applicability of the game concept.

10.2 Participants

The recruited participants of the final evaluation were lower secondary school students and their respective course teachers - two teachers in total. The students were from four different classes at two different schools, and they were recruited through their teachers at the LKK conference. Unfortunately, the evaluations at one of the schools were cancelled two days prior, and it was not possible to schedule new evaluations of the game this close to the end of the school year. Consequently, the total amount of participants turned out to be 14 students and one teacher, originating from one school.

All participants were recruited voluntarily, and since no personal data was gathered, it was not necessary to get approval from NSD. After performing the evaluations, the participants were rewarded with a small gift.

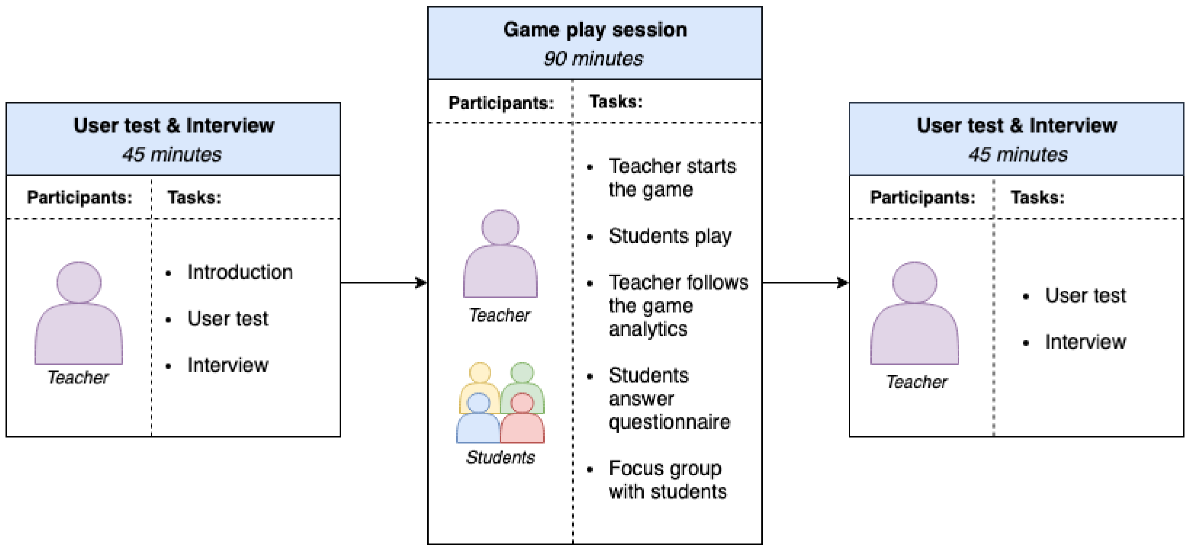


Figure 10.1: A visualization of the process of the main evaluation.

10.2.1 The Participating Class

The participating class was a 9th grade LSS class of the Elective Course of Programming. The class had 18 students, but four of the students were not present during the evaluation. In total 14 students participated, in addition to their teacher.

Their experience with programming in Python was from working with the content of Lær Kidsa Koding[1], mainly at the easy and intermediate levels, but a few had also experimented with the advanced content. They had however not worked with this content since the fall of 2018, which was several months ago.

10.2.2 The Other Scheduled Classes

The three other classes that were supposed to participate originated from another school. One of them was a class of the ECP, while the other two were classes of mathematics. They all had the same teacher, through whom they were recruited.

The ECP class had students with varying Python-experience. All students had recently been through the programming content of Lokus[2] for LSS, while a few had additionally been working on a project for several months where they created a game with Python.

The classes of mathematics had been introduced to the content of Lokus by their teacher, with the intention of using Python as a tool for mathematics. Their teacher thought a programming game could be useful for them, considering that programming is to be part of the mathematics curriculum by 2020 (Regjeringen 2018). Hence, a new set of mathematical tasks was made.

10.3 Process

An overview of the process can be seen in Figure 10.1. In advance of each test to be performed in the classroom, a user test was held with the respective course teacher. 45 minutes were allocated for this first session. The facilitators started by showing the student part of the game, so that the teacher knew what the tool was for. The teacher was not presented to the teacher part, in order to be able to assess the usability of this part during the test. A semi-structured interview was held after, in order to capture the teachers' opinions about the usability of the features tested so far.

After the initial user test with the teacher, two games were readily created for use in class. The game was tested in the scheduled class for the ECP, and the facilitators had the whole scheduled time available - 90 minutes. 10 minutes were allocated for giving an introduction to the game and arranging for the students to be placed in their pre-allocated teams. The introduction included a description of the goal of the game, the game view, and what the different game elements were for. 60 minutes were allocated for playing, and the teacher started the game through the teacher view. The teacher was instructed to pay attention to the teacher view throughout the game-play, and reflect on the information seen there. After the game-play, the students were asked to answer a questionnaire, for which they were provided a URL. Lastly, a focus group was arranged in order to triangulate the quantitative questionnaire data with qualitative data.

After the game had been played through in the classroom, the students were dismissed, and a final session was held with their teacher. This session included a new user test with only one task, followed by an interview. The parts to be covered by this session were the analytics to be seen during game-play, the analytics given after a game had been played, and what the teacher thought of the game as a whole.

10.3.1 User Tests

The user tests were planned with inspiration from Shneiderman and Plaisant (2010). Before starting the user test, the teacher was informed that the participation was voluntary, that no help would offered during the test, and that it was the game that was being tested, not the teacher. The teacher was also encouraged to think out loud. One of the facilitators observed and took notes, while the other led the session, presenting one task at a time. To ensure the anonymity of the teacher, and to save time, a mock user account had already been created, and the teacher was instructed to log in using these credentials. When creating the games, the teacher was told which pictures and task sets to select. The predefined sets of tasks were *Functions*, *Easy Mix*, and *Mathematics*. For the evaluation in the classes of mathematics, the teacher would be instructed to choose the set *Mathematics*. The tasks of the user test were as follows:

- The user test held in advance of the class:
  1. Log in as a teacher
  2. Create a game (choose the picture ”Match” and task set ”Easy Mix”)
  3. In the class you are going to start the game when the students have joined and are ready to play. Show how you would do this.

[1] `https://oppgaver.kidsakoder.no/python`
[2] `https://tjenester.lokus.no/open/programmering/ungdomstrinn.html`

4. Create another game to be used in the class (choose the picture ”Car” and task set ”Functions”)

- The user test held after class:

  1. Let us say the students have played this game in class for several weeks, and been through a lot of games. Find the overview of all the tasks one of your students has solved.

10.3.2 Interviews

Interviews were held with the teacher after the user test, in order to collect qualitative data about the usability of the teacher part of the game, and about the teacher’s impression of how the game worked in the class. The interviews were semi-structured, to cover what was needed while letting the teacher converse freely about the impressions of the game. The planned questions of the interviews are listed below. They are inspired by the System Usability Scale (Brooke 1996). As during the user test, the interviews were led by one of the facilitators, while the other noted down the answers to each question.

- The interview held in advance of the class:
  - How would you describe your overall experience with Code Flip?
  - What did you like the most about Code Flip? Why?
  - What did you like the least? Why?
- The interview held after class:
  - What do you think of the analytics view of the students?
    - Was there something about this feature that was hard to understand?
    - Are there any analytics that you would like to have, that are not already there?
    - Do you have any suggestions for improvements?
    - Could these analytics be useful for evaluation of your students?
  - How do you think the game-play in the class worked?
    - Do you think the game can help increase knowledge about programming?
    - Do you think the game can help increase motivation for programming?
    - Did the game focus enough on programming?
  - Do you think the game would be a useful resource for the Elective Course of Programming?

10.3.3 Questionnaire

The questionnaire to be answered by the students after playing, was for capturing each individual student’s opinion of the game. The questions were once again constructed with inspiration from the System Usability Scale (Brooke 1996). The focus was on whether the game was understandable and easy to use, whether it increased motivation and confidence for programming, and how the students enjoyed different elements of the game that were included for motivational purposes, such as competition and cooperation. Each question was formulated as a statement in first-person and with a simple language, in an attempt to make them easy to understand for the students. They were to be answered on a 5-point Likert scale, ranging from ”Strongly disagree” to ”Strongly agree”. The statements translated to English are listed below, and the original questionnaire in Norwegian can be seen in Appendix F.

1. I would like to play the game again.
2. I found it difficult to pay attention to everything that was going on in the game.
3. I found it easy to understand what I was supposed to do.
4. I think the game is suitable for the Elective Course of Programming.
5. I thought the game was cumbersome to play.
6. I needed help before I could start playing the game.
7. I thought it was fun to solve the tasks.
8. I thought it was fun to play the game.
9. I was motivated by being part of a team.
10. I was motivated to solve tasks.
11. I was motivated because I wanted to win.
12. The game helped me troubleshoot my code.
13. I managed to focus on solving the tasks even though I wanted to win.
14. I have a better understanding of programming after playing the game.
15. I think the game should be used in the Elective Course of Programming.

10.3.4 Focus Group

The focus group held with the students was for triangulating the quantitative data of the questionnaires with some qualitative data. The facilitators asked open-ended questions about the game for the students to answer. Several of the questions were similar to those of the questionnaire in order to give the opportunity to elaborate on their answers. The predefined questions were as follows:

- What did you enjoy the most about the game?
- What did you not like? Why?
- Were there things that were hard to understand? What?
- What did you think of the tasks? Were they easy or hard to answer?
- What did you think about guessing on the picture?

- Did you feel that you learned something?
- Would you prefer competing in teams or individually in this game?
- Do you have suggestions for improvements?

10.4 Results

This section lists the results of the evaluation that was carried out. Since the game was evaluated from two different perspectives, results are presented in two different sections - *Teacher* and *Students*.

10.4.1 Teacher

From the teacher evaluation, data was gathered through observations during the user tests, and through interviews after the user tests. The results are presented below, and they will be discussed in Section 10.5.

Observations

The user test held with the teacher in advance of the class suggested that the teacher part was intuitive and easy to understand. The teacher had some questions during the tests, such as *"is it possible to add a name to a classroom afterwards?"*, which is currently not possible. It was also not obvious that when separating students into more than two different teams by toggling the color of student names, it was required to click on names multiple times, not just once. Since the teacher's screen size was small, all the student names of the game were displayed in a single column. This resulted in a very long list, and the teacher had to scroll up and down to see all the names, and remember which names got the different team colors.

After the teacher had successfully created a new game, it appeared in the list of games. However, the teacher did not understand that in order to start the game, it had to be clicked on in the game list. Instead, the teacher navigated to the home page, and back to classrooms, clicked on a classroom and so on, before finally seeing that it was possible to click on games in the game list. When creating a second game, as part of the fourth assignment, it went a lot faster. The teacher emphasized that it was very easy to divide students in teams now that the process was familiar.

For the user test after the game play session with the students, where the assignment was to find the after-game analytics, no problems were encountered.

Interviews

The interview held after the first user test reflects the observations - that the teacher part was intuitive and easy to understand. The teacher stated that it was clear, simple, understandable, and that there was not much to do wrong. A suggestion for improvement was to be able to see and choose previously used team allocations, to know who had been teamed up before, and to not have to construct an identical team distribution again. Another suggestion was that when the team color was changed for a student name, the box containing the student name could be moved up or down to be placed together with the corresponding team. Also as observed, the teacher would prefer that all names were visible on the screen at the same time, to avoid scrolling.

Important features of the teacher part of Code Flip are the analytics views, both during and after game-play. For the analytics view seen during game-play, the teacher emphasized that it was easy to understand. The teacher especially liked that it was possible to see a team's score, which tasks a student had solved and which had just been looked at. It was also appreciated that the view showed time spent on a task and difficulty of the solved tasks. However, the teacher did not figure out that it was possible to click on a task during game-play to see how a student had solved it. When this was pointed out, the teacher asked whether the solution displayed for a task was the student's solution, or the correct answer. Regarding suggestions for improvements or desired analytics, the teacher would have liked to see if a team had managed to guess the picture. Additionally, it was suggested to implement a matrix displaying students and tasks to give an overall overview of the students' performance. Each task for each student would then be highlighted in green for the correct answer, and red if it were opened but not solved. For the questions concerning the after-game analytics, the teacher stated that it could be used to evaluate the students in Python, as the students are evaluated for each topic (Python, Scratch, etc.).

> *"The number of solved tasks usually gives a good idea of the students understanding"* - the teacher.

The game-play went well according to the teacher. Some of the teams did not cooperate much, while others took advantage of each other and discussed the tasks. It was observed that some students opened google's search engine to search for answers, which the teacher thought was a good thing, as it suggested that the students were eager to seek help for solving tasks instead of giving up when stuck. The students showed a lot of enthusiasm for the game throughout the game-play, even though some lost focus towards the end. Through the analytics it was seen that students were quick to open tasks without solving them, so towards the end only the most difficult tasks were left. The teacher thought the picture worked as a good motivator for the students who were not as skilled in programming, since they at least could participate by guessing the picture. A suggestion for a gamification feature that would make the game more fun was to add sound effects. However, the teacher was unsure if this would disturb some students.

