

Niklas Molnes Hole

Programming in Introductory Physics: an Online Learning Platform to Support Teachers

Master's thesis in Informatics (MIT)

Supervisor: Monica Divitini

August 2020

Niklas Molnes Hole

Programming in Introductory Physics: an Online Learning Platform to Support Teachers

Master's thesis in Informatics (MIT)

Supervisor: Monica Divitini

August 2020

Norwegian University of Science and Technology

Faculty of Information Technology and Electrical Engineering

Department of Computer Science



Norwegian University of
Science and Technology

Abstract

The Norwegian government has decided that as of 2021, programming will be added to the physics curriculum at Norwegian upper secondary schools (USS). Teachers have been able to attend ProFag¹ and other initiatives to learn how to implement programming in their respective courses. Thus, developing tools to aid in this transition could be essential for success in advanced courses like physics.

However, there is a lack of tools specifically made for introducing programming in introductory physics courses. Also, the available tools for introducing programming generally lack the functionality of creating custom tasks.

To attempt to help solve these problems, a design science research (DSR) process was conducted. By designing a tool, an online learning platform (OLP), that could introduce programming in an introductory physics course, it was possible to identify which elements that were important in such a tool. The OLP was also designed to allow the user to create programming tasks. This made it possible to find out how a user interface (UI) could be designed to benefit the physics teachers. In order to get a justified answer on both of these problems, the OLP was further designed and evaluated on three different audiences: university students (pilot), experts working with this field (expert), and physics teachers with no prior expertise in this field (main).

A total of 18 elements were found to be crucial in the design of a tool that attempts to introduce programming in an introductory physics course. It was also found that when creating a UI for creating programming tasks suited for introductory physics, it was important to have the option to hide any distracting code and to be able to test the task in a realistic environment while creating. Additionally, making the UI support the way task creators usually create tasks was also important. It was also found that physics teachers are more interested in using premade tasks than creating them themselves. However, they were interested in modifying existing tasks.

Besides what was found during the evaluation, the OLP that was made also aims to inspire work on research and development of tools explicitly made for introducing programming in specific topics. The OLP artifact designed and developed during the work with this thesis is available as an online demo² and source code³.

¹Programmering for Fagenes skyld: <https://www.mn.uio.no/kurt/livslang-lering/profag>

²<https://master-thesis-artifact.now.sh>

³<https://github.com/niklasmh/master-thesis-artifact>

Preface

This master thesis marks the end of a five-year-long study of informatics at NTNU. I have always been interested in programming, and I have been lucky to find a major that allowed me to make a career out of it.

My interest in educational programming, mostly in physics, sparked when I went to upper secondary school. I was lucky enough to have a teacher, Inga Hanne Dokka, that encouraged me to experiment with it during my time in school. In my senior year of upper secondary school, she was selected to receive a teaching award at NTNU. Before she traveled to Trondheim to receive her award, she invited me to join her to speak about the digitalization of the physics course. I was really thrilled to get the chance and accepted the offer. The speech went great, and I found the topic to be interesting and important. However, once I started attending university, I no longer found the time to work on it. Because of this, the spark was lost over time. But five years later, while brainstorming ideas for my thesis, I found back to my old roots again. I have now written a thesis about the same topic at the same university that the speech was held.

I was lucky enough to find a supervisor with similar interests. Monica Divitini has been of great help and has given me guidance, motivation, and thorough feedback when I have needed it.

I would also like to thank all the teachers, researchers, and fellow NTNU students that participated in the interviews conducted in this study. Without them, this master thesis would not have been possible.

Niklas Molnes Hole
Trondheim, August 3, 2020

*“You think you know when you can learn,
are more sure when you can write,
even more when you can teach,
but certain when you can program”*

— Alan Perlis

Table of Contents

Abstract	i
Preface	ii
Table of Contents	viii
List of Tables	x
List of Figures	xi
Abbreviations	xii
1 Introduction	1
1.1 Motivation	2
1.1.1 Personal Motivation	2
1.1.2 General Motivation	2
1.2 Research Questions	2
1.2.1 Sub Research Questions	3
1.3 Research Method	4
1.4 Results	5
1.5 Report Outline	5
2 Problem Elaboration	7
2.1 The Environment	7
2.1.1 Upper Secondary School	7
2.1.2 The Physic Course in Norwegian USS	7
2.2 The New Physics Curriculum	8
2.2.1 Development of The New Curriculum	8
2.2.2 First Hearing of The New Physics Curriculum	8
2.2.3 Second Hearing of The New Physics Curriculum	8
3 Expert Interviews	11
3.1 Participants	11
3.2 Process	11
3.3 Interview Guidelines	12
3.3.1 Part (a):	12
3.3.2 Part (b):	12
3.3.3 Part (c):	12

3.4	Results	13
3.4.1	Backgrounds	13
3.4.2	Use of Computers in The Physics Course	13
3.4.3	Use of programming in The Physics Course	14
3.4.4	Use of CT in The Physics Course	14
3.4.5	The Norwegian CT is Not The Same as The Original CT	14
3.4.6	Problems with Defining CT	15
3.4.7	Focus on Modeling in Physics may be a Good Idea	15
3.4.8	Loops are The Most Difficult in Programming	15
3.4.9	Some Pupils Have Used Programming in The Physics Course	16
3.4.10	The Future of The Physics Course	16
4	Literature Review	19
4.1	Theory	19
4.1.1	Learning	19
4.1.2	Learning Environments	20
4.1.3	Programming in Physics Education	22
4.2	Background	23
4.2.1	Using Computers to Learn	23
4.2.2	Using Programming in Physics Education	25
4.3	Related Work	25
4.3.1	Programming Environments	25
4.3.2	Teaching Programming	30
4.3.3	Creation of Programming Tasks	31
4.4	Method	32
4.4.1	Structured Literature Review: How It Was Done	33
4.4.2	Alternative Search Strategies	36
5	Answering Research Question 1.1	39
5.1	Falling Object	39
5.2	Pendulum	41
5.3	Block Down a Slope	41
5.4	Bead on a Wire	42
5.5	Ball Collisions	42
5.6	Electron on Uniform Magnetic Field	43
5.7	Planet Orbits	43
5.8	Asteroids Game	43
5.9	Conclusion	44
6	Requirements	47
6.1	Discussion of Requirements	49
6.1.1	Research Question 1.1	49
6.1.2	Research Question 1.2	49
6.1.3	Research Question 1.3	51

7	Design of The Initial Artifact	55
7.1	RQ1.2: Designing The Task UI	55
7.1.1	Elements Included from Requirements	55
7.1.2	Elements Included Due to Design Choices	57
7.1.3	Summary	59
7.2	RQ1.3: Designing The Task Creation UI	59
7.3	General Design Choices	61
8	Implementation of The Initial Artifact	63
8.1	Technical Choices	63
8.1.1	Web Technology	63
8.1.2	User Interface	63
8.1.3	Storing Data	64
8.1.4	Running Python 3	65
8.1.5	Writing Python 3 Code	65
8.2	Task UI	65
8.2.1	List of All Elements in Figure 8.1	66
8.3	Task Creation UI	67
8.3.1	Describing Figure 8.2	68
8.3.2	Describing Figure 8.5	68
8.3.3	Describing Figure 8.6	68
8.4	Alternative Task Creation UI	73
8.5	Limitations	74
8.5.1	Missing Graphs	74
8.5.2	Slow Loading Time	74
8.5.3	Incorrect Line Numbers on System Errors	74
9	First Evaluation: Pilot	75
9.1	Participants	75
9.2	Method	75
9.2.1	Preparation Routine	75
9.2.2	Process Routine	76
9.3	First Part: Task UI	76
9.3.1	Results	76
9.3.2	Discussion	76
9.4	Second Part: Task Creation UI	83
9.4.1	Results	83
9.4.2	Discussion	84
10	Second Evaluation: Expert	85
10.1	Participants	85
10.2	Method	86
10.2.1	Preparation Routine	86
10.2.2	Process Routine	86
10.3	First Part: Task UI	86
10.3.1	Results	86
10.3.2	Discussion	87

10.4	Second Part: Task Creation UI	92
10.4.1	Results	92
10.4.2	Discussion	94
11	Third Evaluation: Main	95
11.1	Participants	95
11.2	Methods	95
11.2.1	Preparation Routine	95
11.2.2	Process Routine	96
11.3	First Part: Task UI	96
11.3.1	Results	96
11.3.2	Discussion	96
11.3.3	Implications For The Design	98
11.4	Second Part: Task Creation UI	100
11.4.1	Results	100
11.4.2	Discussion	100
12	Conclusion	101
12.1	Answering The Research Questions	101
12.2	Future Work	103
	Bibliography	105
	Appendix	111
A	NDA for Expert Interviews	113
B	Original Task UI Screenshot	117
C	Original Task Creation UI Screenshots	119
D	NDA for Evaluation Interviews	123

List of Tables

2.1	Summary of the first hearing (UDIR, 2019b) with focus on programming.	9
2.2	Summary of the second hearing (UDIR, 2020a) with focus on programming. Numbers in parentheses are duplicate answers. . .	9
3.1	The interviewees.	13
4.1	Some advantages and disadvantages of using OLPs (Albashaireh and Ming, 2018, p. 632) combined with our comments on them. .	27
4.2	Terms and synonyms (groups) used in query. Table structure inspired by Kofod-Petersen (2015).	34
4.3	Inclusion and quality criteria. Inspired by Kofod-Petersen (2015).	35
4.4	The studies that were included after the final selection phase in the SLR.	36
4.5	Studies found using references from SLR studies.	37
4.6	Studies found using the google strategy.	38
5.1	Evolution of programmable physics phenomena. There are three levels of difficulty: Easy (E), Medium (M), and Hard (H). The ones marked with green are the ones that were considered as simple enough to program in the introductory physics course, having at least a few additional elements to build upon the phenomena. . .	45
6.1	List of requirements. The adapted requirements are marked with *.	48
7.1	Initial elements placed in the design of the task UI in the initial artifact. RID is the ID of the requirement.	59
9.1	Feedback from the first part of the pilot evaluation.	77
9.2	Feedback on specific elements from the pilot evaluation. The yellow ones were added from suggestion.	79
9.3	Changes from feedback in Table 9.1.	80
10.1	Feedback from the first part of the expert evaluation.	88
10.2	Feedback on specific elements from the expert evaluation. The green rows contains elements that was moved into the inclusion area. The yellow are new ones added by suggestions	90
10.3	Changes from feedback in Table 10.1.	91

11.1 Feedback from the first part of the main evaluation.	97
11.2 Feedback on specific elements from the main evaluation.	99

List of Figures

1.1	Design Science Research Cycles (Hevner, 2007, p. 88)	4
4.1	LOGO (left) VS. Boxer (right). ⁷	24
4.2	Our keywords and their relation in the search.	33
5.1	Example evolutions of the falling object phenomenon: (a) falling ball in 1D, (b) thrown ball in 2D, (c) thrown ball with air resistance, and (d) thrown ball with air resistance and bounce.	40
5.2	Example evolutions of the block down a slope: (a) simplest form, and (b) with friction.	42
8.1	The initial task UI with highlighted elements.	67
8.2	The top of the initial task creation UI.	69
8.3	An open help bubble in the initial task creation UI.	69
8.4	The descriptions of the initial task creation UI.	70
8.5	The middle of the initial task creation UI.	71
8.6	The bottom of the initial task creation UI.	72
8.7	Alternative task creation UI based on Markdown syntax.	73
8.8	Conceptual design of graphs.	74
9.1	Fixes for code editor.	81
9.2	Fixes for FPF08 and FPF18.	81
9.3	Fix for FPF09.	81
9.4	Fix for FPF11.	82
9.5	Automatically populate section button.	84
10.1	Fix for FEF03.	89
10.2	Third task creation UI.	93
B.1	The task UI with no edits.	118
C.1	The task creation UI with no edits. (Top)	119
C.2	The task creation UI with no edits. (Middle)	120
C.3	The task creation UI with no edits. (Bottom)	121

Abbreviations

- CL** = **Computer Literacy.**
Understanding how to use a computer. E.g. open a browser.
- CP** = **Computer Programming.**
Order the computer to do tasks, using instructions.
- CS** = **Computer Science.**
- CT** = **Computational Thinking. “Algoritmisk tenking”.**
Definitions: UDIR (2019a); Wing (2012)
- DSR** = **Design Science Research.**
Definitions: Hevner (2007).
- ODE** = **Ordinary Differential Equation.**
- OLP** = **Online Learning Platform.**
- PE** = **Programming Environment.**
The same as a “microworld”, as described in Papert (1980), p. 120.
- STEM** = **Science, Technology, Engineering and Mathematics.**
- TSPE** = **Task-Specific Programming Environment.**
See [section 4.1.2](#).
- RQ** = **Research Question.**
A question used to define a problem.
- SLR** = **Structured Literature Review.**
Definitions: Kofod-Petersen (2015)
- UI** = **User Interface.**
- USS** = **Upper Secondary School. “Videregående skole”.**

Chapter 1

Introduction

It has been decided that programming will be included in introductory physics courses at Norwegian upper secondary schools (USS) in 2021 (UDIR, 2020d). This means that pupils are not that far away from being enabling to explore the world of physics through a new way of thinking. This is a really good thing as it also opens up for new and exciting material, but it is important to be careful. Just adding programming to a course that is not about computer science, in general, could remove attention from the actual material itself.

ProFag¹ and other initiatives have already been preparing teachers for this transition since 2016. They have focused on how the teachers can implement programming in their own courses, including how to teach programming in a classroom setting. But it is important to note that this solution was the fastest solution to get all the teachers on board the programming train on short notice, meaning it may not be enough to prepare all the teachers with the set of skills that they need to teach their classes. They also lack scientific research support in teaching programming for the sake of the courses, simply because this has not happened before in Norwegian schools. In turn, this leads to a proposal for a tool that can help ease some of this work that is currently ongoing as well as increasing the chance for making the transition a success.

Today there exist several tools that can support teachers in introducing programming. However, only a few of them are made for introducing programming in introductory physics, specifically. After interviewing three experts working with programming and physics teaching, four tools² were found to be good alternatives when introducing programming in an introductory physics course. However, only one of them, Tychos, was actually made for introducing programming in introductory physics — which proves that this topic is relevant, but it also proves that there have been too few attempts to make this work. The other alternatives were general-purpose environments that could teach multiple fields of sciences as well as just programming. They were also designed for effective use by professional scientists, meaning they lack the ability to introduce programming in a good way — which other tools can³. Brown and Wilson (2018) also adds that

¹Programmering for Fagenes skyld: <https://www.mn.uio.no/kurt/livslang-lering/profag/>

²Jupyter, Spyder, Tychos, and Trinket.

³Codecademy and Khan Academy

it is important to remember that novices are not experts and, therefore, need different tools as they programs differently.

There is also another problem with the available tools today. Most tools created for specific tasks do not make it possible to create tasks. This forces the teachers to choose tasks that may not fit with their teaching. This can be an issue as the learners then may need to learn a part of programming that is not necessarily needed for introductory physics, e.g., arrays and loops. Having an option to create or modify programming tasks such that they can skip right to the programming material that is important may be beneficial for both the teachers and the learners. Research also suggests that teaching programming material that is relevant for the learner makes programming easier to learn (Claypool et al., 2004; Guzdial and Naimipour, 2019). This means that customizing the programming material could play an essential role in making programming success in an introductory physics course.

1.1 Motivation

1.1.1 Personal Motivation

I chose this topic as I once was fiddling around with programming in the physics course at USS. I definitely got more into physics for that reason and understood some of the concepts more thoroughly. While I was programming in the physics course, I also got more interested in creating games and simulations using what I had learned, which in turn made me more interested in physics. I even ended up creating a dummy physics engine⁴.

I also created a few more advanced creations in GeoGebra⁵, but GeoGebra was primarily made for doing math with equations and could not handle interaction and advanced behavior (if and else statements) in an easy way. In this thesis, I have decided to go more in-depth on how we can make programming a part of a USS physics course.

1.1.2 General Motivation

There is also another motivation that needs to be mentioned, which also plays an important role in this thesis: the curriculum in Norwegian schools is in the middle of a transition (UDIR, 2020c). The reason why this is a motivation factor is that it could not have happened at a better time. We are just in time for thinking new about the whole introductory physics course. This research could even have a positive impact on how physics is taught in Norwegian schools.