To the question *"Do you think the game could increase knowledge about programming?"*, the teacher very much agreed. It was stated that the students had gone through a lot of different concepts in programming, and that the game worked as a motivating way of solving tasks. Also the fact that the students worked in groups and could discuss tasks, and that competition was involved, would make them more focused when solving the tasks.

Concerning the question about the students' motivation for programming, the teacher thought that everything involving games would make things more entertaining for the students. As already stated, the element of competition was also an important factor for increasing motivation. The teacher thought the game had enough focus on programming, especially since it was to be used an elective course, and elective courses are meant to increase motivation for school.

The last questions was *"Do you think the game would be a useful resource for the Elective Course of Programming?"*, to which it was answered:

> *"Yes, absolutely. I imagine that we first need to have a basic introduction of Python. The game can then be a used as a great recap of the concepts, as a way to let them test what they have learned."*

The authors also asked if the teacher would employ the game in the next year's class if it were released, to which it was stated: *"yes, I've had the course for almost one year now, and am always looking for new tools to use in my lectures"*. The teacher's challenge with using the game was that it required an introduction of Python in advance, which was not heavily weighted for this year's class, but it would be a good encouragement for emphasizing Python more the next year.

10.4.2 Students

From the student perspective, data was gathered through observations, questionnaires and a focus group. The results are presented below, while a discussion follows in Section 10.5.

Observations

From the observations of the game-play it was evident that the students had not programmed in Python for a while. Most struggled with both programming concepts and Python syntax. The facilitators therefore helped some students with tasks, so that they would be able to play through the whole game. Still, the students showed a lot of enthusiasm for the game and for solving tasks. They were eager to find out what the picture showed, and hence worked hard with the tasks. When one of the students of a team received help, the other students on the team carefully listened. On teams where one already had some knowledge, and was able to solve several tasks, the others received help from the skilled student. As the teacher stated, some of the teams were very good at cooperating, while others did not cooperate at all, and some students lost focus towards the end.

Focus Group

The focus group did not work as well as the authors hoped for. This was most likely because there were a lot of participants, hence they may have found it intimidating to contribute, and as mentioned they became a bit unfocused at the end of the session. However, the answers that were given reflect that the students enjoyed the game. Several of them stated that it was fun, and that they enjoyed competing in teams. The game was *simple and nicely made*, and they liked guessing on the picture. The students thought the difficulty of the tasks was a bit advanced compared to their level of Python skills. For the question *"Did you feel like you learned something?"*, the students emphasized that they felt they had learned from playing the game.

Questionnaire

The post-game questionnaire reflects that the game succeeded in motivating the students for programming. It also shows that most students enjoyed playing the game. Table 10.1 shows the average scores of statement 1, 7, 8, 9, 10 and 11, i.e. statements regarding the motivation and amusement for the game, on a scale from 1 (Strongly disagree) to 5 (Strongly agree). All of the statements received rather high average scores, which indicates that most of the students thought the game was motivating and fun to play. The results also show that statement 1, 7, 8 and 11 only has answers varying from 3 (Neutral) to 5 (Strongly agree). For statement 9 and 10 option 2 (Disagree) got one vote. Graphs of all the answers can be seen in Appendix G.

ID	Statement	Mean
1	I would like to play the game again	4.0
7	I thought it was fun to solve the tasks	4.2
8	I thought it was fun to play the game	4.4
9	I was motivated by being part of a team	4.1
10	I was motivated to solve tasks	3.9
11	I was motivated because I wanted to win	4.5

Table 10.1: Mean score for statements regarding motivation and amusement (n=14).

ID	Statement	Mean
2	I found it difficult to pay attention to everything that was going on in the game	2.6
3	I found it easy to understand what I was supposed to do	3.3
5	I thought the game was cumbersome to play	2.3
6	I needed help before I could start playing the game	3.9

Table 10.2: Mean score for statements regarding difficulty (n=14).

Another part of the statements was concerned with the difficulty of the game - if there was too much happening in the game, if they thought it was hard to understand what they were supposed to do, if it was cumbersome to play, and if they needed help to start playing. The mean answers to these statements are shown in Table 10.2. All four statements reflect that many of the students thought the game was simple enough to get a good grasp of what was happening, and that it was simple to play. There were however a couple of students who thought the game was difficult to play and follow. Statement 6 is strongly related to statement 3, as the students would need help to start playing the game if they did not understand what they were supposed to do. However, ten of the students answered that they agreed or strongly agreed that they needed help before playing the game, while only two thought it was hard to understand what to do.

When developing the game, the authors created customized error messages for each of the tasks. This was to make the errors and exceptions more understandable for the students, as some of the teachers from the interviews had stated that their students struggled with understand error logs. Statement 12 indicated whether the students found the error messages helpful. The answers, as seen in Figure 10.2, show that eleven of the fourteen students thought the game helped them with troubleshooting their code.

A concern of the authors was that the students might forget to focus on the tasks, struggle with reading the instructions and use most of their time looking at the parts of the picture. Statement 13 addresses this concern, and the answers, as seen in Table 10.3, indicate that most of the students were able to concentrate on the tasks, even if they wanted to win.

Answers to statement 14, as seen in Figure 10.4, reflect that ten of the students felt like they had learned something from playing the game. Four of the students answered neutral, which

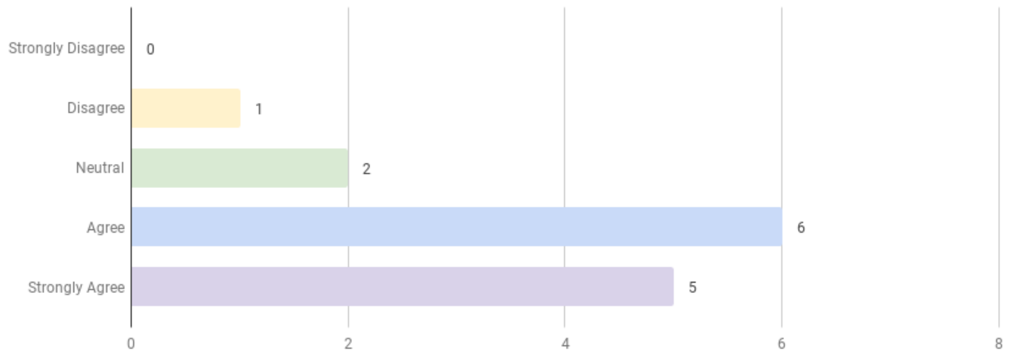


Figure 10.2: Answers to statement 12: *The game helped me troubleshoot my code.*

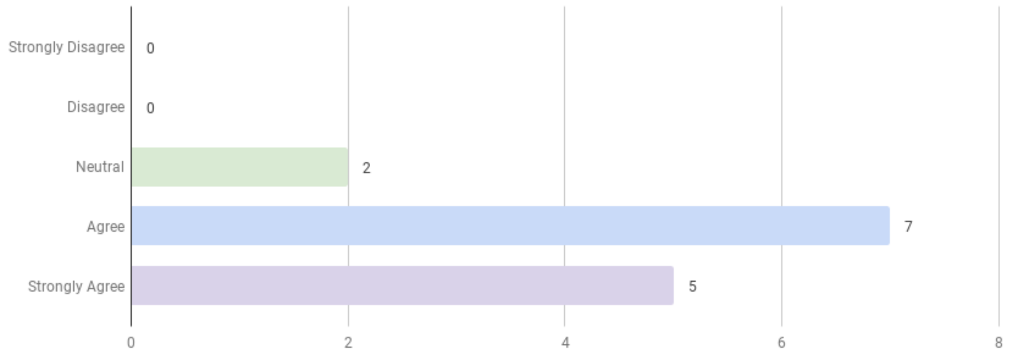


Figure 10.3: Answers to statement 13: *I managed to focus on solving the tasks even though I wanted to win.*

probably means that they were not sure. This does not reflect the actual learning outcome of the game, but it gives an idea about whether their confidence for programming increased.

Two statements concerned the usage of the game in the ECP - statement 4 and statement 15. Both of them received positive answers. For statement 4, all of the students, except one who voted neutral, agreed or strongly agreed that the game is suitable for the ECP, as can be seen in Table 10.5. Statement 15 (Table 10.6) got even more positive answers, where 10 answered that they strongly agree that the game should be used in the ECP.

10.5 Discussion

This section presents the discussion of the results obtained in the main evaluation. Due to the small amount of students that ended up being able to participate, the conclusions of the main evaluation are not necessarily correct for other classes and teachers. The discussion is centered around the usability, content and concept of the game.

Considering that programming is to be part of the mathematics curriculum by 2020 (Regjeringen 2018), and one of the participating teachers suggested to test the game with the classes of mathematics, a short discussion of the game's potential in other courses is given as well.

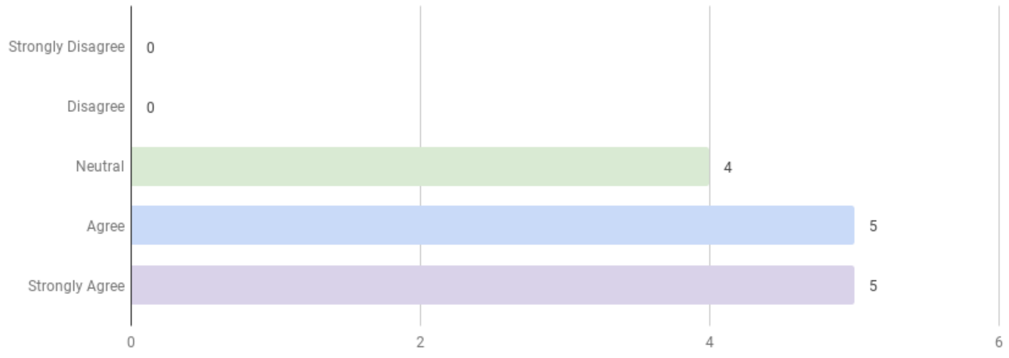


Figure 10.4: Answers to statement 14: *I have a better understanding of programming after playing the game.*

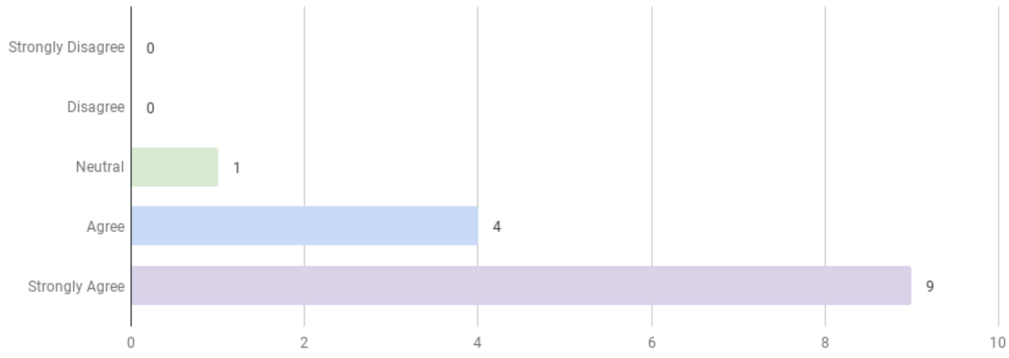


Figure 10.5: Answers to statement 4: *I think the game is suitable for the Elective Course of Programming.*

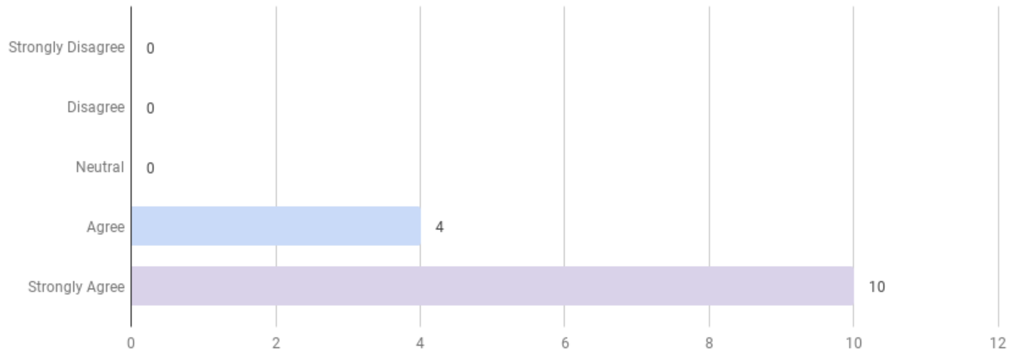


Figure 10.6: Answers to statement 15: *I think the game should be used in the Elective Course of Programming.*

10.5.1 Usability

The observations from the user tests with the teacher indicate that the teacher part of Code Flip is intuitive. Since the interfaces were completely new to the teacher, some time was spent observing and gathering information from the interface when doing the assignments. As the teacher became familiar with the application, and was to create the second game, the operations went a lot faster and smoother. The authors see this as quite natural, and that the reason for using some time the first time was simply that a user needs time to get familiar with a system. However, more effort can be put into improving the user interface in the future.

The usability of the game Code Flip was perceived as neither bad nor very good. Statement 2 got a score of 2.6 (very close to neutral - 3), which indicates that a few of the students thought it was hard to pay attention to everything that was going on in the game. A reason for this may be that it was the first time they played the game. Unfortunately the class only had time to play the game once, hence further evaluation of the game should be done in future work to investigate this statement.

Statement 6 shows that almost everyone needed help before they could start playing the game, even though many also thought it was easy to understand what to do. Based on the observations and the focus group, this is probably related to the difficulty level of tasks. Almost everyone needed help for their first task, mainly due to syntactical errors, hence they were not able to solve tasks before they got help.

The teacher very much liked the analytics that were given both before and after the game play. There were some additional wishes for more analytics, especially a better overall overview of the students during game play, in the form of a matrix. The authors agree that a matrix would in fact be a nice way to show an overview of the students' performance, and should therefore be considered for future work. The after-game analytics could also help with evaluating the students in Python programming. One of the requirements of the teachers from the interviews (Chapter 3), was that they wanted an evaluation tool to help them grade their students. Code Flip cannot assist them in setting the final grade for the whole course, but according to the teacher, it could assist them with setting parts of the grade.

10.5.2 Content

In regard of the content of the game, it was clear from both the observations and the focus group that the tasks were too hard for the students. Some students were very close to the correct solution for a task, but the task did not pass due to syntax errors. This would most likely improve if the game had been played right after they had been given the introduction to Python. Since it was several months since the students had worked with textual programming, they struggled with remembering both concepts and syntax. Future user tests should therefore be performed right after an introduction to Python.

10.5.3 Concept

The concept of the game was very much approved by the teacher. The teacher thought that by having a game where the students competed, they became more engaged. It was also a motivational way of teaching programming. Additionally, the teacher would employ the game in the Python lecture for the next year's class, given that it was released. The students also agreed that the game is suitable for students of the ECP, and that it should be used in the course.

Although this feedback suggests that the game has good presumptions for being used in the ECP, it is still important to remember that it comes from only one teacher and one class with 14 students. To strengthen the results the game should be tested by more teachers and classes.

The students' overall perception of the game was good, as shown in Table 10.1. Most of the students thought the game was fun, motivating, and they would like to play the game again. The highest average between the statements in the table was given for statement 10 - *I was motivated because I wanted to win*. This suggests that the motivational element of competition worked very well for motivating the students. The element of cooperation also succeeded in motivating students. Even if some of the teams did not cooperate at all, only one student disagreed and two students were neutral to the statement of being motivated by being part of a team. Hence 78.6% of the students were motivated from being part of a team. Observations by both the authors and the teacher indicated that the goal of guessing the picture also worked well as motivation for solving tasks. This would suggest that the game managed to incorporate requirement **HR18** even though it was not prioritized, and that the motivational purpose for solving task was to uncover more pieces of the picture.

The research questions of this thesis focus on the motivational part of the game, as it would be demanding to evaluate the actual learning outcome of the game, and since the game is meant to be used in an elective course aiming at increasing students' motivation for school. However, to get an idea of what the students think about the increase in programming knowledge by playing the game, statement 14 attempts to answer this. As stated, most students felt like they got a better understanding of programming after playing the game. Additionally, statement 12 shows that the game helped with troubleshooting their code, which may also indicate a learning outcome. However, the learning outcome of the game is not properly evaluated. This will be recommended for future work with Code Flip.

10.5.4 Code Flip's Potential in other LSS Courses

As an important objective of Code Flip is to engage students in programming, and not just teach content of the ECP, the authors thought it would be interesting to test the game in classes of mathematics, to see how the game could be adapted for other courses. The mathematics students were not expected to be confident with programming concepts, nor with Python syntax. In order to adapt the game for those classes, a new set of tasks was defined, inspired by the content of Lokus which they had already been through with their teacher. The tasks involved writing code for computing mathematical problems like the mean of a list of numbers, or the area of a triangle. To help the students with syntax, hints were included in the task descriptions. A hint could for instance be how to write exponents in Python if an exponent was needed for solving the task. Mathematical formulas were listed in some of the tasks, that could be directly translated to Python by the student. Most tasks also contained code skeletons, so that the students did not have to know how to define a function, but could insert their calculations where indicated.

As already mentioned, the evaluations with the classes of mathematics were not conducted. However, the little work that was needed to adapt the game for these classes suggests that the game can easily be used by other courses than the ECP as well. The tasks of the game are loosely connected to the game itself, so it is easy to create new sets of tasks - targeting certain fields. The tasks also have hints that can be customized depending on the students skills in programming. Evaluations of the game in other courses would be needed in order to conclude with the game's applicability in other courses. The interest for it among LSS teachers should also be surveyed. The mathematics teacher of the planned evaluations was positive which may indicate a potential, but the matter has not been discussed with other teachers.

Chapter 11

Conclusions

The main contributions of this master thesis are: the teachers' perspectives of the Elective Course of Programming and their thoughts about a game that will support them during lectures (Chapter 3), a finalized list of high-level requirements for a game that teaches basic programming from both student and teacher perspectives (Chapter 4), an analysis of what game elements teachers prefer as part of an educational game for the ECP (Section 5.2), and an educational programming game designed for the ECP - Code Flip - as described in Chapter 6.

11.1 Research Questions

The research questions presented in Chapter 1 formed the foundation for this research. In this section, each of the questions will be presented with an answer.

> **RQ1:** How can we make a web-based game that teaches programming concepts to Norwegian lower secondary school students, in the context of the Elective Course of Programming?

This research aims at creating a game that teaches programming concepts to Norwegian lower secondary school students, in the context of the Elective Course of Programming. The work had already been initiated through a specialization project, where focus was on the students' perspectives through a systematic literature review, examination of the state of the art, and a workshop with lower secondary school students. It was found that children are motivated through games by elements of competition, cooperation and playfulness. It was further suggested that two-fold learning, specific learning goals and avoidance of having to remember a specific syntax, are effective methods for learning programming to children.

In this thesis the focus shifted towards the teachers of the Elective course of Programming. Their perspectives were collected through interviews and surveys. Combining all findings so far, a list of high-level requirements was proposed. Based on this, three game concepts were suggested. After comparing and evaluating these in terms of the interests of teachers and students of the ECP, one concept was chosen for implementation. This was implemented in an agile way, continuously assessing how it met with the game requirements.

Code Flip is a web-based game targeting the ECP, where students compete in teams and solve programming tasks to uncover a picture. The game has two goals, where the first is to guess what the picture displays and the second is to solve all the tasks of the board. The programming in the game is performed by the student through an integrated editor accepting

Python code. The game comes with a set of predefined tasks based on learning content seen to be widely used in the ECP. The programming tasks are highly independent of the game concept, and there is hence room for teachers to define new tasks, to accommodate teachers with varying programming competence.

For an educational game to be successful, it will have to be useful to the teachers, as they are the ones who decide whether to use it in the course. Interests of teachers for the game included that it should make it possible to monitor the students' progress, that it had clear learning goals, and that it should encourage problem solving. The teacher part of Code Flip was evaluated by a teacher student, and by a teacher in a classroom setting. Both thought the analytics offered by the game made it easy to follow the progress of students, and claimed that it made it possible to use the game as an aid when evaluating students. The usability of the teacher view was perceived as good. The teacher who tested the game in his classroom saw that several students consulted each other and the web for help, which suggests that the game encourages problem solving.

The main evaluation of Code Flip suggests that it serves its purpose of motivating students and making them more engaged in programming. Seeing that the ECP is an elective course, and the main purpose of elective courses is to motivate students instead of measuring a big learning outcome, motivation was more emphasized than learning outcome when designing, developing and evaluating Code Flip. However, the evaluations also suggest an increase in confidence for programming, and most students thought the game was helpful with troubleshooting their code, which does suggest a learning outcome.

> **RQ1.1:** What challenges do teachers of the Elective Course of Programming face when teaching programming to their students?

In order to create a game that teachers of the ECP will find useful, their challenges when teaching the course had to be addressed. Interviews with 13 different teachers of the ECP, as described in Chapter 3, were held in an attempt to answer this research question. The interviews indicate that several teachers face the same challenges when they teach the ECP, mainly connected to the motivation of students, and varying levels of programming skills among both students and teachers.

A lot of the students do not know what they sign up for when they choose the course, and hence lose motivation when they find out that the course is not about playing video games or designing complex systems. Since programming is a demanding subject to learn, and requires a way of approaching problems that students have not been taught through other school courses, several students struggle with patience in the programming courses. The varying degree of motivation among students, causes for a big gap in the students' programming skills, as the most eager students spend a lot of time at home programming on their own initiative. Hence, a struggle arises when the teacher is to create a teaching plan for the course that should accommodate all students.

It is seen from the interviews that teachers of the ECP are often employed due to their interest for programming, and that their programming competence varies a lot. This is supported by the problem elaboration of Chapter 2. As a consequence, some teachers end up in a situation where their most motivated students have more competence than themselves. This is hard both when teaching students, and when evaluating them. Some teachers also find knowledge transfer hard for the less motivated students, as students seem to understand what is taught by the teacher, but struggle with solving tasks independently, and give up quickly when they face resistance.

RQ1.2: What do teachers of the Elective Course of Programming wish for in a game that teaches programming?

The teacher interviews were also conducted to target **RQ1.2**, and numerous desires were gathered for a game that teachers programming. These were formed into a list of high-level requirements, where some overlapped with high-level requirements already gathered from students through the specialization project. Teachers' wishes for the game includes being able to monitor their students' progress, that it should encourage problem-solving, offer immediate feedback, and have clear learning goals. It was also uttered that the game should focus on textual programming, as there already exist several tools for teaching block-based programming, and textual programming is often harder for students to grasp. The full list is seen in Table 3.4.9.

During the LKK conference, participating teachers were asked to answer a questionnaire to prioritize different game elements gathered from interviews with teachers, and a focus group with students during the specialization project. The elements were related to both motivation and knowledge, as these were seen to be challenges the teachers face in the course. The elements of cooperation in pairs, cooperation in groups, competition, opportunity for students to help each other, opportunity to get hints, as well as graphics were assessed through the questionnaire, in relation to motivation and learning outcome. All but graphics were seen to be highly important to the teachers, and especially the opportunity for students to help each other.

RQ1.3: By addressing the interests of teachers and students, can we make a game for the Elective Course of Programming that increases LSS students' motivation for programming?

Code Flip was evaluated in a class of the ECP in a lower secondary school (n=14). It seems that the concept of the game succeeded in motivating the students for programming. Results from observations, questionnaires and a focus group all suggest that the students enjoyed playing the game, were motivated through competing in teams with their classmates in order to guess the picture, and felt more confident with programming after playing. Statements from the questionnaire related to these topics all have means between 4.1 and 4.5 on a scale from 1 to 5. "*I was motivated because I wanted to win*" had the highest mean score of 4.5, which confirms that competition is an effective motivational element in educational games. This was already seen in the literature (Chapter 2), focus groups with LSS students during the specialization project, and interviews and questionnaires with teachers through this thesis, and was the reason for including competition in the game in the first place. The students of the focus group after the classroom evaluation stressed that they enjoyed being part of a team, especially when being teamed up with friends.

The usability and content of the game however, did not seem to be as successful. Some students found it difficult to pay attention to everything that was going on in the game (mean 2.6 - where 1 is easy and 5 is difficult), and a few stated that they found the game cumbersome to play, with a mean of 2.3. Most students claimed to need help before they could start playing the game, which can be related to either usability or content. Observation and utterings from the focus group stating that they had forgot Python syntax and thought the tasks were hard, suggest that it is most likely related to the content of the game.

Seeing that the content is not tightly connected to the game concept, and that the students clearly seemed to be motivated from playing the game, Code Flip may be successful in motivating students for programming. However, 14 participants are not enough to conclude or assume trends for ECP students. The game should be evaluated in more classes of the ECP and in different schools for further evaluation in order to make generalizations about the results. Evaluations in another school was scheduled for this thesis, but were cancelled only two days in advance when it was too late to schedule other evaluations.

11.2 Strengths and Limitations of the Work

This section discusses strengths and limitations of the work performed through this master thesis. They are related to the game design, evaluations, and data gathering.

11.2.1 Game Design

The main contribution of this work is the educational programming game Code Flip. A lot of effort was therefore put into the design process of the game. Both the interests of students and teachers were thoroughly investigated before game concepts were proposed. A list of high-level requirements for an educational programming game was constructed early on, first addressing just the literature, ECP curriculum, and LSS students' game preferences. Through several iterations covering an increasing amount of the target group, the high-level list of requirements was assessed and modified. Accordingly, a game concept was formed with help from those who will be its end users. This would probably increase the likelihood of its success.

The implementation of Code Flip was also emphasized. Architecture and technologies were carefully chosen, ensuring that the full focus could be directed at the game logic and user experience, avoiding spending time on infrastructure. The development was performed in an agile way. The game was continuously assessed according to the game requirements, and it was evaluated by peer students, as well as experts on the way. Consequently, the version of the game that was evaluated in the classroom, was not the initial one, but a robust version where full focus could be directed at evaluating its concept and content. The game was almost ready for a possible release at that point. This helped the authors and the game in being perceived as professional, and may have helped with collecting serious answers to the evaluation.