1.2 Research Questions

To help find answers to the problems identified in the beginning, this research question is posed:

⁴<http://nikkapp.com/physics4.html>

⁵GeoGebra: <https://www.matematikkenteret.no/nyheter/geogebra>

Research Question (RQ1) *How can we design an OLP that supports teachers with teaching programming to pupils in the introductory physics course at Norwegian USS?*

An OLP, or Online Learning Platform, is a tool where teachers can create and share material with pupils in one single place. A more thorough description of OLP is given in [subsection 4.3.1](#). Anyway, by using an OLP, the teachers automatically get advantages like automatic grading, control over pupil's and content, and tracking of pupils progress. More advantages are described in [Table 4.1](#).

1.2.1 Sub Research Questions

Before answering the research question (RQ1), more information about the topic will be needed. A few sub research questions that can help us find this information have therefore been created.

Research Question 1.1 (RQ1.1) *What phenomena in physics can be programmed in the introductory physics course?*

To design an OLP that supports teachers with teaching programming in physics, it is essential to know what will be taught. In [RQ1.1](#), the goal was to find the phenomena in the current introductory physics course that is well suited for being programmed with no prior programming experience.

Historically, programming has not been a part of the physics curriculum. However, that may change in 2021 as the latest draft of the new physics curriculum (see [section 2.2](#)) has added it as one of the learning goals. More specifically, programming has now been suggested to explore problems with non-constant acceleration (UDIR, 2020b). Anyway, it was still a draft, as of the time writing, meaning [RQ1.1](#) still focused on all areas of physics in the introductory physics course, not dependent on what was sketched in the drafts, even though it was a good indicator.

Research Question 1.2 (RQ1.2) *What elements does an OLP need to teach pupils to use programming in the introductory physics course?*

In [RQ1.2](#), the focus is on how the pupils perceive the OLP used by the teachers to teach them. The teachers today are assumed to use Spyder or Jupyter (see [section 4.1.2](#)) to teach programming, according to ProFag⁶. Nevertheless, these tools may not contain what is needed for a good learning experience for a first-time programmer and a first-time physicist — and that is what this question is asking to find out, specifically. E.g., does loops need to be involved in the tool, or should they be abstracted? Is graphics beneficial, or does it make the tasks more complex?

Research Question 1.3 (RQ1.3) *How can we design a user interface for creating programming tasks in the introductory physics course that supports the teachers?*

⁶First presentation: <https://www.mn.uio.no/kurt/livslang-lering/profag/presentasjon/>

At last, [RQ1.3](#) is focusing on how teachers should create programming tasks for the pupils. More specifically, how should a user interface (UI) for creating programming tasks be designed. Designing a UI that makes it possible to create programming tasks may enable teachers to adapt the programming part to their teaching. This may also engage teachers in learning programming, which in turn makes them able to help their pupils and make programming more interesting. Teachers are already a great resource, and involving them more could lead to better learning outcomes for both them and their pupils. However, this may also be too much work, even if the design for creating programming tasks is perfect, and it could, therefore, be that teachers would rather choose from a collection of premade tasks instead. It may also be a mix where teachers want to choose from a collection, then modify the task for their purpose.

1.3 Research Method

To find the answers to the research questions, a design science research (DSR) process was used. DSR is a methodology that makes it possible to develop an artifact, or an OLP in this case, in a new and innovative way (Hevner et al., 2004, p. 75). After the artifact is created, it is then possible to test it on the real users, checking if the artifact meets its requirements.

It is important to state the fact that DSR is “the science of the artificial”, or *design science*, and not *natural science*. That is, the *design science* focuses on understanding elements or phenomena that are man-made, while *natural science* focuses on understanding elements or phenomena from nature. The difference is described more in-depth in Simon (1996). It is also worth noting that physics is a *natural science* in itself. It is, therefore, important to differentiate between the *process* of learning physics and learning physics. This thesis will focus on the *process* of learning physics, thus the *goal* is to learn physics.

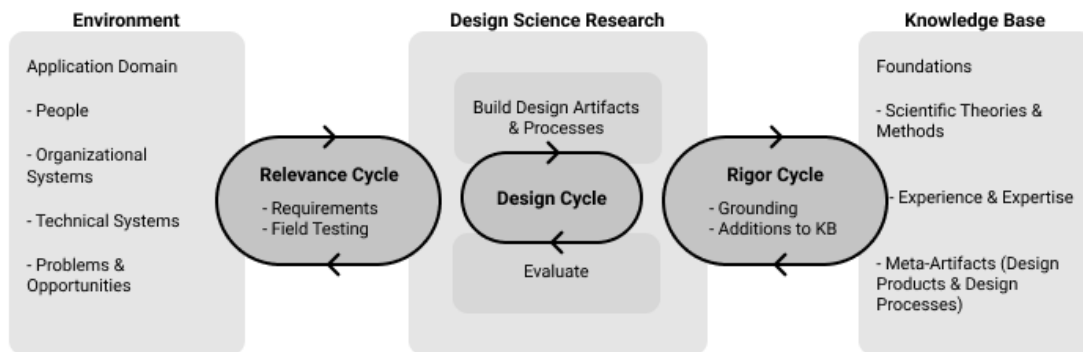


Figure 1.1: Design Science Research Cycles (Hevner, 2007, p. 88)

The methodology is usually described using a three cycle view: *relevance cycle*, *design cycle* and *rigor cycle*. See figure 1.1. It is said “that these three cycles must be present and clearly identifiable in a DSR project,” (Hevner, 2007, p. 88). This research first started with the *rigor cycle*, by finding literature, related work, and previous attempts. To make sure that what was found was useful to the artifact, four expert interviews were conducted. This belongs to the *relevance cycle*. Next,

the artifact was initially designed using requirements that were created based on the two earlier cycles. This was done in the *design cycle*. When the artifact was ready to be tested, it was first tested by a pilot group. This goes back to the *relevance cycle*. The artifact was then further developed based on feedback from the pilot group, which is done in the *design cycle*. The last two cycles were repeated two more times until the requirements were met, and the project was able to conclude.

1.4 Results

This master thesis contributes with:

- Interviews with experts in this field, giving an idea of what the situation was when writing this thesis. See [chapter 3](#).
- List of tasks that are suited for being taught in introductory physics using programming. See [chapter 5](#).
- List of elements that should exist in a tool that attempts to introduce programming in an introductory physics course. See [Table 11.2](#).
- Description of how an UI for creating tasks should be designed. See [section 9.4](#), [section 10.4](#), and [section 11.4](#).

1.5 Report Outline

The thesis starts with problem elaboration in [chapter 2](#) and expert interviews in [chapter 3](#), noting the relevance of the topic. It then moves onto a literature review in [chapter 4](#) to find grounding. Based on the previous chapters, it was then possible to find the answer to [RQ1.1](#), which got its chapter: [chapter 5](#). Next, a set of requirements needed to be made; this was done in [chapter 6](#). When the requirements were done, it was possible to design the initial OLP in [chapter 7](#). This design was further implemented in [chapter 8](#). The next three chapters, [chapter 9](#), [chapter 10](#), and [chapter 11](#), consists of testing, discussion and further changes to the OLP. The thesis ends with the conclusion in [chapter 12](#).

Chapter 2

Problem Elaboration

This chapter further elaborates on the environment that this thesis works with. The first part of this chapter described how the Norwegian upper secondary school (USS) works, including the physics course. The second part describes the changes that are about to happen to their curriculum.

2.1 The Environment

It is important to know the school system in Norwegian USS that surrounds the physics course in further reading. It is also essential for understanding the environment that the pupils learn in.

2.1.1 Upper Secondary School

In Norway, a pupil usually starts at USS right after ten years of ground education. Pupils are generally between 15 and 16 years old when beginning such an education.

2.1.2 The Physic Course in Norwegian USS

In the Norwegian USS, the introductory physics course is an elective course for the pupils to enroll in the second year. There also exists an intermediate course that comes after the introductory course, but that course is not the main focus of this thesis. However, since the physics course at the second year, the pupils are expected to know some mathematics from the first year.

In the future (from 2021), the pupils taking this course will probably already have an introduction to programming from other STEM courses in the first year. This means that the level of programming expected in physics does not need to be necessarily introductory, but it is not safe to aim for that yet as it could be optional, and some pupils may need some introduction anyway. The change in the curriculum is more thoroughly described in the next section.

2.2 The New Physics Curriculum

As mentioned in the introduction of the report, a new curriculum for the physics course is in development and will be ready in 2021 (UDIR, 2020d). This new curriculum will be crucial for introducing programming into the physics course as most teachers are dependent on the curriculum when they are prioritizing activities in the classroom (interviewee D). This means that if the physics curriculum does not contain any programming, most of the interest in teaching it will be gone. However, the current results from hearings of the drafts of the new curriculum show that there is a great interest in teaching programming already. This further may imply that it will stay in the final version as well.

2.2.1 Development of The New Curriculum

The new physics curriculum has been developed in parallel with other curriculums, as all Norwegian curriculums are going through a comprehensive change (UDIR, 2020c). This new change aims to make the school system more in line with the future workforce. To develop these new curriculums, UDIR — which has been given the main responsibility for the development, has held a few hearings on drafts of each of the new curriculums. These hearings make it possible for teachers, universities, and others interested in telling UDIR what they think about them and giving suggestions. Then, after each hearing, UDIR goes through the feedback and iterate further on the drafts. In the end, a new curriculum should have been developed.

2.2.2 First Hearing of The New Physics Curriculum

So far, as of before August 2020, there have been two hearings of the new physics curriculum. The first hearing was about the first draft of the physics curriculum that was released 30th of October 2019 (UDIR, 2019c). In this hearing, there was a lot of positivity around programming. Of 53 answers, 11 of them commented that they were positive to programming, and no one was against it, at least not in the comments. [Table 2.1](#) gives a short summary of how the answers were distributed across the stakeholders. The 11 answers that were positive to programming either wanted to know more specifics or came with suggestions of what the programming part should contain, like air resistance, simulations, and micro-controllers. The rest who did not comment on the programming part may not have had any strong opinions about the topic. Alternatively, as the first draft was not that specific in general, they could also be waiting for more specifics as there was not much to comment on.

2.2.3 Second Hearing of The New Physics Curriculum

The second hearing was about the second draft of the new physics curriculum that was released on the 27th of February 2020 (UDIR, 2020b). In the new version, UDIR was more specific on how programming was going to be used in the physics course. For the first physics course, the students should use programming to

Stakeholder	Answers	For	Against	No comment
Universities	5	1	-	4
Schools	35	8	-	27
Researchers	1	-	-	1
Teachers	12	2	-	10
Sum	53	11	-	42

Table 2.1: Summary of the first hearing (UDIR, 2019b) with focus on programming.

model and explore movement that had non-constant acceleration. For the second physics course, which is not the target of this project, they included programming to explore and model movements in two dimensions.

Interestingly, in this hearing, there was almost a doubling in the discussion about the programming part. Of 83 answers, 23 of them had something to say about it, which is a 34% increase in interest. However, this time, there were three against having programming in the course. How the answers were distributed among the stakeholders can be found in Table 2.2. The ones that were positive to having programming in the physics course had a wide range of answers. This time, some argued that the curriculum was too specific, as opposed to last time. Others argued that programming should have taken more space in the course and being included other topics.

On the other hand, the ones against including programming argued that it would take too much time or that non-constant acceleration would be too difficult to understand for the pupils. Overall, it was a great interest in including programming in the physics course. This may, in turn, result in actually including programming in the physics course in the final version in 2021.

Stakeholder	Answers	For	Against	No comment
Organizations	3	-	1 (+2)	-
Public sector	1	-	-	1
Universities	3	1 (+1)	-	1
Schools	46	13	2 (+1)	31
Companies	1	-	-	1
Other	1	1	-	-
Private persons	2 (+1)	-	-	3
Teachers	26	5	-	21
Sum	83 (+1)	20 (+1)	3 (+3)	58

Table 2.2: Summary of the second hearing (UDIR, 2020a) with focus on programming. Numbers in parentheses are duplicate answers.

Chapter 3

Expert Interviews

Since before the delivery of this project, there has been limited information on how the physics teachers should include programming in their courses (see [section 2.2](#)). To get more insight, it was decided to conduct interviews with some of the teachers and researchers involved in this process.

3.1 Participants

The aim was to interview teachers and researchers that had experience with the physics course in general. After a few mail exchanges, seven qualified people were contacted, and four agreed to have an interview. The participants ranged from fulltime physics teachers to full-time researchers, including participants doing both.

3.2 Process

There was used a semi-structured format on the interviews, doing the interview more like a conversation. This was done to gather as much information from the respondents as possible while having a few questions to guide the conversation.

Two interviews were done in person, one was done over a video, and one was done using text-exchanges. The two in person and the one over video were recorded using a sound recorder provided by NTNU. The recordings were later transcribed and anonymized, then deleted.

All of the respondents that were interviewed had also consented to a non-disclosure agreement from NSD¹ telling that everything that they say in the interview may be used this master thesis, but also that they are free to remove their consent or some of the information later. The consent is added to [Appendix A](#).

¹<https://nsd.no>

3.3 Interview Guidelines

The interview had three parts; (a) background, (b) their experience with computers and programming in the physics course, and (c) what ideas they might have about the future of the physics course. The last part was added to get more of their creative mind to imagine how programming could work in the future classrooms, as it did not yet do.

The questions are listed below:

3.3.1 Part (a):

- Do you have any experience with programming?
- What is your experience with physics?
- Have you worked with anything related to education?

3.3.2 Part (b):

- How do your students use computers in the classroom? Or are they allowed (if not, why)?
- What is your experience with computers in the classroom?
- Do you know what “computational thinking” (algorithmic thinking) means?
- Have you ever used any form of programming/computational thinking in a physics course?
 - If yes, I would like to know your experience with that.
 - If no
 - * Why not?
 - * How would you think you would have used it?
 - * Do you know any students that have used programming in physics?

3.3.3 Part (c):

- In the future (10-20years), how do you think the physics course is going to be then?
- But first steps first, what do you think we can do today to reach that state?
- How would you create an ideal environment for learning physics, using programming/CT as a part of it?

The interviewer elaborated on all the answers with new questions. By using this method, it was possible to get a lot more information from the interviewees.

3.4 Results

3.4.1 Backgrounds

ID	Profession
A	Teacher and researcher
B	Researcher
C	Teacher and researcher
D	Teacher

Table 3.1: The interviewees.

All the interviewees had different backgrounds, which made it possible to gather a broad area of experience and knowledge. See [Table 3.1](#).

Interviewee [A](#) had experience with programming. The interviewee originally had a chemistry background, thus has since moved on to be a teacher in chemistry, physics, and natural sciences. The interviewee also made a course for programming and modeling in their school, meaning the interviewee already had some knowledge of implementing programming in schools from before.

Interviewee [B](#) had a background in physics and had done a lot of scientific programming in the past. Currently, the interviewee was doing research full-time, but also had field experience as a physics teacher, thus at a higher level than USS. The interviewee's research was in the direction of computational essays, a popular technology among physicists.

Interviewee [C](#) was originally a physics teacher for around 15 years, thus has since moved onto doing more research and talking publicly. This interviewee has been active in implementing programming into the physics curriculum and had a lot of insight into what was currently happening on this topic. This interviewee had previously learned to program in school but told that it was nearly forgotten since it was such a long time ago. Thus, the interviewee was newly introduced to programming for the second time and had from there refreshed the memory of programming.

The last interviewee, interviewee [D](#), had been a physics teacher for the last 16 years in a variety of schools, both USS and higher level. The interviewee also had nearly forgotten programming as of the time that had passed. Thus, the interviewee had also newly been introduced to programming for the second time and was, therefore, having programming fresh in memory too.

3.4.2 Use of Computers in The Physics Course

All of the interviewees had allowed the use of computers in their physics courses at some point. Thus they were cautious about using it unless there was a good reason. Mainly, the computers were used to write down results from experiments such that you could analyze the data, but apart from that, it did not bring much value to the course. Thus, none of them had any issues with using computers. Interviewee [B](#) also argued that even if the students were using computers and

other devices in the class, it was not a problem as it is “(...) just how students operate these days.”