The authors could have involved students and teachers even more in the design processes, to further ensure that the game met with the interests of the end users. For instance, students could have been involved in the design and/or election of the different game concepts. However, this would have required more time, and could have impacted either the completeness of the game, or avoided time for evaluations in a classroom setting. Seeing that the evaluation that was held received positive results regarding the game concept, more classroom evaluations would probably be worth more than additional involvement in the design process.

The main evaluation of Code Flip happened in the context it designed to be used in - a class of the ECP led by its teacher. This is a strength, as it is a natural setting for both the students and teacher, and should as such give quite accurate results. However, the presence of the authors could in itself have affected how the ECP students acted during the test.

The Context of Code Flip

Code Flip was designed specifically for use in the ECP. However, seeing that the curriculum of the ECP is vaguely specified and is likely to change in the years to come, Code Flip was designed in a way that the content is loosely connected to its concept. It is as such easy to modify the tasks of the game. As of now, it must be done by directly adding data to the database, however only the creation of a simple user interface is required for making it easy to add new content to the game. This flexibility is a strength which opens for a lot of possibilities. It allows for teachers to define their own tasks for the game, tailoring the content to the level of their students. It makes it easy to define new sets of tasks universally available to all users of the game, supporting those teachers who are not confident with defining new programming content.

Another possibility was identified by the teacher with whom the authors had planned to hold additional classroom evaluations. The mathematics classes of this teacher had been introduced to Python as a tool for solving math problems, and the teacher thus suggested that the game was tested with those students as well. Seeing that programming will be introduced to most mandatory LSS courses by 2020, among them mathematics, it would be highly interesting to evaluate the game's applicability in these courses. The authors prepared for an evaluation with these classes by adding new programming tasks to the game. The tasks were inspired by the content the mathematics classes had already been through with their teacher, and focused more on using Python as a tool for solving math problems, than for learning programming concepts and Python syntax. Unfortunately, these evaluations were cancelled last minute. It would however be recommended to perform such evaluations in the future. 2020 is not far away, and many teachers of courses like mathematics, social studies, natural sciences and music, are probable to need supporting tools in their teaching if they are not confident with programming themselves.

11.2.2 Data Gathering

Both through the process of designing the game concept, and when evaluating the game, data was gathered in multiple ways. Qualitative and quantitative data from observations, questionnaires, interviews and focus groups was triangulated in order to enhance the validity of findings.

A limitation of the data gathering is the population size. It was small, especially in the cases where quantitative data was gathered - the survey at the LKK conference, and the main evaluation where students played the game. Some of the findings from the LKK survey could indeed be triangulated with the qualitative data from the teacher interviews. They were however not significant enough alone to be seen as a general representation of what game elements teachers of the ECP find important, but as most of the individual survey answers coincided, they did help give an indication of what should be focused on in the game. The same goes for the game evaluation in the classroom - most answers coincided with each other, but the population size of 14 does not amount to much in relation to the total amount of ECP students in Norway.

11.3 Recommendations for Future Work

The most important part of future work for this master thesis is further evaluation of Code Flip. More evaluations in several schools and with different teachers are needed in order to make any generalizations about the applicability of the game. During these evaluations, the game should be played multiple times by each class. Additionally, it would be interesting to test the learning outcome of the game, as many of the students claimed their confidence for programming increased by playing.

From the three evaluations of Code Flip - the pilot evaluation, the expert evaluation and the main evaluation - the authors received many ideas of additional features that could be beneficial to implement in the game. The most important feature here is making a user interface for teachers to add additional tasks. Adding peer support - letting students ask each other for help and thus be rewarded with bonus points - could be a good solution if one team struggles with a task, as the finished teams can continue gaining points by helping the others. Implementing *Combo* also might avoid the rapid clicking on different task, as observed during the main evaluation, and make the students more eager to solve all the tasks they select. A list of suggested new features are shown in Table 11.1.

Feature
As a teacher, I want to be able to add new tasks.
As a teacher, I want to be able to add new pictures.
As a teacher, I want to see an overview of the students' performance during a game in the form of a matrix.
As a teacher, I want the boxes with student names to be dynamically rearranged when dividing students into teams.
As a teacher, I want to be able to edit the names of an existing classroom.
As a student, I want the boards to be bigger so that more tasks can be solved, and the picture is harder to guess.
As a student, I want to be awarded with bonus points when solving multiple tasks in a row (combo).
As a student, I want to be able to ask other students for help, regardless of team.
As a student, I want to be awarded with bonus points from helping other students.
As a student, I want to be able to see which tasks my teammates are currently working on.
As a student, I want to be able to see the other teams' performance.
As a student, I want to be able to open a task even if it is solved, in order to see the solution.

Table 11.1: Additional features for future work.

11.3.1 Actions Needed before a Release of Code Flip

Code Flip is a fully-functioning prototype and almost ready for a release. However, there are some additional actions that must be taken before a release. Since the master thesis had a strict time constraint, and the development had to be done in two months, some simplifications were made. The database with the game data was set to open. Some of the fields should have more restrictions so that players cannot freely manipulate the game data. Additionally, the authors used a free plan on Firebase for hosting the database. A release of Code Flip would require the authors to follow this up, so that the plan does not get exceeded.

Execution of code submitted by a user comes with a big security risk. This was assessed and taken into account during the architecture design, and throughout the implementation, by isolating the Python evaluation service from the rest of the system, and by restricting the availability of the executed code. However, further security evaluations should be performed before a release of the game, preferably by consulting a software security expert.

The source code of Code Flip is available on GitHub[1]. It is licensed under the MIT license, and as such everyone is free to contribute to further development the game, make adjustments, or to add the suggested features.

[1]`https://github.com/annikh/CodeFlip`

Bibliography

Bashir, Faizan (2018). *What is Serverless Architecture? What are its Pros and Cons?* URL: https://hackernoon.com/what-is-serverless-architecture-what-are-its-pros-and-cons-cc4b804022e9 (visited on May 11, 2019).

Berland, Matthew et al. (2011). “Programming on the move: Design lessons from IPRO”. In: *CHI’11 Extended Abstracts on Human Factors in Computing Systems*. ACM, pp. 2149–2154.

Brooke, John (1996). “SUS-A quick and dirty usability scale”. In: *Usability Evaluation in Industry*. Ed. by Patrick W. Jordan et al. CRC Press. Chap. 21, pp. 189–194.

Byrne, Pat and Gerry Lyons (2001). “The effect of student attributes on success in programming”. In: *ACM SIGCSE Bulletin*. Vol. 33. 3. ACM, pp. 49–52.

Connolly, Thomas M et al. (2012). “A systematic literature review of empirical evidence on computer games and serious games”. In: *Computers & Education* 59.2, pp. 661–686.

Corneliussen, Hilde and Fay Tveranger (2018). “Programming in Secondary Schools in Norway: A Wasted Opportunity for Inclusion”. In: *Proceedings of the 4th Conference on Gender & IT*. GenderIT ’18. Heilbronn, Germany: ACM, pp. 175–182. ISBN: 978-1-4503-5346-5. DOI: 10.1145/3196839.3196867. URL: http://doi.acm.org/10.1145/3196839.3196867 (visited on Sept. 14, 2018).

Dæhlen M. og Eriksen, I. M. (2015). *”Det tenner en gnist” Evaluering av valgfagene på ungdomstrinnet*. URL: http://hioa.no/content/download/96511/1975976/file/NOVA-Rapport-2-15-Evaluering-av-valgfagene-nett.pdf (visited on Sept. 18, 2018).

Grandell, Linda et al. (2006). “Why complicate things?: introducing programming in high school using Python”. In: *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*. Australian Computer Society, Inc., pp. 71–80.

Hevner, Alan R (2007). “A three cycle view of design science research”. In: *Scandinavian journal of information systems* 19.2, p. 4.

Holst, Anniken and Marianne Magnussen (2018). “Games for Programming: Targeting the Elective Programming Course of Norwegian Lower Secondary Schools”. Available upon request.

Jackson, Mike (2017). *Executing Python code submitted via a web service*. URL: https://software.ac.uk/blog/2017-11-23-executing-python-code-submitted-web-service (visited on May 11, 2019).

Jenkins, Tony (2002). “On the difficulty of learning to program”. In: *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*. Vol. 4. 2002. Citeseer, pp. 53–58.

Jørgenrud, Marius (2016). *Koding blir valgfagtil høsten. På noen ungdomsskoler*. URL: https://digi.no/artikler/koding-blir-valgfag-til-hosten-pa-noen-ungdomskoler/320523 (visited on Sept. 12, 2018).

Jovanov, Mile et al. (2016). "Computing as a new compulsory subject in the Macedonian primary schools curriculum". In: *Global Engineering Education Conference (EDUCON), 2016 IEEE*. IEEE, pp. 680–685.

Kaučič, B and T Asič (2011). "Improving introductory programming with Scratch?" In: *MIPRO, 2011 Proceedings of the 34th International Convention*. IEEE, pp. 1095–1100.

Koding, Lær Kidsa (2019). *Lærerkonferanse Trondheim*. URL: https://kidsakoder.no/konferanser/trondheim2019/ (visited on Mar. 14, 2019).

Majgaard, Gunver (2015). "Multimodal Robots As Educational Tools In Primary And Lower Secondary Education". In: *the 9th International Conference on Interfaces and Human Computer Interaction*, pp. 22–24.

Maloney, John H et al. (2008). *Programming by choice: urban youth learning programming with scratch*. Vol. 40. 1. ACM.

Martinez, Joe (2017). *What is Considered a Single Microservice?* URL: https://codeburst.io/what-is-considered-a-single-microservice-74cd6353b886 (visited on May 10, 2019).

Mayer, Richard E (2013). *Teaching and learning computer programming: Multiple research perspectives*. Routledge.

Meucci, Matteo and Andrew Muller (2019). *OWASP Testing Guide v5*. URL: https://github.com/OWASP/OWASP-Testing-Guide-v5 (visited on May 11, 2019).

Moser, Robert (1997). "A fantasy adventure game as a learning environment: why learning to program is so difficult and what can be done about it". In: *ACM SIGCSE Bulletin*. Vol. 29. 3. ACM, pp. 114–116.

Neagoie, Anderi (2018). *Top Libraries + Tech to Learn in 2019 for Full Stack Developers*. URL: https://medium.com/zerotomastery/top-libraries-tech-to-learn-in-2019-for-full-stack-developers-f8c0331b8a00 (visited on May 10, 2019).

Oates, Briony J (2005). *Researching information systems and computing*. Sage.

Papert, Seymour and Idit Harel (1991). "Situating constructionism". In: *Constructionism* 36.2, pp. 1–11.

Pears, Arnold et al. (2007). "A survey of literature on the teaching of introductory programming". In: *ACM sigcse bulletin*. Vol. 39. 4. ACM, pp. 204–223.

Regjeringen (2016a). *Koding blir valgfag på 146 skoler*. URL: https://regjeringen.no/no/aktuelt/koding-blir-valgfag-pa-146-skoler/id2481962/ (visited on Sept. 18, 2018).

— (2016b). *Skoleoversikt - Koding som valgfag*. URL: https://regjeringen.no/contentassets/2fa090ebfbd7427790ae5668ecc09fac/skoleoversikt---koding-som-valgfag2016.pdf (visited on Jan. 16, 2019).

— (2017). *Åpner for koding som valgfag ved alle skoler*. URL: https://regjeringen.no/no/aktuelt/fra-hosten-kan-alle-ungdomskoler-tilby-programmering-som-valgfag-til-elevene/id2552808/ (visited on Sept. 18, 2018).

— (2018). *Fornyer innholdet i skolen*. URL: https://regjeringen.no/no/aktuelt/fornyer-innholdet-i-skolen/id2606028/?expand=factbox2606082 (visited on May 14, 2019).

Robins, Anthony, Janet Rountree, and Nathan Rountree (2003). “Learning and teaching programming: A review and discussion”. In: *Computer science education* 13.2, pp. 137–172.

Roscoe, Jonathan Francis, Stephen Fearn, and Emma Posey (2014). “Teaching computational thinking by playing games and building robots”. In: *Interactive Technologies and Games (iTAG), 2014 International Conference on*. IEEE, pp. 9–12.

Schneider, G Michael (1978). “The introductory programming course in computer science: ten principles”. In: *ACM SIGCSE Bulletin*. Vol. 10. 1. ACM, pp. 107–114.

Sherman, Paul (2018). *How Single-Page Applications Work*. URL: `https://blog.pshrmn.com/entry/how-single-page-applications-work/` (visited on May 10, 2019).

Shim, Jaekwoun, Daiyoung Kwon, and Wongyu Lee (2017). “The Effects of a Robot Game Environment on Computer Programming Education for Elementary School Students”. In: *IEEE Transactions on Education* 60.2, pp. 164–172.

Shneiderman, B and C Plaisant (2010). “Evaluating Interface Designs”. In: *Designing the User Interface*. Addison-Wesley. Chap. 4, pp. 149–188.

Spieler, Bernadette et al. (2016). “The role of game jams in developing informal learning of computational thinking: a cross-European case study”. In:

Tangney, Brendan et al. (2010). “Pedagogy and processes for a computer programming outreach workshop—The bridge to college model”. In: *IEEE Transactions on Education* 53.1, pp. 53–60.

Tomcsányiová, Monika (2013). “Using computer games as programming assignments for university students and secondary school pupils”. In: *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer, pp. 91–102.

Tveranger, Fay (2017). “Programming in school. An insight to the Norwegian programming pilot and the inclucion/exclusion of girls in computer programming education”. MA thesis. The University of Bergen.

Utdanningsdirekoratet (2014). *PC i skolen, læremiddel eller hjelpemiddel*. URL: `https://udir.no/regelverkstolkninger/opplaring/Elever-med-sarskilte-behov/PC-i-skolen-laremiddel-eller-hjelpemiddel/` (visited on June 12, 2018).

— (2016a). *Forsøkslæreplan i valgfag programmering*. URL: `http://data.udir.no/kl06/PRG1-01.pdf` (visited on Sept. 18, 2018).

— (2016b). *Veiledning til programmering valgfag*. URL: `https://udir.no/Udir/PrintPageAsPdfService.ashx?pdfId=119907` (visited on Sept. 18, 2018).

Utdanningsdirektoratet (2018). *Valgfag på ungdomstrinnet*. URL: `https://udir.no/tall-og-forskning/statistikk/statistikk-grunnskole/valgfag-pa-ungdomstrinnet/` (visited on Sept. 18, 2018).

Verstad, Andrej Levin Tørkel (2017). “Programmering+ Matematikk= Sant? En casestudie om overføringsverdien mellom programmering valgfag og matematikkfaget”. MA thesis. UiT Norges arktiske universitet.

Von Alan, R Hevner et al. (2004). “Design science in information systems research”. In: *MIS quarterly* 28.1, pp. 75–105.

Appendix A

Information Email to Teachers

Emne: Masteroppgave - Ønsker kontakt med lærere i valgfaget programmering

Hei!

Vi er to studenter som skriver masteroppgave for NTNU - Institutt for Datateknologi og Informatikk dette semesteret, og ønsker i den forbindelse å komme i kontakt med lærere som underviser i valgfaget programmering. *Dersom skolen deres ikke tilbyr dette faget, kan dere se bort fra denne mailen.*

Kort om masteroppgaven:
Masteroppgaven tar for seg hvordan spill kan brukes for å lære bort programmering til barn i ungdomsskolealder. Vi vil derfor finne ut av hva slags ønsker lærere har for et slikt spill og hvordan de kan inkluderes i spillet.

En viktig del av oppgaven er også å se på hvordan faget blir undervist i dag, hvilke verktøy som benyttes, og hvordan det som helhet fungerer. Vi ser for oss å holde et intervju på ca 30 minutter med hver lærer, og forhåpentligvis kunne få til en fokusgruppe med flest mulig i løpet av vinteren/våren.

Vi håper dere kan hjelpe oss med å komme i kontakt med de riktige personene, og vi svarer gjerne på spørsmål om det skulle være noen.

Mvh
Anniken Holst
Marianne Magnussen

Appendix B

Interview Guidelines

Intro
Så hyggelig at du ville være med på intervju! Før vi begynner med spørsmålene, tenkte vi å presentere kort hva vi ønsker å få ut av intervjuet. Vi ønsker å finne ut hva slags bakgrunn lærere i programmering har, og litt om hvordan undervisningen foregår på ulike skoler. Målet vårt er å lage et spill som kan brukes i undervisningen av faget, og vi vil derfor også høre litt om dine tanker om slike spill.

Er det noe du lurer på før vi begynner?

Intervjuguide
Hvorfor har du valgt å undervise i valgfag i programmering?

Hvordan har du tilegnet deg kompetanse i programmering?

Hvilke andre fag underviser du i?

Hva er mest utfordrende når det gjelder å lære bort programmering til barn?
- Hvordan løser du det?

Hva slags ressurser og læringsaktiviteter benytter du i undervisningen?

Hvilke programmeringsspråk bruker du i undervisningen?

Hvordan organiserer du faget?

Omtrent hvor stor andel av elevene har tatt faget fra før?

Hvordan legger du opp faget for de som allerede har tatt det?

Bruker du noen spill i undervisningen?

Har du hørt om noen spill som lærer bort programmering?

Hva er din holdning til å bruke spill i undervisningen?

Hvordan ville du inkludert et spill i undervisningen? (Til intro av programmeringskonsepter, repetisjon, lekser, gruppeaktiviteter, konkurranse?)
- Har dere allerede noen gruppeaktiviteter eller konkurranser? Hva slags aktiviteter virker morsomst for elevene?

Hvilke konsepter er vanskeligst for elevene å forstå? (f.eks.)

Hvordan foregår vurdering av elevene i faget?

Til slutt: Har du noe mer å tilføye om faget?

Appendix C

Consent Forms

Vil du delta i forskningsprosjektet

"Games for Programming: Targeting the Elective Programming Course of Norwegian Lower Secondary Schools"?

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å undersøke hvordan spill kan brukes til å lære bort programmering i valgfaget programmering på norske ungdomsskoler. I dette skrivet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål
Forskningsprosjektet er knyttet til en masteroppgave ved Institutt for Datateknologi og Informatikk ved NTNU.

Formålet med prosjektet er å utvikle et spill som kan brukes i undervisningen i valgfaget programmering på ungdomsskolen. Masteroppgaven bygger videre på et fordypningsprosjekt hvor målet var å finne ut av hva slags spilltyper og spillelementer som passer for ungdomsskoleelever, og få et inntrykk av hvordan man best kan lære bort programmering til målgruppen.

I masteroppgaven vil det være mer fokus på lærere. Vi må finne ut av hva slags ønsker lærere har for et slikt spill og hvordan de skal inkluderes. I tillegg vil masteroppgaven involvere utviklingen av selve spillet.

Hvem er ansvarlig for forskningsprosjektet?
Forskningsprosjektet gjennomføres i forbindelse med en masteroppgave som skrives av Marianne Magnussen og Anniken Holst, og veiledes av professor Monica Divitini. Den ansvarlige institusjonen er Institutt for for Datateknologi og Informatikk.

Hvorfor får du spørsmål om å delta?
Lærere som underviser i valgfaget programmering kontaktes for å delta i prosjektet. De kontaktes via rektor eller kommunikasjonsansvarlig ved skolen gjennom kontaktinformasjon på nettsidene til skolene/kommunene. Siden kontakten foregår gjennom et mellomledd, vil lærere også rekrutteres gjennom konferansen "Lær Kidsa Koding" for å komme i kontakt med flest mulig.

Hva innebærer det for deg å delta?
Hvis du velger å delta i prosjektet, innebærer det at du svarer på et intervju. Det vil ta ca. 30 minutter å gjennomføre intervjuet. Spørsmålene vil handle om hvordan du underviser programmering, samt hvordan et spill vil kunne brukes i undervisningen av faget. Det vil bli tatt lydopptak fra intervjuet for å få med detaljer fra svarene.

Alle deltakere vil også bli spurt om å delta i en fokusgruppe hvor målet er å diskutere funnene fra prosjektoppgaven, se på eksisterende spill som skal lære bort programmering, og diskutere hvordan et sånt spill kan brukes i undervisningen. Fokusgruppen vil vare i omtrent to timer.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykke tilbake uten å oppgi noen grunn. Alle opplysninger om deg vil da bli anonymisert. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger
Vi vil bare bruke opplysningene om deg til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket.

Lydopptakene og data fra intervju og fokusgruppe vil kun være tilgjengelige for deltakende forskere i prosjektet, Marianne Magnussen og Anniken Holst. Dataene vil bli fullt anonymisert ved at navn og kontaktopplysninger erstattes med en kode. I tilfelle av vitenskapelig publisering av prosjektresultatet, vil ikke deltakerne være gjenkjennelige i utgivelsen. Prosjektet skal etter planen avsluttes 1. juni 2019. Etter ferdigstillelsen vil all data bli slettet.

Dine rettigheter
Så lenge du kan identifiseres i datamaterialet, har du rett til:
- innsyn i hvilke personopplysninger som er registrert om deg,
- å få rettet personopplysninger om deg,
- få slettet personopplysninger om deg,
- få utlevert en kopi av dine personopplysninger (dataportabilitet), og
- å sende klage til personvernombudet eller Datatilsynet om behandlingen av dine personopplysninger.

Hva gir oss rett til å behandle personopplysninger om deg?
Vi behandler opplysninger om deg basert på ditt samtykke.

På oppdrag fra NTNU Norges teknisk-naturvitenskapelige universitet / Institutt for Datateknologi og Informatikk har NSD – Norsk senter for forskningsdata AS vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Hvor kan jeg finne ut mer?
Hvis du har spørsmål til studien, eller ønsker å benytte deg av dine rettigheter, ta kontakt med:
- Institutt for Datateknologi og Informatikk ved masterstudent Anniken Holst (tlf: [redacted], email: [redacted]) eller Professor Monica Divitini (tlf: [redacted], email: [redacted])
- NSD – Norsk senter for forskningsdata AS, på epost ([redacted]) eller telefon: [redacted].

Samtykkeerklæring
Jeg har mottatt og forstått informasjon om prosjektet *Games for Programming: Targeting the Elective Programming Course of Norwegian Lower Secondary Schools*, og har fått anledning til å stille spørsmål. Jeg samtykker til:

- □ å delta i intervju
- □ å delta i fokusgruppe

Jeg samtykker til at mine opplysninger behandles frem til prosjektet er avsluttet, ca. 01.06.2019.

(Signert av prosjektdeltaker, dato)

Appendix D

LKK Conference Survey

Bruk av spill i valgfaget programmering

Kunne du tenke deg å hjelpe oss med å utvikle vår spillidé? Vi vil holde en fokusgruppe med lærere i valgfaget programmering for å få tilbakemeldinger og innspill. Vi trenger også lærere som kan teste ut spillet vårt i en skoletime.

Dersom du er interessert i å hjelpe oss med noe av dette, send en mail til annikenholst@gmail.com.

1. **Har du brukt spill i undervisningen, i så fall hvilke?**

Under følger to påstander. Oppgi hvor enig du er i dem på en skala fra 1 (helt uenig) til 7 (helt enig).

2. **Et spill kan brukes til å øke kunnskap i programmering.**
   Markér bare én oval.

	1	2	3	4	5	6	7	
Helt uenig	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Helt enig

3. **Et spill kan brukes til å øke engasjement for programmering.**
   Markér bare én oval.

	1	2	3	4	5	6	7	
Helt uenig	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Helt enig

På spørsmålene under skal du prioritere elementene fra 1 (minst viktig) til 6 (viktigst)

4. **Hva slags støtte ønsker du å få i et spill for å lære bort programmering?**
   Merk av for alt som passer

- ☐ Velge innhold i oppgaver basert på tema (f.eks. å ha oppgaver kun relatert til variabler)
- ☐ Velge innhold i oppgaver basert på vanskelighetsgrad
- ☐ Kunne definere nye oppgaver selv
- ☐ Få statistikk om elevenes resultater underveis i spillet
- ☐ Få statistikk om elevenes resultater i etterkant
- ☐ Annet forslag:

5. **Hvilke av elementene mener du er nyttige for å øke kunnskap i programmering?**
   Merk av for alt som passer

- ☐ Samarbeid i par
- ☐ Samarbeid i grupper
- ☐ Konkurranse
- ☐ Mulighet for å hjelpe hverandre
- ☐ Mulighet for å få hint hvis man står fast
- ☐ Grafikk

6. **Hvilke av elementene mener du er nyttige for å øke engasjement for programmering?**
   Merk av for alt som passer

- ☐ Samarbeid i par
- ☐ Samarbeid i grupper
- ☐ Konkurranse
- ☐ Mulighet for å hjelpe hverandre
- ☐ Mulighet for å få hint hvis man står fast
- ☐ Grafikk

Appendix E

Predefined Task Sets of Code Flip

E.1 Easy Mix

Table E.1 shows the the title and description of tasks that are part of the predefined task set *Easy Mix*. For the tasks with a code skeleton, this is listed together with the task description for simplicity.

Title	Description

Variabel 1	Default code:
	`navn = "Pusur"`
	`print("Hei,", navn)`
	Koden i editoren printer nå `"Hei, Pusur"`. Endre på variabelen `navn`, slik at programmet printer: `"Hei, [navnet ditt]"`

Variabel 2	Lagre navnet ditt i en variabel kalt `navn`, og print den til skjermen.

Lister 1	Code skeleton:
	`godteri = ["Stratos","Potetgull","Lakris"]`
	`print(godteri)`
	`#Din kode her`
	Listen `godteri()` inneholder nå 3 elementer. Skriv kode slik at det legges til et nytt element bakerst, og print listen på nytt.