3.4.3 Use of programming in The Physics Course

Neither of the interviewees had used programming in their physics course at Norwegian USS. Interviewee [A](#) said that there was a strict time constraint in the current course plan that did not make it feasible. However, that constraint would hopefully be removed in the new curriculum. Interviewee [D](#) pointed out that they always did an evaluation of activities that were relevant to include in the course, and that programming was never considered relevant enough as it was not included in the curriculum.

Interviewee [C](#) mentioned that there are two reasons that programming has not been used in the current physics course: First of all, many physics teachers do not know how to use it. Even though if they have previous experience with it, it is not the same when teaching a physics course. The second reason is that the current tradition is profoundly affected by analytical approaches — not numerical ones. This means that the teachers will find it hard to teach; they will also not find any material that covers these kinds of topics.

3.4.4 Use of CT in The Physics Course

Computational thinking (CT), or, more specifically, “algoritmsk tenking” (AT, see next section), was, according to interviewee [A](#), [C](#), and [D](#), a process that already was used in the physics course. The most typical problem described was how the pupils need to decompose problems in terms of motion. For example, separating the motion in the x and y-axis makes it possible to get two equations that are simpler to solve than a single equation for both. Interestingly, this is also something that the pupils see, meaning CT is not something that necessarily needs to be taught — it may come when seeing the relevance (see [subsection 3.4.10](#), interviewee [B](#)).

3.4.5 The Norwegian CT is Not The Same as The Original CT

Interviewee [C](#) strongly noted that the original definition of computational thinking (CT) is different from the Norwegian CT, “algoritmsk tenking”, which is closer to algorithmic thinking (AT). The reason is that it was hard to find a better Norwegian translation for CT. Interviewee [C](#) was actually involved in this process, together with UDIR — the main responsible for the new curriculum and all parts involved in this decision knew this. However, UDIR has made a definition on their website, UDIR (2019a), such that teachers searching for the term can see its definition and not mix it with the English algorithmic thinking.

The main problem with this “unfortunate” translation is that it is missing the “computational” part, meaning it does not suggest that it can be used on a computer, the interviewee said. Thus, the original CT and the Norwegian CT is not that different in terms of definition, making it maybe the best choice of words

yet. In the future, this may also change, meaning the current definition could change to, for example, “algorithmisk tenking 2020” instead, to mark that it is the old definition.

3.4.6 Problems with Defining CT

Interviewee B, on the other hand, was very into the definition of CT, and told that its definition is not final yet, and might never be. There is a lot of disagreement about what exactly the definition is. This may also be because people with different experiences focus on different aspects, making it a relative term based on the field it is used. This brings us to having to define CT in terms of physics, or use another term.

3.4.7 Focus on Modeling in Physics may be a Good Idea

According to interviewee A, adding more modeling (which is a part of CT) to the course could be beneficial. Usually, the pupils get a very mechanical relation to the calculations; they get a few formulas, then compose them. Instead, the pupils should be able to model phenomena, as physics should be more about understanding, not doing. The interviewee thinks the reason for the pupils doing this is that the course tradition has been this way for a long time. Thus, just by adding programming as a part of the course, modeling suddenly makes sense to do. The interviewee also mentioned that they had tried adding modeling to the physics course once in the past, but after trying it, they saw that it did not reach that far as they had hoped. This was mostly because the rest of the course did not fit with using modeling, and none of the pupils had any relation to programming from before.

3.4.8 Loops are The Most Difficult in Programming

Interviewee C suggested that loops are one of the hardest concepts to learn as a beginner programmer, using themselves as an example. When the interviewee was learning programming for the second time after 20 years, the interviewee noted that there was much easier to understand loops than the first time. The interviewee also observed that others, learning the exact same thing, had much more of a struggle and could not wrap their heads around the concept. The interviewee had also experienced this later when teaching others to program for the first time as well.

Functions are also a concept that is hard to learn for the first time. Thus it is not essential in the beginning or at any time, meaning it is mostly left out when introducing programming in non-programming courses.

Rich et al. (2018) also suggests that loops are hard to learn for beginners, thus distinguishes between how loops with conditions are more challenging to learn than loops with no condition, also known as forever-loops. Evidence also suggests that forever-loops are more favorable too among beginners learning Scratch (Maloney et al. (2010) and Guzdial (2020b), slide 47).

3.4.9 Some Pupils Have Used Programming in The Physics Course

Interviewee A, C and D had all experienced pupils that had used programming in the physics course at some point in their career, even though the pupils were not instructed to do so. Mostly, this was done to simulate a phenomenon, visualize, or automate a task. How this affected the pupils is unknown, but it made them explore the physics field on their own, which is something the physics teachers should encourage pupils to do.

3.4.10 The Future of The Physics Course

As the interviewees had very long answers to this part of the interview, this section is divided into one section for each interviewee and a summary section.

Interviewee A

Interviewee A envisioned that the pupils, in 10 years maybe, may already know some level of programming before enrolling in the physics course at USS. This would result in making the course more exploratory — not limited by the math that they are dependent on today. The interviewee also pointed out that the focus should be more on the models, as that is closer to reality, and that this will be easier if the pupils already know some programming. Thus, the interviewee claims that the course tradition, as mentioned earlier, could be an obstacle if these should be the future of the course.

However, to get to this future, the interviewee said that after-education of teachers would be essential. Also, to get programming into the course, one could start implementing it into oral exams as they do not require any major changes in the exam format. One should also start with courses for pupils so that they can learn some programming basics before using it in the physics course. The interviewee also claimed that if the pupils do not know some basic programming they will “(...) just see syntax.”

Interviewee B

Interviewee B divided the future into three scenarios: best-case, likely-scenario, and worst-case. The worst-case was that programming would die out, and nothing changes, and this could happen in reality in some places. The interviewee spoke from experiences from universities in the US, claiming that some places will be very resistant to changes. The best-case scenario would be that computational literacy (CL) will become widespread. The interviewee was also pushing towards this in their research, such that “(...) computation is going to become an essential skill, just like reading and writing and sort of basic mathematics. And that every student will be able to write a for-loop. Every student will be able to assign a variable. Every student will understand the bare minimum of what a program is in the form of step-by-step instruction, what an algorithm is.” Thus, the middle scenario — what the interviewee expects to happen — is that one

will see interesting things with computation, and some universities will push this forward, and others will look similar to what they are doing right now.

To get to the best case, the interviewee had a lot of ideas. Firstly, it is important to publicize why it is important to become CL. One would always need some level of hype to make people understand the reason. Secondly, one would need good computational tools that are free, easy to access and work with, and that everybody can use. Thirdly, teachers should know how to teach computation and not teach computation, as there are a few caveats. Lastly, the interviewee pointed out how making computation relevant is essential for success. By making enough computational tools, one could cover enough angles into computation such that people with any background would find it relevant.

Interviewee C

Interviewee C hoped that would at least have included air resistance by the time of 10-20 years into the future. The interviewee also hoped that one would use more models with discrete values such that not everything had to be solved analytically. Another effect of this would be that more pupils would like the physics course, not only the pupils who are very skilled in the analytical part. The best-case was that pupils choose the physics course because it is a course where pupils can play with computers to explore concrete examples. On the other hand, the worst-case is that the teachers do not engage in this transition and that the programming examples given will be created because “(...) it had to be done.” And that there would be fewer pupils taking the course. Thus, the interviewee thinks the physics course will end up in the middle of these two cases, arguing that it is likely going to be a course for more pupils as programming will open up new ways to learn physics, implying that more pupils will fit in.

However, to get to the best case, as told by the interviewee, one would need to get “(...) how to teach programming in physics (...)” into the teacher education. It will not be fine just to learn Python; the teachers need to learn to teach it. The interviewee also added that it would also help if the pupils learned basic programming earlier, thus to succeed, the teachers would need to know how to teach this.

Interviewee D

Interviewee D also used the best-case, middle-case, and worst-case approach. The best-case was that programming would be more integrated into the physics course. Programming should be used to simulate, model, solve more realistic problems, and encourage pupils to explore. Additionally, the pupils have some basic understanding of programming from before. The middle-case would be that the pupils would have a varying level of knowledge in programming, resulting in varying outcomes. The last case, and the worst, would be that the teachers lack the ability and motivation to make a real change. For the transition to have a positive outcome, it needs to be a reasonable way to assess programming competence in a written exam — else it will not be taken seriously by the schools. As quoted from the interviewee, “Many schools and teachers need a powerful push.”

However, for this change to be a reality, the interviewee said that programming needs to be included in the lower levels of education. Further, the schools need to know what the universities are currently doing — how they are teaching Python and how they create programming tasks. Also, methods in physics are more dependent on programming today, which teachers should learn from. Moreover, one could “sell in” the programming activity to the physics teachers by having workshops.

Summary

- The best case for the future is that programming comes into the physics course, such that the pupils can explore more realistic phenomena, and achieve more modeling. This could also result in more diversity among pupils enrolling in the course as it becomes less analytical.
- The worst case is that schools will continue to do the same as they do today, in the future.
- To get to the best case, there were multiple suggestions:
 - After-educate teachers. (This is already being done in ProFag; thus one may be needed to educate specifically for the physics course too.) (suggested by [A](#), [B](#), [C](#), and [D](#))
 - Pupils need to learn basic programming before the physics course. (suggested by [A](#), [C](#), and [D](#))
 - Teachers need to learn how to teach programming in the physics course. Not only learn how to program. (suggested by [B](#) and [C](#))
 - Make teachers include programming in oral exams. (suggested by [A](#))
 - Publicize why programming is important. (suggested by [B](#))
 - Have good tools that are free and make it easy to access, use, and work with using programming. (suggested by [B](#))
 - Teachers should learn how to not teach programming. (suggested by [B](#))
 - Focus on making programming relevant for all. (suggested by [B](#))
 - The schools need to learn from universities, as they already have experience with programming in physics education. (suggested by [D](#))
 - Could “sell in” programming activities through workshops. (suggested by [D](#))

Chapter 4

Literature Review

This chapter goes through theory, background, and related work in this field. The theory section (4.1) consists of how humans learn and how teachers should teach programming in physics. Next, the background goes through early work on this field and how that applies to the field today. At last, the related work section (4.3) covers the state-of-the-art in this field with recent research over the last decade.

4.1 Theory

The theory has been divided into three parts: learning (subsection 4.1.1), learning environments (subsection 4.1.2), and programming in physics education (subsection 4.1.3). The learning part contains definitions of terms that are used when discussing learning. The learning environment part focuses on

4.1.1 Learning

There are a few theories about learning that are important to mention in this thesis.

Constructivism

Constructivism is a cognitive theory, invented by Jean Piaget, and an educational philosophy that is often used to describe how learners construct their own meaning for everything that is learned (Cakir (2008); On Purpose Associates (2011)). The difference between the theory and philosophy is that the educational philosophy also specifies that learners construct their own *unique* meaning (Guzdial, 2018), meaning no one ever will have the exact same construction of the same meaning. Thus, despite the precise definition, the general idea is that it is tough, or even impossible, to recreate the same meaning from a teacher's mind to their learners. This also implies that more methods are needed for teaching new concepts, such that the teaching is more adapted to how the learners construct new meanings.

Constructionism

Constructionism (note the difference) is another term that is used throughout educational literature. The term was invented by Seymour Papert and is more of an education *method* that is built upon the constructivism cognitive theory. He believes that if learners will be more engaged in their learning if they construct something that others will notice and give feedback on, which in turn will make them willing to solve problems and learn as they get motivated by their own construction (Guzdial, 2018).

In Guzdial (2020a), Guzdial argues that he is convinced that constructionism is not the way to use computing in education anymore. One argument he uses is that the inventor of the term only focused on a small group of people interested in code and mathematics, and excluded the teachers. Therefore, it is preferred to use the constructivism theories instead of the constructionism method in this thesis, as the teachers are included.

Active learning

Active Learning is based on constructivism, which is defined above and aims to make the learners their own creators of their own understanding, or intellectual structures, as Seymour Papert puts it in his book, Papert (1980). Cambridge Assessment International Education (2018) also gives modern and a more detailed overview of what active learning is, suggesting that it is a process that focuses on *how* the learners learn, and not just *what* they learn. Moreover, they claim that the teachers should be the ones giving this opportunity to their learners. They also link this theory with relevance, which is defined next.

Relevance

Bernard (2010) suggests that making lessons relevant is crucial for learners. The learner's brain is more capable of learning new knowledge based on what they have from before; they are also more likely to remember it. An alternative to making the lessons relevant, which is much used, is to use a method that some refer to as drill-and-kill; the teacher makes the learner repeat a task until their curiosity is killed. Some would argue that repetition still is important, and evidence shows that it works, but combining it with relevance has shown to be truly powerful in learning (Claypool et al., 2004).

4.1.2 Learning Environments

Learning environments are important for adding context and relevance to the learners. This part defines a few terms that are used in this thesis when describing such an environment.

Programming Environments

A programming environment (PE) is a closed system that attempts to make it easier to use programming for specific tasks, providing the user with all the tools

they need to solve the task (Steven P. Reiss, 1994). To make programming easier, the PE abstracts parts of the computer’s inner workings such that the focus is on the task that the PE helps to solve.

Task-Specific Programming Environments

A “Task-Specific Programming Environment” (TSPE) is a PE that is tailored for a *specific task in a domain* (Guzdial, 2019). An example could be a PE that is only able to solve matrices or modify sound. The TSPE does not make it possible to do other tasks, like a general-purpose PE would do, which is described in the next definition.

However, the term should not be mixed with “Domain-Specific Programming Environment” (DSPE), which is close to the definition of Domain-Specific Languages defined in Kosar et al. (2012). The difference between TSPE and DSPE is that DSPE covers a whole domain — like physics — while TSPE covers only a specific task in that domain. DSPE cannot focus on a single task, meaning it cannot focus on a single task, making it closer to a general-purpose PE. Guzdial and Naimipour (2019) and Guzdial et al. (1997) also attempts to define Task-Specific Languages, which are programming languages that are tailored for a specific task. This could have been relevant to use in this thesis as well, but the Norwegian USS has indirectly decided to use the Python programming language by using it in all the new material that will be used in the new curriculum (interviewee C and ¹).

General-Purpose Programming Environments

A “General-Purpose Programming Environment” (GPPE) is a PE that is usually used by today’s developers. The PE is different for each developer, depending on what they program, but the essence is that the developers can solve any problem using this PE. The downside of using a PE like this is that this way of working needs some knowledge of computers beforehand and may not, therefore, be not a good choice for first-time programmers that should focus on learning a specific task.

Computational Essays

A “Computational Essay” (CE) is a PE that is mostly used by today’s scientists (Hannay et al., 2009). CEs make it possible to mix code with documentation in a linear fashion, creating a story — or an essay.

One of the most popular CEs is Jupyter Notebook (see more in [section 4.3](#)), formerly known as IPython Notebook, which makes it possible to write documentation using Markdown², LaTeX³, or just plain text, and add code

¹UiO Recommending Python: <https://www.uv.uio.no/forskning/satsinger/fiks/kunnskapsbase/realfaglig-programmering/5.-undervisning-i-realfaglig-programmering/>

²Markdown syntax: <https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>

³LaTeX syntax: <https://en.wikibooks.org/wiki/LaTeX/Basics>

blocks that can run Python in between. The code-blocks also transfer the program's state to the next code block, making it behave like a single python file. The code blocks can also deliver outputs in different formats, like text and images. The PE is also able to export the content, including the output from each code block, in multiple file formats. This feature makes it possible to share the results with other people. This is also what makes it useful for scientists working with computation.

4.1.3 Programming in Physics Education

Programming has been used in a higher level of physics education over the last past 50 years (Martin, 2016, p. 1). This has resulted in many theories regarding how to correctly teach programming in the physics course as well as how it is introduced initially.

Differential equations: the problem

Usually, when doing a higher level of physics, it is not possible to avoid differential equations (DE) to solve a problem if you tend to solve the problems analytically. It is not important to know exactly what a DE is in this thesis, but a short version is that it is a single equation that handles multiple functions and derivatives. Usually, this is used in physics as it is well suited for describing phenomena in the physical world.

At Norwegian USS, the pupils do not learn how to use DEs to find formulas for different phenomena. Instead, they receive the results from different DE solutions. This means they get premade equations for all cases they need in the physics course. While this makes it much easier for the pupils to use, they are limited by not expanding into more complex phenomena. This has also shaped the physics course into certain directions, leaving out friction and other realistic scenarios, that may be relevant to the pupils.