Lister 2	Code skeleton: `sport = ["Ski","Fotball","Turn","Basketball","Svømming","Karate"]` Hent ut elementet `"Karate"` fra listen, og print det.
Lister 3	Opprett en liste med navn `tall` som inneholder tallene 1, 2, 3, og 4, og print ut det tredje elementet i listen.

If & else 1	Code skeleton:
	`sport = ”Langrenn”`
	`if(sport == ”Langrenn”):`
	`    print(”Johannes␣Høsflot␣Klæbo␣er␣verdens␣`
	`        beste␣skiløper.”)`
	`else:`
	`    print(sport, ”er␣gøy!”)`
	Gjør ferdig koden slik at det printes `"Turn er gøy!"`

If & Else 2	Opprett to variabler: `lekser_ferdig`, og `rommet_ryddet`, og sett dem til `True` (sant). Lag en if-setning som har følgende kriterier: Hvis `lekser_ferdig` og `rommet_ryddet` er lik `True`, skal det printes `"Jeg er flink!"`. Hvis `lekser_ferdig` er `True` og `rommet_ryddet` er `False`, skal det printes `"Jeg må rydde rommet"` Hvis `lekser_ferdig` er `False` og `rommet_ryddet` er `True` skal det printes `"Jeg må gjøre lekser"` Hvis begge variablene er `False` (usant), skal det printes `"Jeg må gjøre lekser og rydde rommet`
If & else 3	Opprett en variabel med navn `sport`, og sett den lik favorittsporten din. Lag en if-setning som sjekker om variabelen sport er lik favorittsporten din. I så fall skal det printes: `"[sport], er min favorittsport!"`. Hvis den ikke er lik skal det printes: `"[sport], er bra, men ikke best"`
If & else 4	Opprett en variabel med navnet `alder`. Sett den lik et tall mellom 18 og 120. Lag en if-setning som sjekker om alderen er over 18 år, og under 120 år. Hvis den er sann skal det printes `"Du er voksen!"`
For-løkker 1	Code skeleton: `farger = [”blå”,”rosa”,”gul”,”grønn”,”lilla”]` Lag en for-løkke som printer ut alle elementene i listen.
For-løkker 2	Opprett en liste som inneholder tallene fra 1 til og med 9. Bruk så en for-løkke for å gå gjennom listen og summere alle tallene. Print resultatet.

While 1	Code skeleton: `antall_kjeks = 12` Lag en while-løkke som fjerner én og én kjeks, og for hver gang printer antall kjeks som er igjen. Når det ikke er flere kjeks igjen, skal det printes `"Det er tomt for kjeks!`
Funksjoner 1	Code skeLag en funksjon med navn `hei()` som tar inn et navn og returnerer det.
Funksjoner 2	Code skeleton: `def velkommen():` `print("Velkommen␣hjem␣til␣meg")` `#Din kode her` Hvis du prøver å kjøre koden skjer det ingenting. Endre koden slik at det som skjer i `velkommen()` blir utført.
Funksjoner 3	Lag en funksjon som heter `favorittfarge(farge)`, som tar inn en farge og printer `"Min favorittfarge er [farge]"` Kall på funksjonen med en farge slik at funksjonen printes. Eksempel på kjøring: `favorittfarge("rød")` printer `"Min favorittfarge er rød"`
Funksjoner 4	Code skeleton: `def aldersgrense(alder, aldersgrense):` `#din kode her` `print(aldersgrense(10,16))` `print(aldersgrense(18,16))` Fullfør funksjonen. Hvis `alder` er mindre enn `aldersgrense`, skal funksjonen returnere antall år det er til alder er lik aldersgrense. Hvis alder er større enn aldersgrense, skal funksjonen returnere `True`. Eksempel på kjøringer: `aldersgrense(10,16)` skal returnere `6` `aldersgrense(16,16)` skal returnere `True` `aldersgrense(18,16)` skal returnere `True`

Table E.1: The task set *Easy Mix*

E.2 Functions

Table E.2 shows the title and description of tasks that are part of the predefined task set *Functions*. For the tasks with a code skeleton, this is listed together with the task description for simplicity.

Title	Description
Funksjoner 4	Opprett en funksjon kalt `addisjon()`, som tar inn to tall. Funksjonen skal plusse tallene sammen, og returnere svaret.

Funksjoner 5	Code skeleton:
	`tall = [1, 2, 3, 4, 5]`
	`def summen(tall):`
	`    #din kode her`
	
	`summen(tall)`
	Fullfør funksjonen `summen()` slik at den summerer tallene i listen `tall`, og returnerer summen.

Funksjoner 6	Lag en funksjon med navn `partall`, som skal ta inn et tall. Dersom tallet som tas inn er et partall, skal funksjonen returnere `True`. Hvis det er et oddetall skal den returnere `False`
	Eksempel på kjøring:
	`partall(2)` skal returnere `True`
	`partall(5)` skal returnere `False`

Funksjoner 7	Code skeleton
	`def ord_som_begynner_på(liste , bokstav):`
	`    #din kode her`
	
	`print(ord_som_begynner_på(["ball","tre","boller"],"b"))`
	Fullfør funksjonen. Funksjonen `ord_som_begynner_på()` skal ta inn en liste med ord og en bokstav. Funksjonen skal returnere en ny liste med ordene som begynner på bokstaven.
	Eksempel på kjøringer:
	`ord_som_begynner_på(["ball","tre","boller"],"b")` skal returnere `["ball","boller"]`
	`ord_som_begynner_på(["ball","tre","boller"],"t")` skal returnere `["tre"]`
	`ord_som_begynner_på(["ball","tre","boller"],"a")` skal returnere `False`

Funksjoner 8	Lag en funksjon med navn `rabattpris()` , som skal ta inn et beløp og en prosent. Funksjonen skal returnere den nye prisen, ved å trekke prosenten fra summen. Eksempel på kjøringer: Du finner en genser som koster 500 kr, og som er på 40% salg, `rabattpris(500, 40)` skal returnere `300`.

Funksjoner 9	Code skeleton:
	```
	def summen(tallListe):
	    sum = 0
	    for tall in tallListe:
	        sum = tall
	    return sum

	print(summen([1, 2, 3, 4]))
	```
	Funksjonen `summen()` skal legge sammen tallene i listen, og returnere summen. Hvis du kjører programmet vil du se at feil tall returneres. Finn og løs feilen slik at den returnerer riktig sum.

Funksjoner 10	Code skeleton:
	```
	def hundeår(menneskeår):
	    #din kode her

	print(hundeår(4))
	```
	Fullfør funksjonen `hundeår()`. For å regne ut hvor gammel man er i hundeår, må man regne med 10,5 hundeår per menneskeår for de første to årene, deretter 4 hundeår per menneskeår for hvert påfølgende år.
	
	Eksempel:
	
	Hvis et menneske er 6 år, må man ta de to første årene og multiplisere med 10,5 år, som blir 21 år. Deretter tar man de fire siste årene og multipliserer med 4 år, som blir 16 år. Altså er et menneske på 6 år, 37 år (21+16) i hundeår.
	
	Eksempel på kjøringer:
	
	`hundeår(2)` blir `21`
	
	`hundeår(4)` blir `29`
	
	`hundeår(12)` blir `61`

Funksjoner 11	Opprett en funksjon med navnet ¡strong¿tips¡/strong¿, som tar inn et beløp og en prosent. Beløpet er summen på en restaurantkvittering, mens prosenten er andel man skal gi i tips. Funksjonen skal returnere hvor mye man skal gi i tips. Eksempel på kjøring:

	Hvis kvitteringen er på 100kr, og man skal gi 10% i tips, må man gi 10kr i tips: `tips(100, 10)` skal gi resultatet `10`.
Funksjoner 12	Opprett en funksjon med navn `reverser()`, som tar inn en liste, reverserer listen, og returnerer den. Eksempel på kjøring: `reverser(["banan", "eple", "kiwi"])` gir resultatet `["kiwi", "eple", "banan"]`
Funksjoner 13	Opprett en funksjon med navnet `fjern_ordene()`, som tar inn en liste med ord og en bokstav. Funksjonen skal fjerne alle ord som begynner på bokstaven og returnere den nye listen.
Funksjoner 14	Opprett en funksjon med navn `bokstavtelling()`, som tar inn en tekststreng og en bokstav. Funksjonen skal returnere hvor mange ganger bokstaven forekommer i tekstrengen. Tips: Ta hensyn til både små og store bosktaver!

Funksjoner 15	Code skeleton:
	```
	alfabet = "abcdefghijklmnopqrstuvwxyzæøå"
	def krypter(ord, nøkkel):
	    kryptert_ord = ""
	    for bokstav in ord:
	        posisjon = alfabet.find(bokstav.lower
	            ())
	        ny_posisjon = posisjon + nøkkel
	        #Hvis vi er i enden på alfabetet, må
	            vi kunne gå rundt og begynne på a
	            igjen
	        if(ny_posisjon >= 29):
	        ny_posisjon -= 29
	        kryptert_ord += alfabet[ny_posisjon]
	    return kryptert_ord

	def dekrypter(ord, nøkkel):
	    dekryptert_ord = ""
	    #din kode her
	    return dekryptert_ord

	kryptert = krypter("sjokolade", 2)
	dekryptert = dekrypter(kryptert, 2)
	print("Kryptert␣ord:", kryptert)
	print("Dekryptert␣ord:", dekryptert)
	```
	Fullfør `dekrypter(ord,nøkkel)`. Med kode kan man lage hemmelige tekster ved hjelp av kryptering. I denne oppgaven har du allerede fått en funksjon som krypterer et ord. Din jobb er å lage dekrypteringsfunksjonen.
	
	Krypteringen fungerer slik:
	
	Den tar inn et ord og en nøkkel. Nøkkelen forteller antall plasser man skal forflytte seg til høyre i alfabetet. Hvis man får inn ordet "Hei", med `nøkkel=1`, Skal bokstaven h bli til i, bokstaven e til f, og bokstaven i til j. Altså blir det krypterte ordet som returneres lik "ifj".

Funksjoner 16	Opprett en funksjon med navn `baklengs()` som skal ta inn et ord og returnere ordet baklengs. Eksempel på kjøring: `baklengs("potet")` gir resultatet "tetop"
Funksjoner 19	Opprett en funksjon med navnet `roper_du()`, som tar inn en tekst. Funksjonen skal sjekke om teksten bare inneholder store bokstaver, i så fall skal den returnere "Du roper!". Hvis ikke skal den returnere "Kan du snakke høyere?"

	Hint: Funksjonen `upper()` gjør om alle bokstavene i en tekststreng til store bokstaver. Hvis man har en variabel `tekst="hei"`,og skriver `tekst.upper()`, vil `tekst` endres til ”HEI”.
Funksjoner 18	Opprett en funksjon med navn `lengste_ord()`, som tar inn en liste med ord. Funksjonen skal returnere det lengste ordet i listen. Dersom det er flere ord på lik lengde, returneres det siste ordet.
Funksjoner 17	Opprett en funksjon som med navn `størst_tall()`, som tar inn en liste med tall. Funksjonen skal returnere det største tallet i listen.

Table E.2: The task set *Functions*

E.3 Mathematics

Table E.3 shows the title and description of tasks that are part of the predefined task set *Mathematics*. For the tasks with a code skeleton, this is listed together with the task description for simplicity.

Title	Description

Matematikk 1	Code skeleton:
	`def størst_tall(tall):`
	`    størst = 0`
	`    #din kode her`
	`    return størst`
	
	`print(størst_tall([1,2,3,4,5]))`
	
	Funksjonen `størst_tall` tar inn en liste med tall. Endre på koden slik at den returnerer det største tallet i listen.

Matematikk 2	Code skeleton:
	`def addisjon(tall1 ,tall2):`
	`    summen = 0`
	`    #din kode her`
	`    return summen`
	
	`print(addisjon(3,4))`
	
	Skriv kode slik at funksjonen `addisjon()` returnerer summen av de to tallene den tar inn.

Matematikk 3	Code skeleton:
	`def rabattpris(beløp,prosent):`
	`    ny_pris = 0`
	`    #din kode her`
	`    return ny_pris`
	
	`print(rabattpris(500,40))`
	Funksjonen `rabattpris()` tar inn et beløp og en prosent. Endre på den slik at den returnerer den nye prisen, ved å trekke prosenten fra summen.
	Eksempel på kjøringer:
	Du finner en genser som koster 500 kr, og som er på 40% salg:
	`rabattpris(500, 40)` skal returnere `300`.

Matematikk 4	Code skeleton:
	`def gi_tips(beløp,prosent):`
	`    tips = 0`
	`    #din kode her`
	`    return tips`
	
	`print(gi_tips(100,10))`
	Funksjonen `gi_tips()` tar inn et beløp og en prosent. Beløpet er summen på en restaurantkvittering, mens prosenten er andel man skal gi i tips. Endre på funksjonen slik at den returnerer hvor mye man skal gi i tips.
	Eksempel på kjøring:
	Hvis kvitteringen er på 100 kr, og man skal gi 10% i tips, må man gi 10 kr i tips:
	`gi_tips(100, 10)` skal gi resultatet 10.

Matematikk 5	Code skeleton:
	`for tall in range(1, 101):`
	`    if(tall % 7 == 0):`
	`    print(tall)`
	Programmet til venstre har en **for-løkke** som går fra og med `1` til og med `100`.
	For hvert tall, sjekker den om tallet er delelig på 7.
	Hvis tallet er delelig på 7, printer den tallet. (7-gangen)
	Skriv om koden slik at programmet printer tall fra 8-gangen.

Matematikk 6	Code skeleton:
	`def hundeår(menneskeår):`
	`    #din kode her`
	
	`print(hundeår(4))`
	Fullfør funksjonen `hundeår()`. For å regne ut hvor gammel man er i hundeår, må man regne med 10,5 hundeår per menneskeår for de første to årene, deretter 4 hundeår per menneskeår for hvert påfølgende år.
	Eksempel:
	Hvis et menneske er 6 år, må man ta de to første årene og multiplisere med 10,5 år, som blir 21 år. Deretter tar man de fire siste årene og multipliserer med 4 år, som blir 16 år. Altså er et menneske på 6 år, 37 år (21+16) i hundeår.
	Eksempel på kjøringer:
	`hundeår(2)` blir 21
	`hundeår(4)` blir 29`hundeår(12)` blir 61

Matematikk 7	Code skeleton: `antall_kjeks = 12` Lag en while-løkke som fjerner én og én kjeks, og for hver gang printer antall kjeks som er igjen. Når det ikke er flere kjeks igjen, skal det printes `"Det er tomt for kjeks!"`

Matematikk 8	Code skeleton:
	`def fahrenheitgrader(temperatur):`
	`    farhenheit=0`
	`    #din kode her`
	`    return fahrenheit`
	
	`print(fahrenheitgrader(20))`
	Funksjonen `fahrenheitgrader()` tar inn en temperatur (i celsiusgrader). Endre på koden slik at den gjør om til fahrenheitgrader og returnerer resultatet.
	Formelen for å gjøre om celsius til fahrenheit er: celsiusgrader * (9/5) + 32
	Eksempel på kjøring:
	`fahrenheitgrader(20)` returnerer `68.0`

Matematikk 9	Code skeleton:
	`def modulus(tall1 ,tall2):`
	`    rest=0`
	`    return rest`
	
	`print(modulus(20,8))`
	
	Funksjon `modulus()` tar inn to tall. Endre slik at den returnerer resten når man dividerer det første tallet på det andre.
	
	Eksempel på kjøring:
	
	`modulus(17, 5)` returnerer `2`
	
	`modulus(20, 8)` returnerer `4`

Matematikk 10	Code skeleton:
	`def hypotenusen(katet1 ,katet2):`
	`    hypotenus=0`
	`    return hypotenus`
	
	`print(hypotenus(5.2,4.7))`
	
	Funksjon `hypotenusen()` tar inn to tall (`katetA` og `katetB`). Endre slik at den returnerer hypotenusen.
	
	Hint: `hypotenus^2 = katetA^2 + katetB^2`
	
	For å ta kvadratroten av et tall kan man bruke: `tall**0.5`
	
	Eksempel på kjøring:
	
	`hypotenusen(5.2, 4.7)` skal returnere `7.08`

Matematikk 11	Opprett en funksjon med navnet `median()`, som tar inn en liste med tall og returnerer medianen.
	
	Dersom listen har et partall antall tall, skal den returnere gjennomsnittet av de to tallene i midten.
	
	Hint: Liste-indekser må være heltall. Når man deler på noe, gjør python om tallet til en float. Bruk `int(tall)` for å gjøre om til int igjen.
	
	Eksempel på kjøring:
	
	`median([0, 9, 5, 8, 6, 2, 9])` returnerer `6`
	
	`median([1, 3, 5, 7])` returnerer `4`

Matematikk 12	Opprett en funksjon med navnet `delelig_på()` som tar inn to tall. Funksjonen skal sjekke om det første tallet er delelig på det andre tallet. Hvis tallene er delelige skal den returnere resultatet. Hvis tallene ikke er delelig skal funksjonen returnere `False`

	Eksempel på kjøring:
	`delelig_på(10, 2)` skal returnere `5`
	`delelig_på(14, 3)` skal returnere `False`

Matematikk 13	Opprett en funksjon med navnet `gjennomsnitt()`, som tar inn en liste med tall og returnerer gjennomsnittet av tallene. Eksempel på kjøring: `gjennomsnitt([1, 2, 3, 4, 5, 6])` skal returnere `3.5`
Matematikk 14	Opprett en funksjon med navnet `trekant_areal()`. Funksjonen skal ta inn to tall: `grunnlinje` og `høyde`, og returnere arealet av trekanten. Eksempel på kjøring: `trekant_areal(20, 6)` skal returnere `60`

Matematikk 15	Lag en funksjon med navn `partall`, som skal ta inn et tall. Dersom tallet som tas inn er et partall, skal funksjonen returnere `True`. Hvis det er et oddetall skal den returnere `False`
	Eksempel på kjøring:
	`partall(2)` skal returnere `True`
	`partall(5)` skal returnere `False`

Matematikk 16	Code skeleton:
	`x = 1`
	`while x < 10:`
	`print(x)`
	`x = x + 1`
	Skriv om koden slik at den skriver ut tallene fra og med 1 til og med 30

Table E.3: The task set *Mathematics*

Appendix F

Student Questionnaire for the Main Evaluation of Code Flip

Spørreskjema

Brukbarhet

1. **Jeg kunne tenkt meg å spille spillet igjen**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

2. **Jeg synes det var vanskelig å følge med på alt som skjedde i spillet**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

3. **Jeg synes det var enkelt å forstå hva jeg skulle gjøre**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

4. **Jeg tror spillet passer for elever som tar valgfaget programmering**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

5. **Jeg synes spillet var tungvint å spille**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

6. **Jeg måtte få hjelp for å komme i gang med spillet**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

Motivasjon og læring

7. **Jeg synes det var gøy å løse oppgavene**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

8. **Jeg synes det var gøy å spille spillet**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

9. **Jeg ble motivert av å være en del av et lag**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

10. **Jeg ble motivert til å gjøre oppgaver**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

11. **Jeg ble motivert fordi jeg ville vinne**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

12. **Spillet hjalp meg med å feilsøke koden min**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

13. **Jeg klarte å fokusere på oppgavene selv om vi konkurrerte**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	◯	◯	◯	◯	◯	Veldig enig

14. **Jeg har fått bedre forståelse for programmering**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	⬭	⬭	⬭	⬭	⬭	Veldig enig

15. **Jeg synes spillet skal brukes i programmeringsfaget**
Markér bare én oval.

	1	2	3	4	5	
Veldig uenig	⬭	⬭	⬭	⬭	⬭	Veldig enig

Appendix G

Answers of the Student Questionnaire for the Main Evaluation of Code Flip

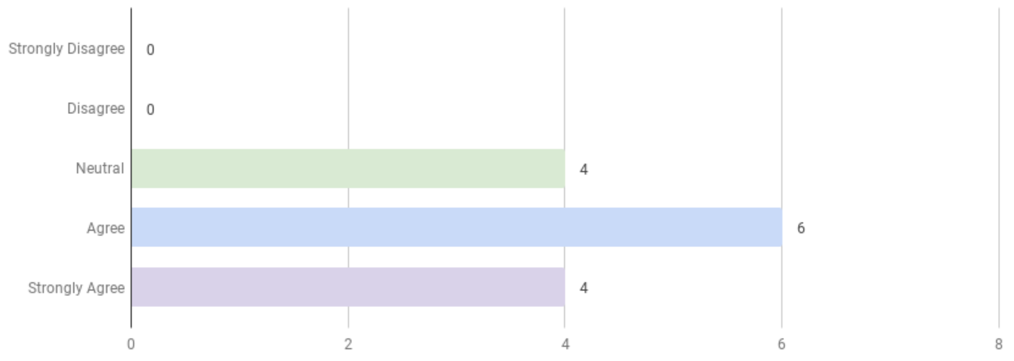


Figure G.1: Answers to statement 1: *I would like to play the game again.*

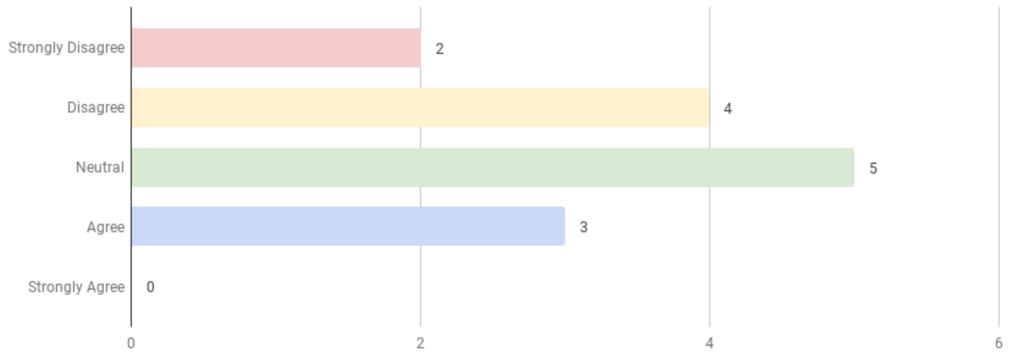


Figure G.2: Answers to statement 2: *I found it difficult to pay attention to everything that was going on in the game.*

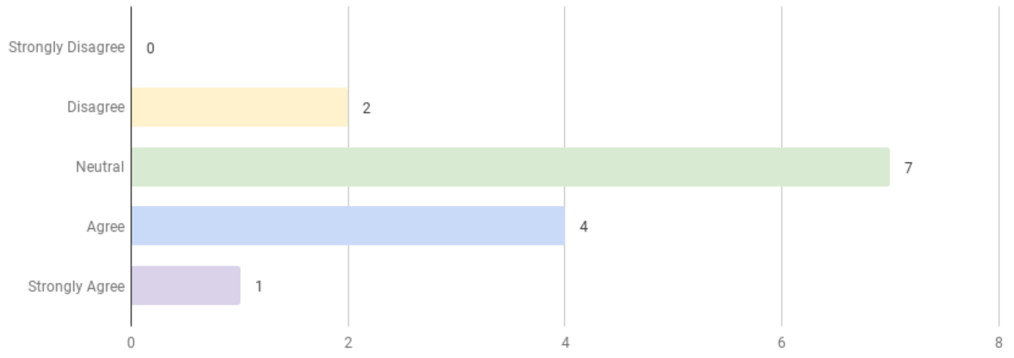


Figure G.3: Answers to statement 3: *I found it hard to understand what I was supposed to do.*

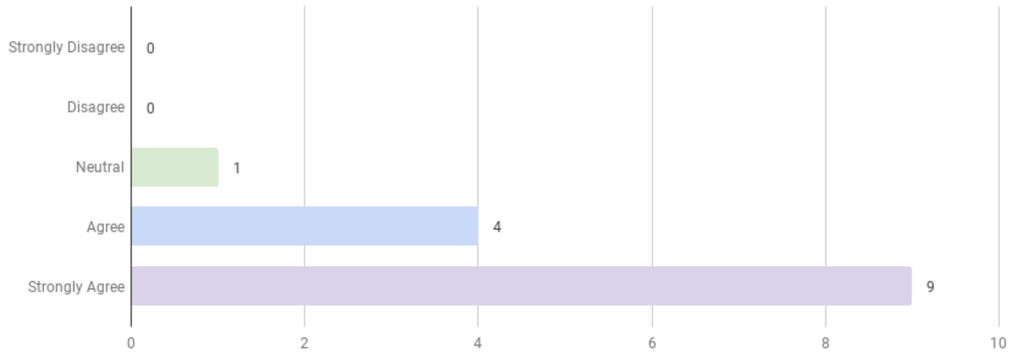


Figure G.4: Answers to statement 4: *I think the game is suitable for the Elective Course of Programming.*