Thus, since the physics course is now able to use computers in its teaching, it is possible to use a different method for solving the exact same problems that enable the physics course to include "(...) the more fun parts of physics," as quoted from interviewee A.

Euler's method: the solution

One of the methods that have been revolutionizing the field of programming in physics education is Euler's method. There are other methods like this as well that extends this approach. These are categorized as the Runge-Kutta⁴ methods. The other methods may be more precise, but from an educational perspective, Euler's method is much easier to learn and use, and reduces the cognitive overload in the learning process (de Jong, 2010).

By using this method, it is possible to translate any problem that depends on a single variable, e.g., t for time, into a form that makes it solvable by a computer through many iterations. For example (see Equation 1), to calculate a position,

⁴The Runge-Kutta family: https://en.wikipedia.org/wiki/Runge-Kutta_methods

s , that changes based on an arbitrary function, v , the result would be to end up with a quite complex problem — maybe even an unsolvable problem. It depends on what v is. On the other hand, if the problem was solved using Euler’s method, it only needs to change the form into [Equation 2](#), where it is not needed to know what the v function is. The method only requires the learner to decide the initial conditions, which is essential in both methods, and Δt , which is the time step deciding how many iterations that need to be done.

$$s'(t) = v(t) \tag{1}$$

$$s(t_{i+1}) = s(t_i) + v(t_i) * \Delta t \tag{2}$$

The side effects of using this approach are that you may get a lot of errors⁵, and will get an approximate solution — not exact as the analytical version would, but by making the computer use millions of iterations, you can get really close to the exact solution. The method is also quite intuitive and make it easy to simulate a phenomenon, creating understanding more effectively (Flannery, 2019).

4.2 Background

This section presents early work that has shaped this field from the beginning. This section also introduces some terms as well as new ideas that are used throughout this thesis.

4.2.1 Using Computers to Learn

In the 1960s, the computers had just started to be used for learning (e.g., LOGO in 1967, Papert (1980)). The computers were not very available at the time, but with a growing interest and future imagination, a few foresaw the potential of using computers as learning machines. There was especially one researcher, Seymour Papert, who envisioned the potential. He did a lot of work on this topic leading to a programming language, or a microworld (EduTech Wiki, 2020), as he also coined the term to be, which closely resembles a programming environment (PE). This microworld was called LOGO and is still very much alive today⁶. In the 1980s, he released a book that he called *Mindstorms* (Papert, 1980) that became quite popular (over 14000 citations in 2020). Educational researchers are even still recommending reading this book. Even I, personally, got recommended this book by one of the interviewees. Anyway, in this book, he goes through his mindset of how the future will be, especially with focusing on using computers as an educational tool. Surprisingly, his predictions were pretty accurate, which it is possible to read more about in Resnick (2012). He also describes how he wanted the learners to be “(...) the active builders of their own intellectual structures” (Papert, 1980, p. 19). He then argues that computers are an important

⁵An error, like in: $value_{approximate} + error = value_{exact}$

⁶For example, Turtle graphics: <https://docs.python.org/3.3/library/turtle.html?highlight=turtle>

step towards this, and how a microworld can contribute to this type of learning. LOGO was a result of this idea and proved that children could be their own creators of intellectual structures. Interestingly, he mentions how the LOGO microworld is built upon mathematics, physics, and linguistics, teaching children about Newtonian physics in a natural fashion (Papert, 1980, .p 27), among other concepts. As of this property of the microworld, the learners are then able to learn more advanced topics as they have a microworld where it makes sense to learn it. This idea is something that is still useful today when designing educational tools for physics and mathematics.

There are also two other researchers that are worth mentioning from this period: Andrea diSessa and Hal Abelson. They were both into the same idea as Papert, but decided to create an alternative to LOGO that they called the Boxer Programming Environment, DiSessa and Abelson (1986). The difference is that their “microworld” is built based on graphical elements. While LOGO was purely based on ASCII characters, Boxer created a mental model of the program using, as its name suggests, “boxes”. See Figure 4.1. DiSessa argues that Boxer is better for beginners than LOGO in many ways⁷, with one of them being that Boxer is much faster to learn than LOGO. To summarize this part of the story, having a mental model of how the program works in the environment could be really beneficial.

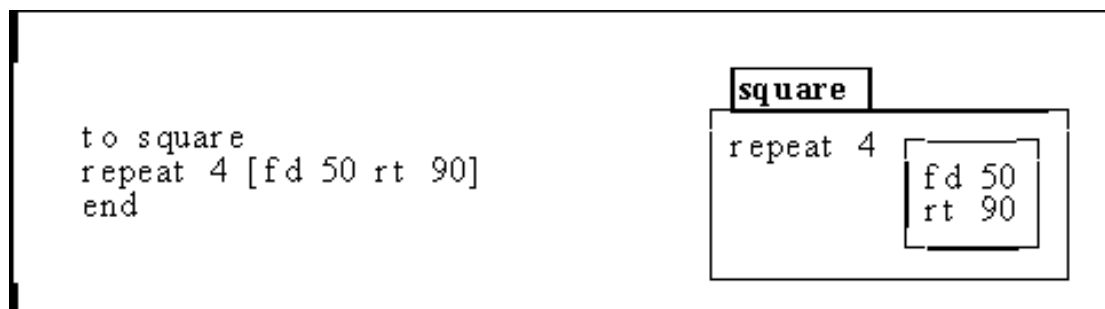


Figure 4.1: LOGO (left) VS. Boxer (right).⁷

This was before 1990. It would seem like research had understood a lot about learning and how humans could use computers to learn better, but what has been done since then? According to Resnick (2012), the children did not become fluent with new technologies as Papert envisioned. However, Resnick’s research group saw this as a problem and started developing a new microworld, or what they called: a visual programming environment, named Scratch⁸(Maloney et al., 2010). As the LOGO and the Boxer microworlds both got less used, and failed to explore “powerful ideas” the way Papert wanted (Resnick, 2012, p. 1), Scratch made a huge step forward. Scratch expanded on the idea of Boxer, using even more graphical and logical elements. Though, it was even more inspired by LEGO and LEGO Mindstorms (actually named after Papert’s book, Mindstorms: Papert (1980)). The research group got the inspiration from how children were able to start tinkering with LEGO without learning and be their own creators of their

⁷Twenty Reasons Why You Should Use Boxer (Instead of Logo)*: <https://eurologo.web.elte.hu/lectures/dis.htm>

⁸Scratch website: <https://scratch.mit.edu/>

own intellectual structures. Scratch was therefore built to resemble LEGO bricks (Resnick, 2012, p. 1-2).

Personal experience with Scratch

Scratch is very much used today. I, myself, must admit that I was also tinkering with Scratch from around the year 2007 to 2011, making me a product of the microworld itself. It made me especially good at some more advanced math topics, like trigonometry and Newtonian motion involving gravity, even though the calculations were not scientifically correct. Though, I got the idea and interest, which later made me choose a more scientific path. It is also important to note here that I saw the relevance of learning physics before actually learning it formally in school. I became an active learner creating my own intellectual structures to understand more of the world. And this is exactly some of what Seymour Papert was trying to make children achieve.

4.2.2 Using Programming in Physics Education

Programming in physics has been taught in education even as far back as 1962 (Martin, 2016, p. 2), thus mostly on a higher level of education through initiatives of eager physics teachers. At that time, computers were less accessible, and only a few universities owned one.

Later, around the 1990s, as computers became more accessible. Programming began to appear in broader parts of the physics education community. Some universities even offered computational physics degrees — also called dual degrees Martin (2017).

After the 2000 shift, computation became an essential part of physics education. It was even seen as a research methodology at the same level as theory and experiment (Martin, 2017). However, using programming in introductory physics has not grown at the same pace. As of the tools developed in the previous effort of joining computers in physics, it has been difficult to see the need for new tools suited for novices.

4.3 Related Work

This section discusses related work that has been done over the last decade in this field. How the information was found is described in [section 4.4](#). The section is divided into three parts. The first part, [subsection 4.3.1](#), is about PEs that can introduce programming or is able to teach physics using programming. The second part, [subsection 4.3.2](#), focuses on the programming that is taught in education and what the best practices are when doing so. At last, [subsection 4.3.3](#), focuses on the creation of tasks. The last part is based on studies on how to create tutorials.

4.3.1 Programming Environments

Over the last decade, there has been an increase in PEs that focuses on different uses of programming. However, only a few PEs focuses on introducing

programming in introductory physics. Of the time writing, Tychos was the only option that was found. And due to the lack of studies on Tychos, finding related work in the form of projects or studies that overlaps with this field was necessary to find enough information. This information was found from two fields: OLPs that use PEs to introduce programming, and conceptual PEs used to teach physics (not necessarily on an introductory level).

OLPs that introduces programming

Before looking into these OLPs, it is important to know what an OLP is and why they should be used to teach programming. OLPs are usually browser-based frameworks that can provide tools and information to teachers, learners, and other stakeholders in an educational environment⁹. According to Albashaireh and Ming (2018), which has done a study of seven OLPs, there are several advantages and disadvantages of using OLPs. Some of these are listed in [Table 4.1](#). By using an OLP, the teachers do only need to find suitable programming tasks for their learners. They then do not need to rely on their teaching skills in programming to be successful.

There was found a number of OLPs that contained PEs for introducing programming. To start out, there is Codecademy¹⁰. Codecademy is an OLP that only teaches programming. This means they do not teach other domains at the same time, like physics, even though they might have the right tool to do so. However, looking from a teacher's point of view, this might be hard to use. Firstly, it is not possible to monitor the learner's progress. At least not for free. Secondly, it is not possible to modify their tasks. As every teacher has their own way of teaching, modifying the programming courses to fit their teaching might be preferable. Thus, besides their disadvantages, they also have the same advantages as a general OLP. They also provide a PE that is specific to the programming they are teaching, making learning programming very easy. Looking more into how they do that, it is possible to note what elements they are using in their PE: (a) they provide instructions on the left side, (b) have a code editor in the middle, and (c) output on the right side, including graphical output or textual output, depending on the task and programming language.

Next out is Khan Academy¹¹. Khan Academy is almost the same as Codecademy, but with a wider variety of tasks — not only programming. But in this case, the focus will still be on the programming task — another difference is that they are more focused on the school curriculum and the classroom setting. I.e., compared to Codecademy, Khan Academy is teacher-friendly. They have even a full integration with Google Education¹² such that teachers can have their students at one place, and just give them assignments through Khan Academy. But there is one catch; teachers cannot create nor modify tasks that are given to their learners. This means that if the Khan Academy does not have the topic the teachers are looking for, they must avoid that topic or find solutions

⁹<https://eliterate.us/what-is-a-learning-platform/>

¹⁰<https://www.codecademy.com>

¹¹<https://www.khanacademy.org>

¹²<https://edu.google.com>

Advantages	Comment
Enables automatic grading.	Giving feedback to the learner as soon they have done a task makes them more likely to learn from their mistakes. Teachers can also see what the learners are failing on faster, being more able to correct the learners understanding of specific topics before moving on.
Learners may be more involved.	OLP makes the learners active users and in control of their learning.
Content management can be integrated.	Teachers do not have to deal with documents nor create their own system. The system supports the teachers by making it both easy and systematic.
Can integrate tools that creates assignments and track progress of learners.	Support teachers with tedious tasks. This benefit has a lot of potentials. This is usually where the OLPs are unique.
Createable and sharable content	Enables teachers to quickly share content they have made, in any setting, making it fit with their teaching.
Content accessible anywhere	Enables learners to learn how they prefer, not limiting them in any way as long they have access to a computer with a browser and internet.
Disadvantages	Comment
Requires internet.	This can be partly solved by making the OLP runnable offline with downloadable content.
Current OLPs does not personalize.	This can be solved using more modular parts, making the teacher more in control of the system.
Due to scalability, the teacher cannot give realtime feedback to everyone.	This will may not be possible to solve until we make artificial intelligence suited for this task.

Table 4.1: Some advantages and disadvantages of using OLPs (Albashaireh and Ming, 2018, p. 632) combined with our comments on them.

elsewhere. Besides from that catch, they provide a really good solution for learning programming. Not only do they give (a) instructions, (b) have an editor, and (c) a graphical output, but they can also have (d) audio that goes through how to code while changing the code in the editor. This enables the students to pause, change the code from that was instructed so far, and check if their understanding was correct, then resume. It will not keep the code, as it works as a video, but it makes it easier to get hands-on experience with coding while seeing how the instructors live code, which is a lot of learning in itself (Brown and Wilson, 2018, tip 3, p. 2).

Last out, there is a more scientific related project that has more focus on

sharing PEs: Jupyter Notebook. This is, as mentioned in the theory section (section 4.1), a computational essay (CE). This is a highly modular programming environment that works more like a document rather than an OLP. It intertwines text and runnable code blocks together in a linear fashion. However, by adding a plugin, called *nbgrades* (Blank et al. (2019); Hamrick (2016)), you can turn this powerful document into a fully working OLP, with an automatic assessment of code blocks and options for personalized feedback. This plugin has become quite popular among teachers¹³. This may show that making it easy for teachers to create programming tasks, and give them control of their learners, is a good idea. The unique thing with Jupyter, which none of the other OLPs have, is that creators can order the content any way they want. They do not require a specific format or structure. And with *nbgrader*, teachers can assess anything in the code blocks. There are still a few disadvantages, though. First of all, it is not optimized for beginner coders. Teachers can create CE tasks that are easy, but the CE in itself is not made for beginners like Codecademy and Khan Academy is. Both Codecademy and Khan Academy uses its own workforce to tailor new courses such that the programming environment is optimized for beginners. Secondly, if the school does not create an easy-to-access Jupyter Notebook, like JupyterHub¹⁴ enables, all the learners have to download Jupyter themselves. This is very often the case in Norwegian USS (told by interviewee A). Anyway, this comes with several technical challenges, like installing Python and Pip or Conda¹⁵, dealing with the filesystem (verified by interviewee A), and having to transfer Jupyter CEs from teacher to their own environment as it cannot be shared through the hub.

Conceptual programming environments that teaches physics

Through the SLR process, described in section 4.4, it was possible to find a few conceptual PEs that had physics teaching as their main goal. The reason for naming these conceptual PEs, it that they do not have a published version of their prototype. But their PEs are well documented in papers, making it easy to study how they work and what their goal with their approach was.

Kitagawa et al. (2019) created a PE named STEPP that was aiming to teach modeling and simulation in 1D and 2D. Their goal was to introduce modeling and CT into high-school physics. Their approach was to create a PE that could model finite state machines (FSM) through visual programming that could be transferred into solving different problems. The FSM could then animate the movements of objects. I.e., they included simulations of graphics. They also provided the option to show graphs of different variables. Their animation is also scaled to finish in a reasonable time. They use a client-server architecture for their OLP, but the PE is running on the client-side — not communicating with the server.

Hutchins et al. (2019) also uses visual programming to achieve their goal, which is also to introduce CT in physics. As opposed to STEPP, the PE used in this study, C2STEM, uses Scratch to model. Their approach is to use customized

¹³Over 10'000 projects on GitHub (Blank et al., 2019).

¹⁴<https://jupyter.org/hub>

¹⁵How to install Jupyter: <https://jupyter.org/install>

blocks¹⁶ that are domain-specific, e.g. “simulation step”. They have named this new version of the Scratch programming language domain-specific modeling language (DSML). This behaves like a programming language, but with special functions running, e.g., every tick in the physical time. Further, they enable the use of graphics and graphs to visualize the phenomena that are created.

Orban et al. (2018), Orban et al. (2017), and Orban (2017) have the same goal as the two previous papers, except for that they focus on programming instead of CT. They also have an approach that involves textual programming of simulations instead of visual programming for modeling. To achieve programming of simulations, they use skeleton code that the learners should write code into. The code is written in JavaScript using a library called p5.js¹⁷ to abstract having to use advanced JavaScript to create graphics as well as not handling time to simulate phenomena. To organize the code, they have focused on having areas in the skeleton code that serve different purposes. At the top, they specify initial conditions, like time steps and constants. Below, a draw function is run every time the simulation is updated as of time. In that draw function, the learners should first update physical properties, then check for user interactions and make a change depending on that. At last, in the draw function, it is possible to add graphics. This structure is not enforced but rather recommended for the learner. However, they have observed that there is a downside of updating physical properties first and then checking for user interactions, as learners tend not to understand that the draw function passes the state of the variables into each iteration. The same structure was also found to be used in Caballero et al. (2012), p. 4, thus without user interaction. As Caballero et al. (2012) did not mention the implications of their structure, the rest of the structure without user interaction may be a good approach when designing a PE suited for teaching physics.

Programming environments that teaches physics

At last, it is important to mention Tychos. This OLP contains a PE that closely resembles what is trying to be achieved in this thesis. Their approach was to design their own programming language that is specifically made for physics. In other words, this programming language is a domain-specific programming language (DSPL, see [section 4.1.2](#)). Further, they have added graphics as their main element in the design, with graphs on the side, if the code enables them. They also have two code editors. These code editors enforce the same structure as Caballero et al. (2012) by having one for initial conditions that run initially, then an editor that is run for every delta time. This means that they abstract the creation of loops. They also abstract time steps, dt , and time duration by not requiring them to be set in code. They are set in the settings instead. Moreover, they support tables, inbuilt physical objects (like particles), vector representation of properties of physical objects (like velocity and acceleration), and custom drawings.

¹⁶Scratch blocks: <https://en.scratch-wiki.info/wiki/Blocks>

¹⁷<https://p5js.org>

4.3.2 Teaching Programming

To get an overview of how to teach programming in an educational context, this part is divided into two areas. The first area focuses on how programming is taught in computer science (CS), as that is a well-documented area. The second area focuses on how programming is taught in introductory physics, specifically, though it may include higher levels of physics.

Programming taught in computer science

Over the last decade, there has been an increased effort to teach programming on a lower level. This has, in turn, led to studies and suggestions on how teaching programming should be done.

Brown and Wilson (2018), summarizes the last decade with ten quick tips for teaching programming. In tip 6, “Use worked examples with labeled subgoals”, they explain that a step-by-step guide with worked examples is a good way to guide the learners. Long step-by-step guides may make it hard to learn the content. Usually, some instructions are repeated across multiple guides as well in the same guide. By adding labeled subgoals, the content is grouped into sections that are a logical set of instructions. This has further been proved to enable the learners to recognize patterns that are isomorphic in a long step-by-step guide, making it easier to learn (Margulieux et al., 2012; Morrison et al., 2015, 2016).

In tip 7, they suggest that the schools should stick to one language. This is because: “Attempting to force transference too early — e.g., requiring them to switch from Python to JavaScript in order to do a web programming course early in their education — will confuse learners and erode their confidence.” (Brown and Wilson, 2018).

In tip 8, it is suggested that to make learners engaged in learning programming, it is important to abstract code that is not relevant to the task. The faster the learner can get to the goal, the more enjoyable the tasks are. In other words, do not make the learners write all the code themselves unless there is a library that can help them achieve quicker results.

At last, in tip 9, they found a recurring issue that beginners are treated as experts. The reason why this is an issue is that beginners program differently than experts. To solve this issue, a set of tools and approaches suited for beginners should be made. This suggests that creating a customized PE could be beneficial when learning to program in physics too.

Freeman et al. (2014) suggests that active learning (see [section 4.1.1](#)) is more valuable than lecturing in CS. This may suggest that designing an OLP with a PE could be better than having a teacher explain programming.

Programming taught in introductory physics

When it comes to teaching programming in an introductory course in physics, there was found a few attempts.

Bensky and Moelter (2013) attempted to teach programming in physics using a bead on a wire. They decided upon this phenomenon as it could be solved numerically, but also had some analogies with the traditional analytical

approaches that are usually taught in introductory physics. The phenomenon could also be justified to be harder on the numerical part, or harder on the analytical side. This was done by changing the wire to either have sharp edges or smooth edges. By having a smooth edge, the task had to be solved numerically as it required a lot of calculations.

In Orban et al. (2018), there was created a PE that taught introductory physics. However, their creation was in collaboration with PICUP¹⁸: Partnership for Integration of Computation into Undergraduate Physics. PICUP attempts to unite the understanding of how to teach programming in physics courses. What is interesting with their website is that they provide exercise sets from all around the world in a single collection. This makes it easier for physics teachers. However, the tasks added to the collection are made in a typical file format and do not divide the tasks into smaller tasks that can guide the learners. This means that PICUP is not meant for novice programmers. And as stated in [section 4.3.2](#), novices need tools suited for them. Anyway, PICUP can still be an inspiration. The tasks just need some modifications such that they can be run in a PE suited for novices.

Marciuc et al. (2016) found that using GeoGebra for mathematical modeling, then VPython for numerical modeling, made students “(...) understand some abstract concepts, like force, speed or acceleration, (...)” and increased the student’s motivation for learning. They also found that working with computer models of what they modeled through abstract mathematical equations contributed to “(...) deepening of the student’s knowledge and helped them discover the physical significances (...)”. This means that it may be an idea to divide modeling into an analytical and a numerical part.

Flannery (2019) explains how physics education has a missing potential that can come from using computers, and that Euler’s method is the way to go to enable this potential. They also demonstrated planets that moved in inhomogeneous force fields, and how easy the numerical approach could be.

Further, there were found a few projects that were attempted in introductory physics courses. These are introduced with references in [chapter 5](#).

4.3.3 Creation of Programming Tasks

When attempting to find information about how to create tasks for PEs, there were no studies that included creating tasks in specific PEs. Most attempts on designing tasks, like PICUP (Orban et al., 2018), does only include skeleton codes with no extra steps explaining and helping the learners. To find literature on this topic, it was possible to find studies on how tutorials were made. Tutorials are carefully thought through step-by-step instructions that guide the learner. As suggested in Morrison et al. (2015), worked examples with step-by-step instructions is a good approach for teaching programming, meaning it may be transferrable to the topic of teaching programming in physics.

Kojouharov et al. (2004) made an Eclipse plugin that they called JTutor. JTutor is a PE that makes it possible to follow a tutorial that is built up by

¹⁸<https://www.compadre.org/PICUP>

steps and substeps. The tutorial may even change the code the learner has, depending on what the next task is. But to get to the point of this study, they have also created JTCreator. This is a tool for creating these tutorials as they can be cumbersome and time-consuming to create. Their approach is to create a “creation and replay” environment for the task creator. This means that the creator code as they are used to, and JTCreator records their actions. When the creator is done with a substep or a step, they can take a snapshot of their actions and code and associate them with instructions. On the receiving part that sees the tutorial, their user interface (UI) consists of a list of the steps, as well as an own list with substeps in the current step. There is also a code editor that changes and contains highlights of important code that needs focus. This is very similar to how Codecademy and Khan Academy works. However, it was not possible to test JTCreator as it is yet to be published.

Zhang et al. (2009) proposes Smarttutor, another approach for creating tutorials. Like Kojouharov et al. (2004), they also use an “creation and replay” technique. Thus they have named it “editable replay” instead. It also only works in Eclipse. The minor difference is on the part of the receiver. Instead of providing a whole PE, they only provide a list of tasks that can float over the PE that the user has themselves. The list can also do actions in the PE, depending on if the user wants it to autocomplete the task. To create the task, they provide a UI that includes a list of actions and a list of steps. The list of steps is associated with a title, description. The UI is supposed to be aside with their normal Eclipse environment, making it not affect the creator’s PE.

Later, Zhang et al. (2010) again proposes an approach for creating tutorials, but this time, it is automated. The reason for this is to make it easier to create tutorials, but also make it possible to generate many possible step-by-step instructions, as all tutorials can branch into multiple paths. They use a “record and replay” technique, which again is the same as in Kojouharov et al. (2004), just with a different name, and combines it with mining comments and actions from experts. The mining is done by looking at sub-tasks of many tutorials and mix them with the associated actions. From the receiver’s part, they will do the tutorial like described in Zhang et al. (2009), but they also get recommended for new paths while in the process. The UI for creating tasks is also similar to what is proposed in Zhang et al. (2009), but it also adds new paths after the creation is done.

4.4 Method

To attain the information in the related work section above, a specific method called Structured Literature Review (SLR) was used. SLR, as defined by Kofod-Petersen (2015), is defining a process for finding related literature in a structured manner. It was first used in medical science; this was later adopted by other sciences as well, including computer science. The benefit of using this process for finding literature is that it makes the review reproducible while discovering a lot of research on the topic that may else have been missed.

Thus, it must also be mentioned that even though it was found many studies

in the field, a lot of valuable material was missed too. There have, therefore, been used both references from the SLR studies and used other search strategies to find a broader set of information. This is explained in [subsection 4.4.2](#).

4.4.1 Structured Literature Review: How It Was Done

In SLR, it is first important to define a protocol that is going to be used in the search phase. This includes defining which terms to use and where to find the information. Then, a lot of articles will then be collected using this protocol. If there are too many articles, it may be preferable to add more specificity in the search. Then, the filtering begins. This is done by removing one by one until only the relevant results are left. In the end, the classification of quality and inclusion criteria is done on the left results.

Defining the search protocol

To perform the search, there were chosen three key terms; *physics*, *computational thinking*, and *secondary school*. The main idea here is to find articles that overlap with physics, CT, or programming, and all kinds of education that introduce physics for the first time. It was also preferable to include a fourth element, *teacher created tasks*, in this thesis, but as it was hard to find any results, including this, it was left out intentionally as it returned too few results. Figure 4.2 attempts to explain what that was trying to be achieved.

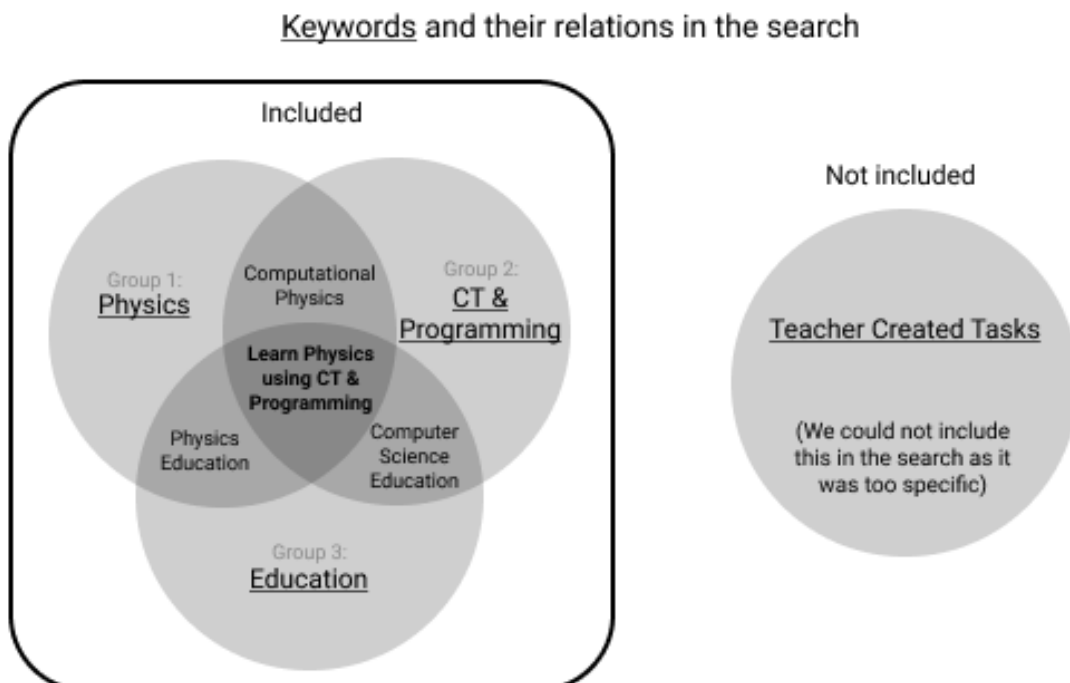


Figure 4.2: Our keywords and their relation in the search.

Based on these key terms, it was essential to find synonyms for each of them. This is done in [table 4.2](#). It was also chosen only to find results that were published

between January 2010 and January 2020. The search was only performed on abstracts, titles, and author keywords.

	Group 1	Group 2	Group 3
Term 1	physics	computational thinking	introductory
Term 2		algorithmic thinking	beginner
Term 3		programming	secondary school
Term 4		coding	high school
Term 5			pre-university
Term 6			STEM education
Term 7			K-12

Table 4.2: Terms and synonyms (groups) used in query. Table structure inspired by Kofod-Petersen (2015).

More specifically, the query is defined like a boolean search:






"physics" AND ("computational thinking" OR "algorithmic thinking" OR "programming" OR "coding") AND ("introductory" OR "beginner" OR "secondary school" OR "high school" OR "pre-university" OR "STEM education" OR "K-12")

By using quotes around all terms (""), it was possible to avoid that search engines will stem the search terms. I.e. "beginner" will not match with "beginning" and "begin" if quotes are added.

Retrieving the literature

For this SLR, five different databases were used: ACM Digital Library¹⁹, IEEE Xplore Digital Library²⁰, Science Direct²¹, Scopus²², and Web of Science²³.

The search in each database required different syntax to achieve the same query. For reproducibility, a list of links has been provided to each of the databases specifying all the criteria that were defined in [the search protocol](#), in the table below. The table also contains the number of results that were returned as of January 2020.

Database	Query	Results
ACM Digital Library	Link to query 	22
IEEE Xplore Digital Library	Link to query 	12
Science Direct	Link to query 	32
Scopus	Link to query 	136
Web of Science	Link to query 	83
Total		285

¹⁹<https://dl.acm.org>

²⁰<https://ieeexplore.ieee.org>

²¹<https://www.sciencedirect.com>

²²<https://www.scopus.com>

²³<https://apps.webofknowledge.com>

Selection of primary studies

After retrieving the literature using the protocol specified earlier, there was received 285 matches in total. Thus, as only studies from five separate databases were retrieved, many of the studies were duplicated. After removing duplicates, only 212 unique matches were left.

A lot of the studies were not relevant. It was, therefore, possible to remove the most obvious ones. This resulted in excluding 130 studies, reducing down to 82 potentially relevant studies.

Study quality assessment

In this step, a more thorough study of the studies needed to be done. All of them had some level of common ground with the field that was searched for, but only the studies that were *thematically* relevant should be included. To achieve that, it was needed to create a few criteria that defined what to look for in each study. The ones that did not match any of the criteria, or matched the excluding ones, were excluded.

The criteria, defined in table 4.3, were divided into quality criteria (QC), inclusion criteria (IC), and exclusion criteria (EC). The inclusion criteria were then again divided into primary and secondary groups, where the primary group contained the studies that were spot on the topic, and the second group contained the studies that had some important elements that were preferred.

Criteria ID	Criteria
IC1	Learning introductory physics using CT or programming
IC2	Focuses on learning physics (less focus on CT or programming)
IC4	Uses a introductory physics course in their evaluation
EC1	Focuses on learning CT or programming (not physics)
EC2	Focuses on higher level physics courses
EC3	Does not contain CT or programming
EC4	Learning introductory physics using other approaches
QC1 (t ²⁴ =0.5)	Is there a clear statement of the aim of the research?
QC2 (t=1.0)	Is the study results reproducible?
QC3 (t=0.5)	Does the study compare their approach with other approaches?

Table 4.3: Inclusion and quality criteria. Inspired by Kofod-Petersen (2015).

First, the results left went through all the inclusion and exclusion criteria. It started with a different set of criteria initially, but after seeing that the criteria were too vague, they got refined step by step, as well as more exclusion criteria, were added. It is possible to see the ending criteria in Table 4.3. However, this resulted in having nine studies left.

At last, it needed to evaluate them on quality. This was done by giving quality points based on the specific quality criteria. If a quality criterion was not filled, they got 0 points. If it matched, they got 1 point. If they partly matched, they

²⁴Points that are needed to pass the quality criterion threshold (t).

got 0.5 points. A threshold was then set for each criterion that decided which studies that should pass or not, based on their points on that given criterion. See the t-values at the bottom of [Table 4.3](#). Thus, after evaluating all the studies, they all passed the quality test.

Final studies

The studies that were included in the end result can be found in [Table 4.4](#).