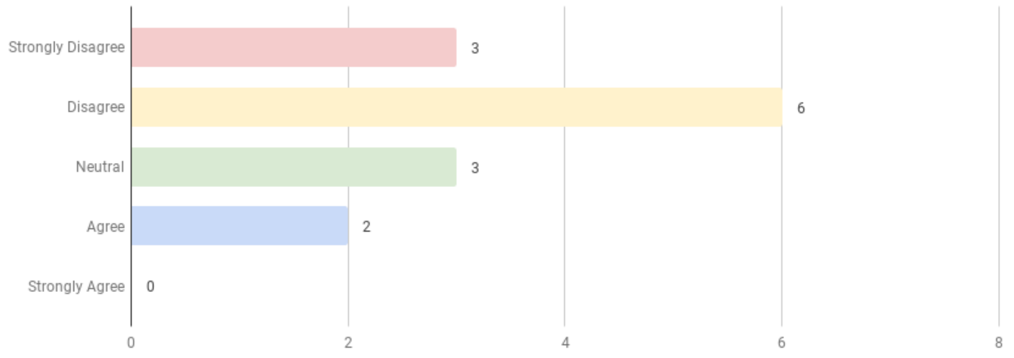


Figure G.5: Answers to statement 5: *I thought the game was cumbersome to play.*

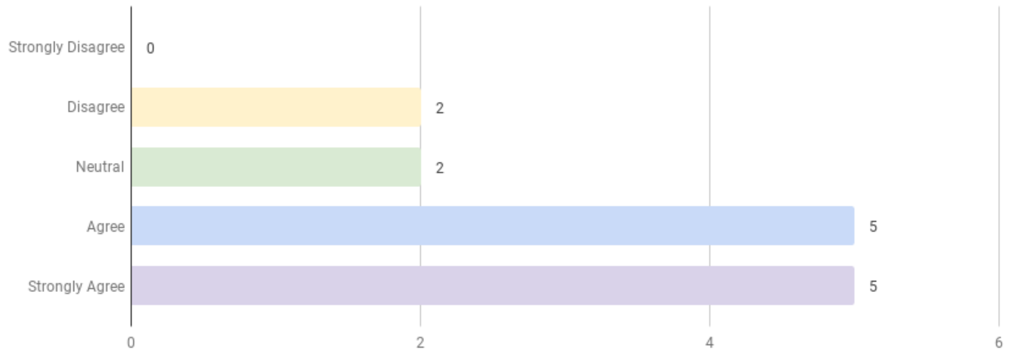


Figure G.6: Answers to statement 6: *I needed help before I could start playing the game.*

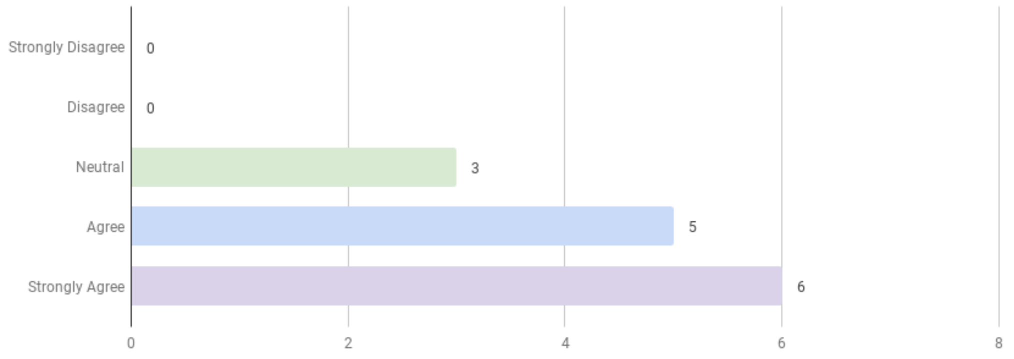


Figure G.7: Answers to statement 7: *I thought it was fun to solve the tasks.*

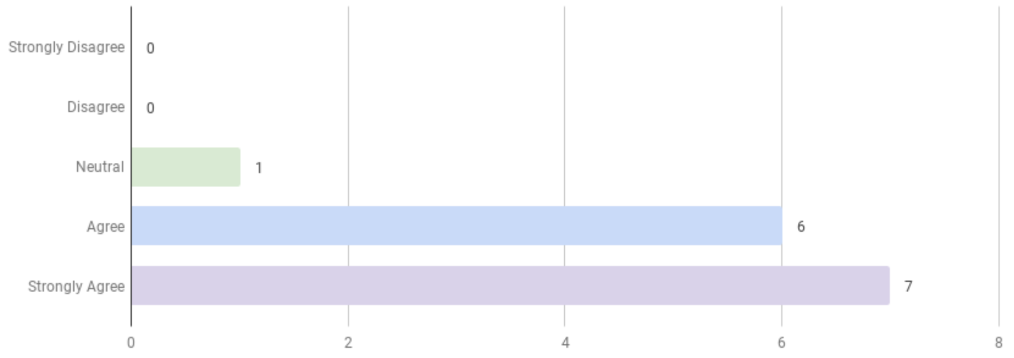


Figure G.8: Answers to statement 8: *I thought it was fun to play the game.*

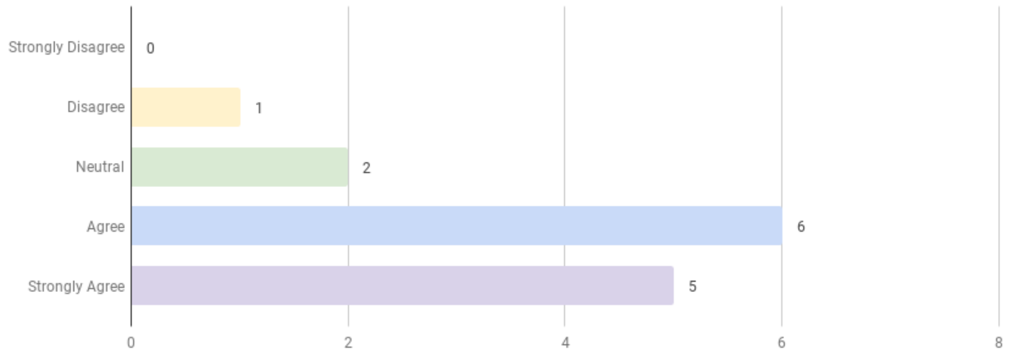


Figure G.9: Answers to statement 9: *I was motivated by being part of a team.*

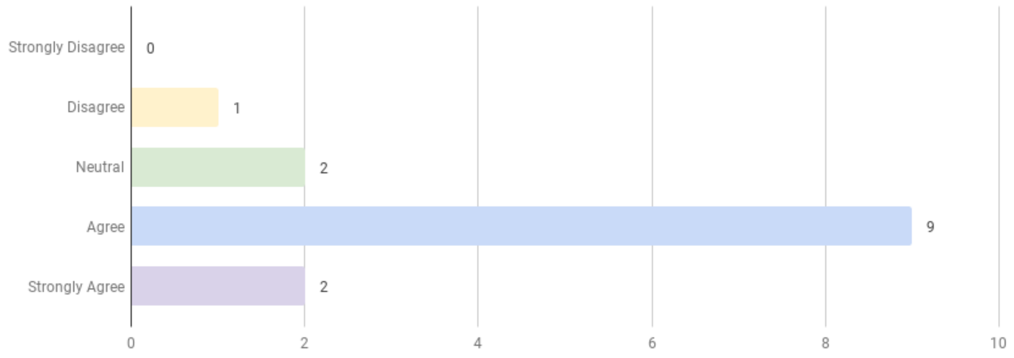


Figure G.10: Answers to statement 10: *I was motivated to solve tasks.*

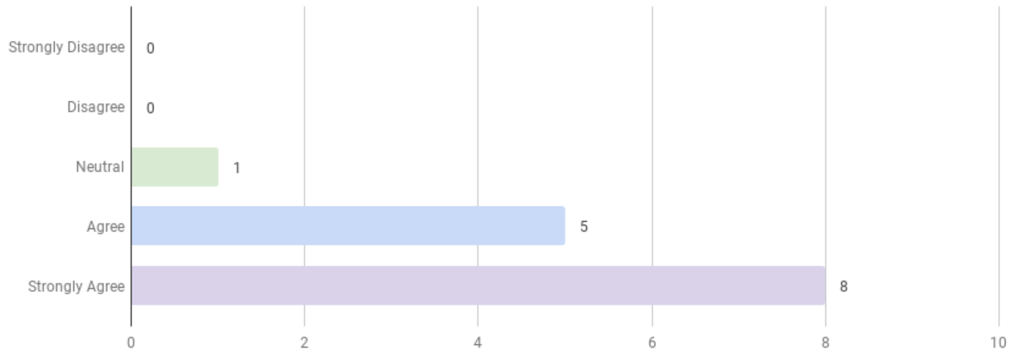


Figure G.11: Answers to statement 11: *I was motivated because I wanted to win.*

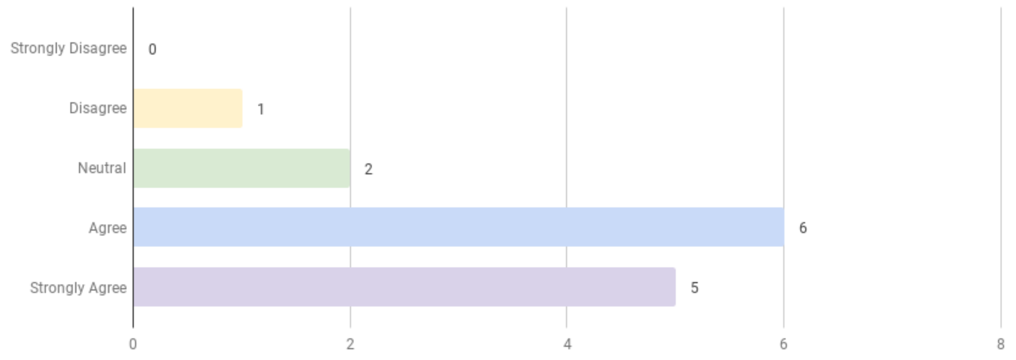


Figure G.12: Answers to statement 12: *The game helped me troubleshoot my code.*

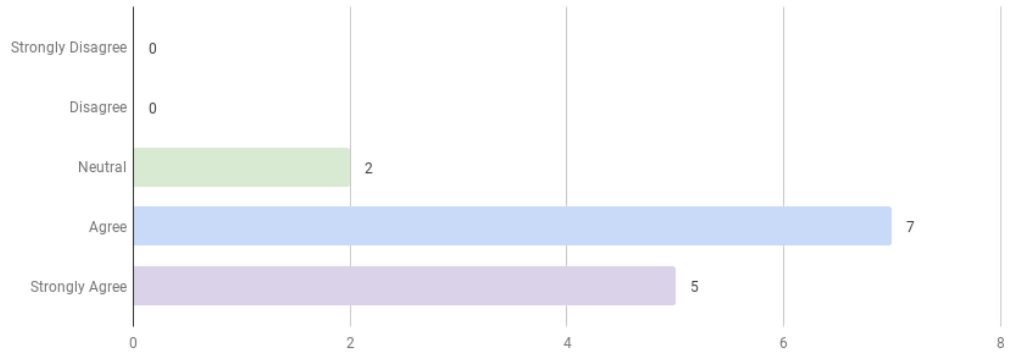


Figure G.13: Answers to statement 13: *I managed to focus on solving the tasks even though I wanted to win.*

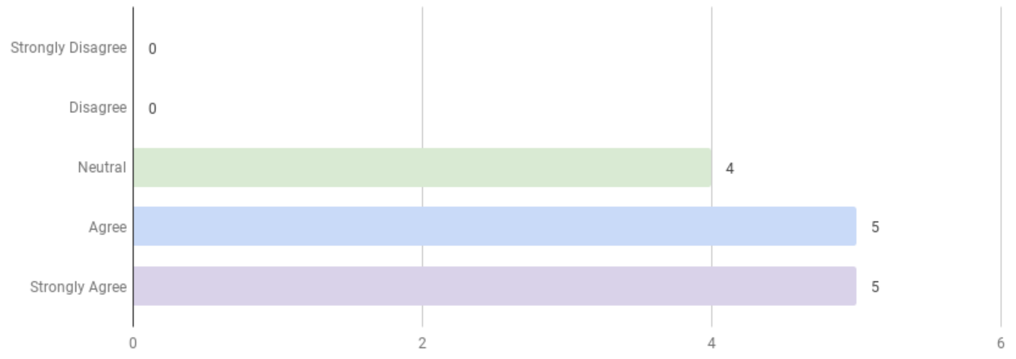


Figure G.14: Answers to statement 14: *I have a better understanding of programming after playing the game.*

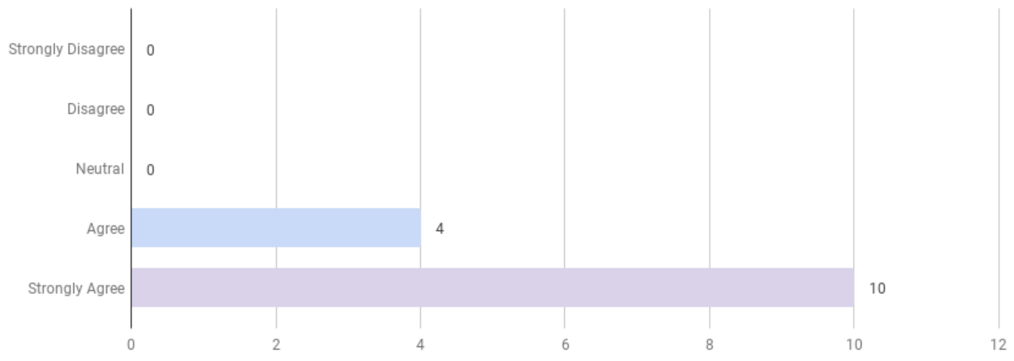


Figure G.15: Answers to statement 15: *I think the game should be used in the Elective Course of Programming.*

Anniken Holst and Marianne Magnussen

Code Flip: A Game for the Norwegian Elective Course of Programming

NTNU
Norwegian University of
Science and Technology