Title	Reference	Year
C2STEM: a System for Synergistic Learning of Physics and Computational Thinking	Hutchins et al. (2019)	2019
Scaffolded Training Environment for Physics Programming (STEPP)	Kitagawa et al. (2019)	2019
A hybrid approach for using programming exercises in introductory physics	Orban et al. (2018)	2018
A novel approach for using programming exercises in electromagnetism coursework	Orban (2017)	2017
Developing students' creativity by Physics lessons	Marciuc and Miron (2017)	2017
Using GeoGebra and Vpython software for teaching motion in a uniform gravitational field	Marciuc et al. (2016)	2016
The effect of computer science on physics learning in a computational science environment	Taub et al. (2015)	2015
Computational problems in introductory physics: Lessons from a bead on a wire	Bensky and Moelter (2013)	2013
Implementing and assessing computational modeling in introductory mechanics	Caballero et al. (2012)	2012

Table 4.4: The studies that were included after the final selection phase in the SLR.

4.4.2 Alternative Search Strategies

As mentioned at the beginning of [section 4.4](#), the search was also expanded for studies by using less precise methods. This was because this field of research lacks common terms, as opposed to medical studies with a lot of unique terms for what they want to express. This may also be because DSR is the study of the artificial, not the natural. Studies using DSR usually create new concepts or ideas and have a hard time agreeing upon a good name. For example, researchers do not exactly know what computational thinking (CT) is yet (interviewee B), resulting in studies that have interpreted it differently, and excluding results that have the same idea but do not think CT is the right term to use. By using alternative search-strategies for attaining the results was therefore necessary for finding the state-of-the-art in the field, or at least the related work.

Another reason for including these strategies is that it was not possible to include the fourth search element: *teacher created tasks*, without the search

excluding too many of the good results. Using these strategies it was possible to find information on this field as well as in other fields nearby.

The reference strategy

The first alternative strategy was to use references from the studies that were found to explore more of the field.

Title	Reference	Year
A Game-Centered, Interactive Approach for Using Programming Exercises in Introductory Physics	Orban et al. (2017)	2017

Table 4.5: Studies found using references from SLR studies.

The Google strategy

The last alternative strategy is based on using different terms than the ones used in the SLR so far. For example, there were found many relevant studies that do not include the physics part, which was a crucial part in the SLR query.

Title	Reference	Year
nbgrader: A Tool for Creating and Grading Assignments in the Jupyter Notebook	Blank et al. (2019)	2019
The Coming Revolution in Physics Education	Flannery (2019)	201
A survey of online learning platforms with initial investigation of situation-awareness to facilitate programming education	Albashaireh and Ming (2018)	2018
A Principled Approach to Designing Assessments That Integrate Science and Computational Thinking	Basu et al. (2018)	2018
Ten quick tips for teaching programming	Brown and Wilson (2018)	2018
Modeling a Pendulum's Swing Is Way Harder Than You Think	Allain (2016)	2016
Novice programmers & the problem description effect	Bouvier et al. (2016)	2016
Creating and Grading IPython/Jupyter Notebook Assignments with NbGrader	Hamrick (2016)	2016
Subgoals help students solve Parsons Problems	Morrison et al. (2016)	2016
Subgoals, context, and worked examples in learning computing problem solving	Morrison et al. (2015)	2015
Active learning increases student performance in science, engineering, and mathematics	Freeman et al. (2014)	2014
Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications	Margulieux et al. (2012)	2012
Towards Automated Synthesis of Executable Eclipse Tutorials	Zhang et al. (2010)	2010
Smarttutor: Creating IDE-based interactive tutorials via editable replay	Zhang et al. (2009)	2009
Understanding The High Performance Computing Community: A Software Engineer's Perspective	Basili et al. (2008)	2008
Software development environments for scientific and engineering software: A series of case studies	Carver et al. (2007)	2007
When software engineers met research scientists: A case study	Segal (2005)	2005
JTutor: An Eclipse Plug-in Suite for Creation and Replay of Code-based Tutorials	Kojouharov et al. (2004)	2004

Table 4.6: Studies found using the google strategy.

Chapter 5

Answering Research Question 1.1

By studying recent work and interpret interviews and hearings, it is possible to find an answer to RQ1.1: “*What phenomena in physics can be programmed in the introductory physics course?*”. This research question will make sure that the artifact focuses on programming the right phenomena in the introductory physics course. In turn, this will guide what elements that are needed in RQ1.2 and what tasks that are made in RQ1.3.

This section has been divided into eight types of typical phenomenon taught in introductory physics and has been proven to be possible to program. Thus, some of these phenomena are not necessarily suited for an introductory course in physics or first-time programmers. Therefore it is important to find what is required from the pupils to program each task. For example, some tasks may require loops, which are very tough to learn (see subsection 3.4.8), for even the simplest version of the task. Others, on the other hand, may only require a loop when the problem is evolved.

As interviewee B mentioned, a great metaphor that is used when evaluating whether a computational task is good or not is that it needs to have a low floor, such that anyone can make an entry. Then the tasks need to have high ceilings, such that the learner can come a long way. Finally, it needs many windows, such that the learner can come from any angle. Therefore, each phenomenon will be evaluated by how the problem can be in its simplest form, avoiding as much knowledge of programming and physics as possible, such that anyone can make an entry. The evaluation will also consider how the phenomena can evolve into a harder problem, giving more challenges and higher learning outcomes for the pupils. In the last part, with many angles to entry, the phenomenon will not be evaluated. Thus a higher ceiling may automatically provide more windows, as more additional elements are added.

5.1 Falling Object

In this phenomenon, an object with a mass is usually either placed mid-air or thrown from the ground. The object is also typically affected by the constant of gravity¹. This phenomenon is further described in Marciuc et al. (2016). Thus,

¹ $g \approx 9.81m/s^2$

this constant does not need to be constant, nor does it need to be alone. Other forces like air resistance and wind can affect the object, too, and the shape of the object. There are many configurations to this phenomenon, making it one of the simplest phenomena ([Figure 5.1a](#)), to one of the hardest to solve analytically ([Figure 5.1d](#)).

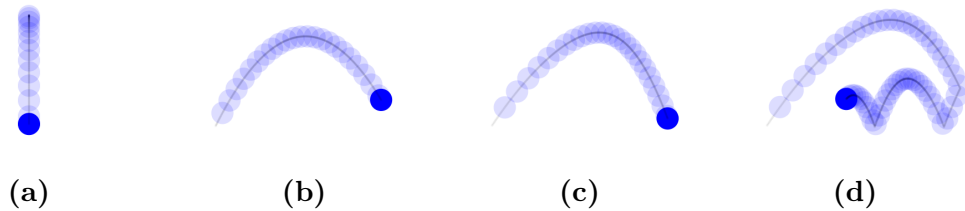


Figure 5.1: Example evolutions of the falling object phenomenon: (a) falling ball in 1D, (b) thrown ball in 2D, (c) thrown ball with air resistance, and (d) thrown ball with air resistance and bounce.

This phenomenon is quite popular to program and has been mentioned by interviewee [A](#) and [C](#), as well as in Marciuc et al. (2016) and Orban et al. (2017). Thus, usually, when this phenomenon should be programmed, air resistance is accounted for. The reason for this is that without air resistance, the phenomenon is easily solvable analytically through a single equation, see [Equation 5.1](#). To get a better idea of how this equation works, see [Figure 5.1a](#) and [Figure 5.1b](#).

$$\begin{aligned}
 y'' &= g \\
 \Rightarrow y' &= g * t + v_0 \\
 \Rightarrow y &= \frac{1}{2} * g * t^2 + v_0 * t + y_0
 \end{aligned}
 \tag{5.1}$$

However, when air resistance is added into the phenomenon, a drag force, D , must be added to the equation. See [Equation 5.2](#). This makes the equation much more tricky to solve, leading to less focus on physics and more focus on mathematics. This issue is also described in [section 4.1.3](#).

$$\begin{aligned}
 y'' &= g - \frac{D}{m} \\
 \Rightarrow y' &= g * t + v_0 - \int \frac{D}{m} dt \\
 \Rightarrow y &= \frac{1}{2} * g * t^2 + v_0 * t + y_0 - \iint \frac{D}{m} dt dt
 \end{aligned}
 \tag{5.2}$$

Thus, by using a computer, one can instead use the numerical approach, or Euler's method, in this case, to simulate the motion in each time step, Δt . This will turn [Equation 5.2](#) into [Equation 5.3](#), where the drag force, D , is only applied once and does not need to be calculated further.

$$\begin{aligned}
 a_y(t_{i+1}) &= g - \frac{D}{m} \\
 v_y(t_{i+1}) &= v_y(t_i) + a_y(t_{i+1}) * \Delta t \\
 s_y(t_{i+1}) &= s_y(t_i) + v_y(t_{i+1}) * \Delta t
 \end{aligned}
 \tag{5.3}$$

Now, by using the numerical approach and a computer, the phenomenon becomes quite easy to simulate for all cases described in [Figure 5.1](#). This is why this phenomenon is well suited for being programmed in the introductory physics course. And with a few minor modifications, all the other phenomena below can be programmed this way as well.

Moreover, the phenomenon can be transferred to multiple domains, adding more elements. For example, when throwing a ball outdoors, it is likely that some (a) wind will affect the motion of the ball. And when the ball hits the ground, it will likely (b) bounce a few times before it stops, because of friction (c). It may even (d) roll a little further after that, creating even more complexity to the phenomenon.

Considering how simple the task can start, and how complex it can evolve without adding unrealistic elements, this may be the safest phenomenon to teach in an introductory physics course.

5.2 Pendulum

The next phenomenon is describing an object with a mass that swings around a joint on a rod, keeping a constant distance from the joint. The phenomenon was mentioned by Marciuc et al. (2016). It is not an easy task to begin with. The reason is that it involves more forces than the gravitational pull. The extra force, which is from the pull of the rod, is also changing constantly, making it hard to model. These issues have also been described in Allain (2016).

Also, the phenomenon does not evolve that much, meaning when the learner finally has managed to get the dynamics working, there are not many easy additions that add much learning value. However, these four additions can be made: (a) air resistance, (b) multiple linked pendulums, (c) spring-loaded rod, and (d) a sudden change in rod length. The first and last additions are easy to add, the others can be very hard as it requires a solid control over the physical system, which in turn requires a lot of programming experience.

5.3 Block Down a Slope

This phenomenon is about a block that slides down a slope. In its simplest form, it is a block with no friction on a straight slope. The only force acting on the block is the gravitation. When evolving this phenomenon, one can add four elements: (a) friction between block and slope, (b) an uneven slope, (c) a ball instead of a block, and (d) a pulley with another object pulling it on the other side with gravity.

It is possible to consider the whole phenomenon as an extension to a falling object, as gravity is the only force working on the block. The only tweak is added when one has to account for the angle of the surface.

Considering how easy it is to start with, and how many ways it can evolve, this is a reasonable phenomenon to program in the introductory physics course.

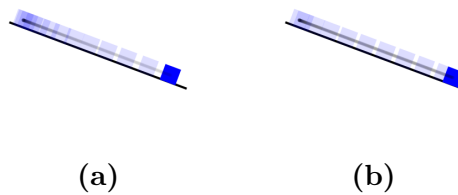


Figure 5.2: Example evolutions of the block down a slope: (a) simplest form, and (b) with friction.

5.4 Bead on a Wire

This phenomenon was inspired by Bensky and Moelter (2013) where they argue that a numerical approximation to this phenomenon gives much more in return to the learners than an analytical approach. The phenomenon itself is in general terms defined as an object that is constrained to move along a curved path.

The simplest form of this phenomenon is just like the previous block down a slope, but with an arbitrary curved path. This means that the learner must know how to constrain an object to a defined path. According to Bensky and Moelter (2013), this can be done through a mathematical function or a list of points. Anyway, this makes the phenomenon a bit more challenging to start with.

To evolve the phenomenon, there are at least two elements that can be added: (a) friction, and (b) measurement of energy, which gives a lot of understanding to the learners, according to Bensky and Moelter (2013). To add even more elements to the phenomenon, one can detach the bead from the wire; thus, the phenomenon is closer to the block down a slope phenomenon, and that breaks with the logic of calling it a bead on a wire.

5.5 Ball Collisions

This phenomenon is moving into another area in physics that is not only about acceleration, velocity and position, but also impulse, which is acceleration applied in a short time period, also known as a collision. The simplest version of this problem consists of two balls, with equal mass, colliding with each other once. There are no other forces that are constantly involved, like in all the previous examples. However, what may make this phenomenon hard to start with is that it involves an if-statement, checking whether or not the balls are colliding. Thus, only one collision check is needed for the simple version.

When evolving the phenomenon, there are about ten elements that can be added: (a) two dimensions, (b) multiple balls, (c) walls, (d) friction between balls, (e) rotation, (f) different masses, (g) different sizes, (h) gravity, (i) inelasticity, and (j) forces between each of the balls, like planets, have.

The phenomenon may be for an intermediate level as it involves an if-statement, but it should not be too hard as the other parts are very easy to understand. However, this phenomenon has quite a few additions that can make the problem interesting for all learners.

5.6 Electron on Uniform Magnetic Field

There are also a few interesting phenomena that can be simulated at the atomic level of physics. Many of the laws of physics at this scale may be hard to understand as they are more abstract and not visible to the naked eye; thus, visualizing them may make the learner better understand the concepts.

This phenomenon has been mentioned in both the hearing of the first draft of the new curriculum, by Orban (2017), and by Marciuc and Miron (2017), p. 474-476. In general, an electrically charged particle moves through a uniform magnetic field. Thus, when the particle is inside this field, depending on the velocity, the charge, and the field, the particle will be forced to move along a specific path.

To set up this phenomenon, only one object is needed, and that is the particle. It is a very simple problem if the learner already knows physics. However, to get a better understanding of the particle's movement, it is important to see the third dimension as well. This may make the problem more dependent on visualization tools, but it should not be much harder to program.

There are also not many additional elements to add to the phenomenon. Thus one could change the parameters while simulating.

5.7 Planet Orbits

This phenomenon focused on describing the motion of the planets in the universe and was mentioned by Caballero et al. (2012), p. 4. It consists of at least two objects with a mass. These objects are attracted to each other, making them affect each other. The most simple version of this phenomenon would be that both object have the same mass and is equally pulling on each other. The most simple version could also be in one dimension, but it is much more rewarding when the objects go around each other in two dimensions.

To evolve this phenomenon, four additional elements could be added: (a) different masses, (b) multiple objects, (c) collisions joining two objects into a more massive one, (d) the third dimension.

5.8 Asteroids Game

This last phenomenon is a game that uses the same physical laws that is taught in an introductory physics course. The phenomenon was proposed by Orban et al. (2017), which pointed out that no forces are affecting the objects in space, making it possible to illustrate Newton's laws without any gravitational pull. The simplest version of the phenomenon is to add a spaceship and an asteroid. To make it a working game, the spaceship must be able to move and shoot, leading to a third object — the bullet. The game may not be that simple to start with, and requires if-statements for the collisions, but the path towards a fully working game is not that hard. However, by giving the learner a skeleton code, where the structure is defined, it may be easy to make the learner understand what is going on.

As it is a game, five elements can be added from a physics perspective: (a) multiple asteroids, (b) multiple bullets, (c) multiplayer, (d) asteroids splitting into multiple small asteroids when shot at, (e) pull from gravity from asteroids.

5.9 Conclusion

This section has given a brief introduction to eight physics phenomena that are typically introduced in introductory physics and discussed how each could be programmed.

Now, to answer [RQ1.1](#), one must consider how each of these physics phenomena could be in their simplest form, and what that simple form requires from the learner. If the simplest form requires the need for if-statements, it may be hard to understand for a beginner learner. Requiring two dimensions also sets a basis for how the problem must be modeled in order to work. [Table 5.1](#) gives a brief overview of what is required from each of the phenomena. The phenomena have also been marked as “introductory level” if they have a simple form that does not involve functions or loops of any kind. The amount of additional elements has also been noted to show how much potential each of the phenomena has when starting from the simplest form. This may also be a good indicator of which phenomena can give the most learning outcomes, as the learner can start with a simple concept and then use it as a foundation for learning new concepts.

To conclude, there were six phenomena that were the introductory level. Thus only four of them were easy (E) to start with, did not require much programming skills, and had at least two or more additional elements to add. These are marked with green in the table.

	Simple form							Evolved versions						
	Difficulty	Dimensions	If-statements	Functions	Loops / Lists	Objects needed	Introductory level	Difficulty	Dimensions	If-statements	Functions	Loops / Lists	Objects needed	Additional elements
Falling object	E	1				1	✓	M	2	✓			1	4
Pendulum	H	2				1		H	2				2	4
Block down slope	E	1				1	✓	M	2		✓	✓	1	4
Bead on a wire	M	2		✓		1		H	2		✓	✓	1	2
Ball collisions	E	1	✓			2	✓	H	2	✓	✓	✓	2+	10
Electron on uniform magnetic field	M	2				1	✓	M	3				1	1
Planet orbits	E	1				2	✓	H	3	✓	✓	✓	2+	4
Asteroids game	M	2	✓			3	✓	H	2	✓	✓	✓	3+	5

Table 5.1: Evolution of programmable physics phenomena. There are three levels of difficulty: Easy (E), Medium (M), and Hard (H). The ones marked with green are the ones that were considered as simple enough to program in the introductory physics course, having at least a few additional elements to build upon the phenomena.

Chapter 6

Requirements

By using the work from the previous chapters, a set of requirements could be defined. This chapter is going through these requirements.

Requirements are important in this project as they make sure that the artifact, that is being designed, answers the research questions (defined in [section 1.2](#)). The requirements also plays a vital role in the design of the artifact as well, such that the artifact is reproducible. More detailed, the artifact is designed in an iterative fashion having an evaluation after each iteration cycle checking if the requirements are met (Hevner, 2007, p. 4). If the requirements are not met, a new iteration takes place. Else, the artifact is performing well enough and the project can conclude, or the requirements need a refinement as they were incomplete.

The requirements have been listed in [Table 6.1](#). A discussion about each requirement is provided next. In the table, there are a few requirements that are adapted (marked with *). These were found from other studies. Their sources are given in the discussion.

ID	Requirement
From Research Question 1.1	
R11	Users should, through programming, be able to explore tasks that have non-constant acceleration
R12	Teachers should be able to create falling object tasks
R13	Teachers should be able to create block down a slope tasks
R14	Teachers should be able to create ball collision tasks
R15	Teachers should be able to create planet orbit tasks
To Answer Research Question 1.2	
R21	Each task UI should contain at least one code editor*
R22	Each task UI should contain at least one instruction*
R23	Each task UI should contain at least one output (in form of graphical or textual)*
R24	Each task UI should be able to give feedback in form of motivation or help*
R25	Each task UI should have a code editor to specify initial conditions*
R26	Each task UI should have a code editor to put code that should be updated on each delta time. This editor is also divided into variable updates and visual updates.*
R27	Each task UI should make it possible to see current variable values*
R28	Each task UI instruction should be divided into sections and subtasks*
R29	Each task UI should make it possible to add graphs*
To Answer Research Question 1.3	
R31	Each task creation UI should be able to start from scratch
R32	Each task creation UI should be able to start from existing tasks
R33	Each task creation UI should give help at each step in the creation process
R34	Each task creation UI should make sections and subtasks logical to use
R35	Each task creation UI should make it possible to give feedback in form of motivation or help
R36	Each task creation UI should make it possible to abstract code from the learners
R37	Each task creation UI should make it possible to automatically test each subtask with Python 3 code to verify its completion
R38	Each task creation UI should make it possible to test each subtask as the learners would such that the teachers can verify that the subtask works as intended
R39	Each task creation UI should make repeating tasks, like writing description and tests, less repetitive

Table 6.1: List of requirements. The adapted requirements are marked with *.

6.1 Discussion of Requirements

This section will discuss each of the requirements to clarify where they come from and their purpose. This section has also been divided into logical splits, making it easier to follow the context of each of them.

6.1.1 Research Question 1.1

The first sub research question was: “*What phenomena in physics can be programmed in the introductory physics course?*”. This requirement already has an answer in the previous chapter (see [Table 5.1](#)). However, it affects [RQ1.2](#) and [RQ1.3](#) by specifying what types of problems that should be focused on.

Requirement 11 (R11) Users should, through programming, be able to explore tasks that have non-constant acceleration

The first requirement, [R11](#), is focusing on what currently has been decided upon in the new curriculum (see [section 2.2](#)). This has to be added such that all tasks that are created can support this exploration. Just by using a programming language in general, it is possible to validate this requirement.

Requirement 12 (R12) Teachers should be able to create falling object tasks

Requirement 13 (R13) Teachers should be able to create block down a slope tasks

Requirement 14 (R14) Teachers should be able to create ball collision tasks

Requirement 15 (R15) Teachers should be able to create planet orbit tasks

The next four requirements are close to the first one; thus in these, it is important to note that the teacher should be able to create these tasks. This means that they need to get sufficient with scaffolding along the way. These requirements also set a few new implicit requirements to the artifact, like that it needs to be able to visualize certain elements, speed up time, and zoom out.

6.1.2 Research Question 1.2

The second sub research question was: “*What elements does an OLP need to teach pupils to use programming in the introductory physics course?*”. To get an answer to this question, a set of elements has been identified from the previous chapters. These elements have then been added as requirements, such that the elements can be evaluated individually. The next paragraphs discuss how these requirements were formed based on previous knowledge.

With regards to the user-interface (UI), there has been found that all programming environments that teach programming, or have programming as a

part of a task, have at least three parts: code editor, instructions, and a graphical output or textual output, depending on the task. Apart from the common parts, there are also some creative variations, like having audio that tells how to proceed while also coding in the editor (Khan Academy), or making it possible to add multiple code editors (or blocks) with instructions around them (Jupyter). Also, Codecademy and Khan Academy both give the learners motivating feedback and help based on the situation. Multiple requirements have been created from these designs:

Requirement 21 (R21) Each task UI should contain at least one code editor

Requirement 22 (R22) Each task UI should contain at least one instruction

Requirement 23 (R23) Each task UI should contain at least one output (in form of graphical or textual)

Requirement 24 (R24) Each task UI should be able to give feedback in form of motivation or help

When going more into programming environments for physics, there are a few specific parts that affect the UI. By looking at Orban et al. (2018), Caballero et al. (2012), and Tychos¹, they all have the same separation of code that is ran one time, and code that should be ran for each delta time. Tychos even has its own code editor for each of them. This is also something that makes much sense as all phenomena in physics has some initial starting point, and from that starting point, a new state can be derived based on delta time. Based on this, these requirements were added:

Requirement 25 (R25) Each task UI should have a code editor to specify initial conditions

Requirement 26 (R26) Each task UI should have a code editor to put code that should be updated on each delta time. This editor is also divided into variable updates and visual updates.

Moreover, Orban et al. (2018) and Caballero et al. (2012), p. 4, suggested that the code that should be run for each delta time should also be divided into at least two sections: (a) update of the physics variables and (b) update of the visualizations based on the physics variables. Orban et al. (2018), p. 835, also further added (c) user input as another section, thus was not sure if it should be placed before (a) and (b), between or after. But as (c) cannot be placed without some side effects, this will not be required as a section. Also, user input is not needed as a part of the tasks that should be created either; however, it may be a useful addition later research. Anyway, section (a) and (b) were incorporated into [R26](#) due to recommendations from these studies.

¹<https://www.tychos.org>

Another requirement was added due to a conversation that happened earlier that was about the use of seeing the current state of the variables. As interviewee [A](#) could verify from the conversation later, one of the reasons to use Spyder over Jupyter was because of this feature. It was not an essential feature, but it was beneficial, and therefore it was added such that it would be possible to evaluate its importance.

Requirement 27 (R27) Each task UI should make it possible to see current variable values

By looking at studies about tutorials, it is unanimous that the instructions should be divided into steps, then substeps, in a two-level hierarchical fashion. Margulieux et al. (2012), p. 72 describe the benefits of this structure, where they argue that the steps of the task become isomorphic across several tasks. E.g., how to define variables have the same patterns, just with minor changes in the variable names. This prevents cognitive overload in future learning as the learners can recognize patterns of certain steps. But, away from the benefits of a two-level structure, none of the studies have agreed upon what to call these levels. Kojouharov et al. (2004) has used “steps” and “substeps”. Zhang et al. (2009) has used “steps” and “sub-steps”. Zhang et al. (2010) has used “sub-task” and “steps”. Margulieux et al. (2012) has used “subgoals” and “steps”. As there is no clear solution to this naming, “section” and “subtask” were used. This was because it was logical to have the “task” as the whole phenomenon, with a lot of “subtasks”. Then, to divide the subtasks into groups, or labeled subgoals as Margulieux et al. (2012) suggested, there were added sections — each with their own label and instructions.

Requirement 28 (R28) Each task UI instruction should be divided into sections and subtasks

A useful element included in most PEs that teaches physics (see [section 4.3.1](#)), is graphs. This element has been used to visualize changes in physical property over time. This is especially useful when learning about non-constant acceleration as it is hard to detect changes in acceleration through an animation. As of the usefulness of the element, it has been added in [R29](#).

Requirement 29 (R29) Each task UI should make it possible to add graphs

6.1.3 Research Question 1.3

The third sub research question was: “*How can we design a user interface for creating programming tasks in the introductory physics course that supports the teachers?*”. To answer this question, these requirements were added:

The first two requirements were added such that the teachers could both create tasks and modify existing ones. This way, the design is forced to be able to do both.

Requirement 31 (R31) Each task creation UI should be able to start from scratch

Requirement 32 (R32) Each task creation UI should be able to start from existing tasks

The next requirement will test how important help is for creating a task. It may be that it is too much or that it is only needed once.

Requirement 33 (R33) Each task creation UI should give help at each step in the creation process

As discussed for requirement [R28](#), the tasks should be divided into two levels: sections and subtasks. This has to be incorporated into the design of the task creation UI such that it makes sense. Else the teachers could structure the task in a way that makes it less beneficial.

Requirement 34 (R34) Each task creation UI should make sections and subtasks logical to use

It could be beneficial to have the option to tell the learners about typical mistakes that can be made in their own task, and not rely on typical error messages only. Also, having the option to give motivation could be beneficial, too, as Codecademy and Khan Academy have done as mentioned earlier with [R24](#).

Requirement 35 (R35) Each task creation UI should make it possible to give feedback in form of motivation or help

To make the learners focus on the actual physics, it should be possible to abstract code in the code editor. This way, the learners cannot be stuck on trying to understand code that is not important for the problem. As interviewee [A](#) told, “beginners just see syntax,” noting the importance of just showing essential code.

Requirement 36 (R36) Each task creation UI should make it possible to abstract code from the learners

Automatic assessment of code is a feature that most interactive programming platforms have today. However, only a few of them, e.g., Tychos and Jupyter with nbgrader plugin, make it possible to create these as regular teachers. This feature is also essential as it is a part of an OLP as described in [Table 4.1](#). Else, the system would not have benefited from being scalable and giving feedback in real-time.

Requirement 37 (R37) Each task creation UI should make it possible to automatically test each subtask with Python 3 code to verify its completion

Having an option for the teacher to actually test the task they are currently creating, together with tests they have created, may be beneficial. This is closely related to the previous requirement as it is about getting feedback early, before a big mistake is made.

Requirement 38 (R38) Each task creation UI should make it possible to test each subtask as the learners would such that the teachers can verify that the subtask works as intended

At last, when creating tasks, it may be many repeating steps. This may be cumbersome and unnecessary. Having the option to generate parts of a description or a test that is typically written, may then be beneficial.

Requirement 39 (R39) Each task creation UI should make repeating tasks, like writing description and tests, less repetitive

Chapter 7

Design of The Initial Artifact

This chapter describes the process of designing the initial artifact with a focus on how it should help answer [RQ1.2](#) and [RQ1.3](#). The list of requirements defined in the previous chapter ([chapter 6](#)).

7.1 [RQ1.2](#): Designing The Task UI

Based on the requirements listed in [Table 6.1](#), it is possible to discuss what elements that are needed in the design of the task UI in the initial artifact. These have been discussed below:

7.1.1 Elements Included from Requirements

A code editor ([R21](#))

As required by [R21](#), a code editor needs to be included. However, as Orban et al. (2018) and Caballero et al. (2012) suggests, the editor needs to have two separate sections for defining initial conditions ([R25](#)) and code that runs every delta time ([R26](#)). To solve this, the code editor was split into two elements in the design:

Code editor for initial conditions ([R25](#))

The main purpose of this editor is to define constants and objects that will be used throughout the task. The code in this editor is intended only to be executed once per new simulation.

Code editor for code that updates each delta time ([R26](#))

This editor is an extension of the first code editor and works as an abstraction. The reason for abstracting code that runs every delta time is because there are mainly two approaches when dealing with time in Python: for-loops (see [Listing 7.1](#)) and loop functions (see [Listing 7.2](#)), where both of them requires some level of understanding of programming (see [subsection 3.4.8](#)). Also, in Python, they both require some indentation knowledge, making the code prone to more errors by the learner. By using this extra code editor, there will be no indentation errors, nor do the learners need to learn for-loops or functions.

```
1 dt = 0.1
2 t = 0
3 while t < 10:
4     t = t + dt
5     # Run loop code here
```

Listing 7.1: For-loops

```
1 def loop(t):
2     # Run loop code here
3
4 # Then the actual running of the loop is hidden from the user
```

Listing 7.2: Loop functions

Instructions (R22)

To present the content of the task to the learners, at least one instructional element is needed (R22). However, as found by Morrison et al. (2015), dividing the instruction into groups (or sections) that are then divided into subtasks (R28) was found to be beneficial when learning to solve computing problems. Two new elements were therefore included in the design:

Section instructions (R28)

Each section instruction should focus on a single goal at a time. For example: define constants, create an object, or make an object fall with gravity. As mentioned earlier, this also enables the sections to be isomorphic with other sections across different tasks.

Subtask instructions (R28)

Each section is divided into subtasks, where each subtask focuses on a simple and testable change in the code editors.

Textual output (R23)

Further, to give feedback to the learners, a textual output is included (R23). The textual output can then give different feedback elements:

Feedback from teacher (R24)

Through the textual output, teachers can give automated feedback to the learners (R24) through tests. These can be in the form of motivation or help.

Feedback from the system (R23)

However, when the learners make mistakes, like syntax errors, and if the task does not focus on that mistake, specifically, the system should give default feedback that can help the learners move forward.

Graphical output (R23)

To get a graphical representation of the physical phenomenon, R23 was included. This can help the learners understand physical concepts, how their code works, and what the values in the calculations mean. It can even make them able to detect logical errors that the system cannot detect. A graphical representation can also make the learners understand what needs to be done in a subtask if the instructions are unclear. To make all this possible, two graphical elements were added:

Graphical output from current code (R23)

This graphical output is the most important one as the learner should be able to visualize their results and make sense of their current values. Moreover, to make this graphical output less of a problem, it should be designed to scale with the elements that it contains. This means the learners should not need to draw nor place the objects on the graphical output, as those are the main issues with graphics (stated by interviewee A), but rather get a well-positioned and scaled image of the phenomenon instead. The graphics should also contain a grid showing the scale and positioning of the objects, as well as the direction of the x and the y-axis.

Graphical output from solution of the subtask (R23)

The next element that is included is a duplicate of the previous element, but instead of displaying the current state of the learner's results, it should display the solution of the subtask. As stated previously, this can help explain the instructions if they were unclear to the learner and be an alternative to the instructions.

See the current variable state (R27)

The second last requirement is the possibility to see the current state of each variable. This can help understand the flow of the programming as they then can see the visual changes together with the variable values.

Graphs (R29)

The last requirement (R29), which is closely related to R27, adds the option to see changes of a variable over time.

7.1.2 Elements Included Due to Design Choices

A few choices were made during the design phase. These were either choices that came naturally to the design, or choices that could be useful as previously seen in other similar designs.

Task title and description

These seemed to come naturally to the task as it introduces the phenomenon before the instructions.

Automatic testing

This was not mentioned for the task UI specifically, but as mentioned in the OLP description (see [Table 4.1](#)) and the requirements for [RQ1.3](#).

Preexisting code in editor

After sketching a few tasks, it was noticed that having the option to add new code to the code editor could be beneficial. This could also remove the errors done by the learners previously, cleaning up their code.

Outline

The supervisor mentioned this during a meeting. This was thought of earlier and was included in most PEs, but none of the studies mentioned its effect. Anyway, by adding an outline, it is possible to evaluate the effect of this element and get a better overview of the task and its goals.

See progress

By having the option to see the progress, may motivate the learners. But that is not tested, so it is not known to have an effect yet. Thus, by adding checkmarks to the outline on what is done, what is failed, and what is not done, it is possible to see the progress.

Underline on syntax errors

To point out the error to the learner, it is typical to underline the line. Maybe even the position at that line. How detailed the underline will depend on what is possible in the implementation.

Autocomplete in code editor

In programming, there is a lot of repetition. This may lead to too much focus on writing and not on the learning. This also makes the learner prone to writing errors. Having the option to autocomplete will fix some of these issues.

Get solution after X attempts

The task may be wrong, or the tests are too precise, meaning even if the tasks are done correctly, it is impossible to complete. This resulted in adding this element such that it is possible to complete it to the next task. It may also be that the learner cannot figure out the task. Adding this option may make it too easy to complete, but not having it could result in impossible tasks for some.

7.1.3 Summary

To summarize the elements that were added to the design, they can be seen listed in [Table 7.1](#). The IDs are referred to later in the implementation chapter and the evaluation chapters.

ID	RID	Element
E01		Task title
E02		Task description
E03	R28	Subtask title
E04	R21	Code editor
E05	R23	Graphical result output
E06	R28	Section title
E07	R28	Section level instructions
E08		Automatic tested subtasks
E09		Preexisting code in editor
E10	R23	Understandable error feedback
E11	R28	Subtask level instructions
E12		Outline
E13		See progress
E14	R26	Separate code editor for initial conditions and loop
E15	R24	Textual feedback from teacher through tests
E16		Underlines on syntax errors
E17		Autocomplete in code editor
E18	R23	Textual output
E19	R23	Graphical solution output
E20	R27	Current variables
E21	R29	Graphs
E22		Get solution after X attempts

Table 7.1: Initial elements placed in the design of the task UI in the initial artifact. RID is the ID of the requirement.

7.2 RQ1.3: Designing The Task Creation UI

The design of the task creation UI was based on the requirements found in [Table 6.1](#).

Start from scratch and existing tasks ([R31](#) and [R32](#))

The design contains a button for creating tasks, making it possible to start from scratch and with an empty UI. To modify an existing task, it is possible to press a button in the task UI that tells to modify the task.

Help at each step (R33)

To make it possible to help at each step, a help bubble is added to each of the elements. New help bubbles may be added or removed based on feedback from the evaluations.

Structure of the UI (R34)

As the task has a two-level structure with sections and subtasks, the UI follows the same structure. It is not proven that this UI will be beneficial nor less beneficial. To find out, this structure was used for the initial artifact. However, new structures could be made after suggestions from feedback.

Providing feedback to learners (R35)

The UI contains a testing field for each subtask. In this testing field, it is possible to give different feedback through unit tests. This way, the teacher can provide both help and motivation based on what the learner has done in their code.

Abstracting code from the learner (R36)

There has been added “hidden code” fields on three different levels on the UI. The first is on the task level, where it is possible to add code that the learner does not see for the whole task. This could be useful when including libraries that are used across all sections and subtasks. Next, a “hidden code” field is added on the section level. The code in this field is running in all subtasks in the section. At last, a “hidden code” field is added for each subtask. Each “hidden code” field contains a code field for both initial code, and code that runs every delta time. The codes are also run before the learner’s code such that the learner can make use of functions, variables, and objects that the teacher has made on beforehand.

Testing the learners code (R37)

To verify each subtask, at least one test needs to be made. To make this possible, a code field for adding unit tests was made. This field made it possible to test variable values and simulate the learner code X seconds into the future so that the variables could be tested at those points.

However, it is hard to test numerical results over time using numbers. To resolve this issue, a solution code was added, making it possible to test the learner’s code against the solution code, which produces the same numbers. This also made it possible to simulate the solution graphically, as added in the task UI design.

Testing the subtasks (R38)

To test the subtask that is worked on along the way, the task UI was added to the task creation UI. This was to make it possible to test the subtasks in the exact same environment as the learner would experience.

Make repeating tasks less repetitive (R39)

To complete this requirement, buttons for adding different elements into the description was added close to the input fields.

This is also a problem when creating tests. However, this problem was solved using a less static method. Using information from the solution of the task, it is possible to guess what variables and what values will be tested. Based on this information, the buttons could change the content. E.g. if the solution contained "g = 9.81", a button with "Test if g equals 9.81" could be generated.

7.3 General Design Choices

To make the evaluation of the two systems resemble a realistic environment, two choices that do not directly impact the evaluation was made:

Build both UIs into the same OLP system

Instead of building the two UIs separately, they were incorporated into a single project. To make this possible, a home page was added such that it was possible to choose to either do a task or create a task. The home page also makes it possible to choose tasks.

Make it possible to log in

To raise the threshold of creating tasks and other content, a login functionality was added as a part of the design. This is to both avoid bots and mark content that is created with user IDs so that they get an owner.

Chapter 8

Implementation of The Initial Artifact

The OLP design was described in the previous chapter, but the design does not necessarily describe the OLP that was developed and evaluated. This chapter aims to give a detailed description of how the OLP ended up looking and working as well as its limitations. A running demo of the initial OLP can be found here: <https://master-thesis-artifact-e08k1qmx7.now.sh>. If the OLP is not running, the source code for setting up the initial version can be found here: <https://github.com/niklasmh/master-thesis-artifact/tree/release/pilot-evaluation>.

8.1 Technical Choices

During the implementation phase, the selection of technologies had to be made. These are mentioned below:

8.1.1 Web Technology

One technology choice which opens up for more choices is using web technology. Web technology also makes it possible to avoid installation of software, making it sharable, which is one of the main benefits of an OLP (see [Table 4.1](#)). Further, there has been a huge effort to make web development easier over the last decade, making this a safe choice for the OLP.

8.1.2 User Interface

To develop the UI, a popular library named React¹ was used. This library was created, maintained, and used by Facebook², and is a reliable option when creating UIs for the web.

¹React JS: <https://reactjs.org>

²Facebook: <https://facebook.com>

8.1.3 Storing Data

When tasks are added to the system, they need to be stored somewhere. To make it possible to add tasks to the system dynamically, without having to add tasks through the code, Firebase³ was used. This is a product owned by Google⁴, which makes it a reliable database as they have a lot of customers.

Other reasons for using Firebase it that they provide a number of useful features for startup projects. This includes using their database for free until a reasonable transaction quota per month⁵. Firebase also consists of unstructured JSON⁶ documents, meaning the structure does not need to be decided upon before adding data to it. This also means that they are using JSON as a storage format, meaning they support nested data — which is optimal for the section and subtask structure. See how the data was stored in [Listing 8.1](#).

```
1 {
2   "title": "Falling Ball",
3   "description": "...",
4   "author": "...",
5   "hiddenCode": "import numpy as np",
6   "image": "...",
7   "sections": [
8     {
9       "title": "Define constants",
10      "description": "Like this:\n\n'' 'a = 1.23' ''",
11      "hiddenCode": "",
12      "subtasks": [
13        {
14          "title": "Set g to 9.8",
15          "description": "",
16          "hiddenCode": "",
17          "predefinedCode": "g = ...",
18          "solutionCode": "g = 9.8",
19          "testCode": "assert g == 9.8, 'Set g to 9.8'"
20        },
21        ... // More subtasks
22      ]
23    },
24    ... // More sections
25  ]
26 }
```

Listing 8.1: Format of the tasks stored in Firebase

³Firebase: <https://firebase.google.com>

⁴Google: <https://google.com>

⁵The quote at the time writing was 55000 transactions per month.

⁶JSON: <https://www.json.org>

8.1.4 Running Python 3

Usually, the learners install Python 3 on their computers. But as this artifact aims to prevent the need for installing software, which usually takes time from the teachers, a solution for playing it in the browser had to be found.

To run Python 3 in the browser, there are two main methods. The first method is to send the code to a server, make the server execute the code using Python 3, and then return it to the browser. The second method is to find a library that runs a Python 3 version in the browser that is good enough to be used. After thorough testing and research it was found that there were two problems with the first method: (a) it took time to send a request to the server and get the result, making it hard to animate, and (b) it had too many security vulnerabilities that needed to be considered, like protecting the server for hacking and DDoS attack (Zargar et al., 2013). It would be possible to overcome these problems using a sufficient amount of time and skills, but since that was too risky, the second method was better suited.

There existed a lot of libraries that used the second method, making it hard to find the best option. But after testing a few of the options, one of the libraries was definitely the best suited for this task: Pyodide⁷. This library was actually recently developed by Mozilla⁸, which aimed at making it possible to create CEs in the browser running Python 3 code for scientific purposes. This is much like what the Jupyter Notebook has achieved, with the difference that it all works on the client-side — not having to run the code on a server. They have even made it possible to import most scientific libraries, which none of the investigated options made possible.

8.1.5 Writing Python 3 Code

When it comes to writing code, there are a few tools that make that easier. One of the tools that have been quite popular in the last years is Monaco Editor⁹. This is the editor engine that is used Visual Studio Code¹⁰, which in turn is maintained and developed by Microsoft¹¹. As of the huge effort used to develop this tool to be fast and reliable, it was chosen to be used in the UI. The tool can add syntax colors to Python3 as well as it makes it possible to add functionality — like red underlines below errors and code completion¹².

8.2 Task UI

This section displays how the elements ended up looking like in the initial task UI after the implementation. To make it easier to see the context of all the elements,

⁷Pyodide: <https://github.com/iodide-project/pyodide>

⁸Mozilla: <https://www.mozilla.org/nb-NO>

⁹Monaco Editor: <https://microsoft.github.io/monaco-editor>

¹⁰Visual Studio Code: <https://code.visualstudio.com>

¹¹Microsoft: <https://www.microsoft.com>

¹²IntelliSense: <https://code.visualstudio.com/docs/editor/intellisense>

an image of the task UI is provided in [Figure 8.1](#) with a list of all the elements. An unmodified image of the task UI can be found in [Appendix B](#).

8.2.1 List of All Elements in [Figure 8.1](#)

- 1 Task title (E01)
- 2 Task description (E02)
- 3 Outline (E12)
- 4 See progress (E13): Note the red **X**-mark on the side denoting if the task has been made or not. The text is also highlighted denoting progress.
- 5 Section title (E06): The section instruction (E07) was not used in the current section in the task, but it looks similar to the subtask instruction (7).
- 6 Subtask title (E03)
- 7 Subtask instruction (E11)
- 8 Code editor for initial conditions and preexisting code in editor (E04, E09, E14): This area also contains the underlines on syntax errors element (E16) as well as autocomplete functionality (E17), but they were not visible when the screenshot was taken.
- 9 Code editor for code that runs every delta time (E04, E14)
- 10 Current variables (E20)
- 11 Textual output, system output and feedback from tests (which are created by the teacher) (E10, E15, E18)
- 12 Graphical result output (E05)
- 13 Graphical solution output (E19)
 - Get solution after X attempts (E22): This element was not visible at the snapshot but will appear as an option when the task is attempted three times.

Due to difficulties with implementing graphs, it was intentionally left out. However, the use of graphs is mentioned to the participants. If multiple participants find it essential, it will be included as an important element.

Hjem / Oppgave / DJ0apzx32MSzwZGBVtK5

Du er logget inn som: Niklas M. Hole + Lag ny oppgave Logg ut

Lag en ny oppgave ut ifra denne

1 Kloss ned skråplan med friksjon

2 Her skal vi først få en kloss til å skli ned et skråplan uten friksjon, så skal vi legge til friksjon.

3

4

5

6

7

8

9

10

11

12

13

Seksjon 1: Definere konstanter (2/2)
Seksjon 2: Lage en kloss (2/2)

Seksjon 3: Få klossen til å skli ned skråplan (1/3)

Deloppgave a) Plasser klossen på toppen av linja (0 grader)
Deloppgave b) Roter klossen med linja (45 grader)

Husk at `kloss.rot` er skrevet i radianer. For å konvertere grader til radianer kan du gjøre slik:

```
radianer = grader * pi / 180
```

Deloppgave c) Sett `kloss.x` til `t - 5` og `kloss.y` lik `5 - t`

Kode som kjører en gang

```
1 g = -9.81
2 dt = 0.02
3 kloss = Kloss(x=5, y=-5, b=1, h=1, rot=0*pi/180, color="red")
4
```

Verdier

```
dt = 0.02 # Tidssteg
t_tot = 1 # Total tid
r = -0.7071867811865476
linje = Linje(
  x1=-4.29,
  y1=5.00,
  x2=5.71,
  y2=-5.00,
  w=3.00
)
g = -9.81
kloss = Kloss
```

Din løsning

Kode som kjører hvert tidssteg, dt

```
1
```

Logg

```
print("tekst")
>
```

Fasit

Figure 8.1: The initial task UI with highlighted elements.

8.3 Task Creation UI

This section displays how the initial task creation UI was looking after the implementation. As the UI ended up being very tall, due to the structure, it is therefore separated into three images: [Figure 8.2](#), [Figure 8.5](#), and [Figure 8.6](#). Two more images are provided as of hidden collapsed functionality. The unmodified images can be found in [Appendix C](#).

8.3.1 Describing Figure 8.2

In Figure 8.2 there are seven highlighted areas. In area (1), it is possible to add both task title and task description. In area (2), there is a help bubble that displays help to the user. It can be seen in opened in Figure 8.3. In area (3), it is possible to provide hidden code. Next, area (4) is the beginning of a section. Inside the section, there is a section title (5), section description (6), and a section specific hidden code field (7). The full section description UI can be found in Figure 8.4, where it is possible to see the buttons for generating text that is seen as repetitive.

8.3.2 Describing Figure 8.5

Figure 8.5 displays seven more areas. Area (8) is marking around what is called a subtask. Inside, there are the same areas (9), (10) and (11) with title, description, and hidden code, just at a subtask level. Inside area (12) there is the starting code editor. This is the code that the learner starts with when starting on the subtask. Next, area (13) displays the solution code for the task. This makes it easier to create tests and to check if their code is actually working. At last, area (14) contains the tests for the subtask. Also, note that there are five buttons in this area. The orange ones was generated based on finding the y variable in “ $y = 0$ ” in the solution code. It also found that the variable was set to 0, suggesting to create a test checking for value 0. The blue ones were added because there is code in the second editor that surrounds changes over time. It then suggests buttons for stepping one second into the future or one step in the test. At last, the green button makes it possible to add feedback to the learner. More tips for creating tests are provided in the help bubble.

8.3.3 Describing Figure 8.6

The last figure, Figure 8.6, displays the bottom of the task creation UI. After creating a subtask, it is possible to test the task by pressing the button in area (15). The screen then moved down with the arrow and initialized the subtask with all the previous information.



Figure 8.2: The top of the initial task creation UI.

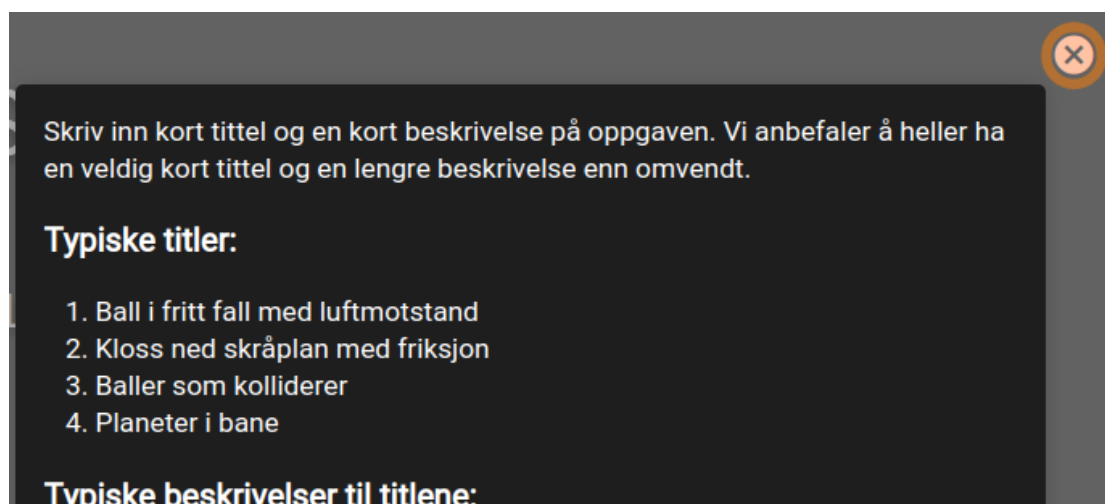


Figure 8.3: An open help bubble in the initial task creation UI.

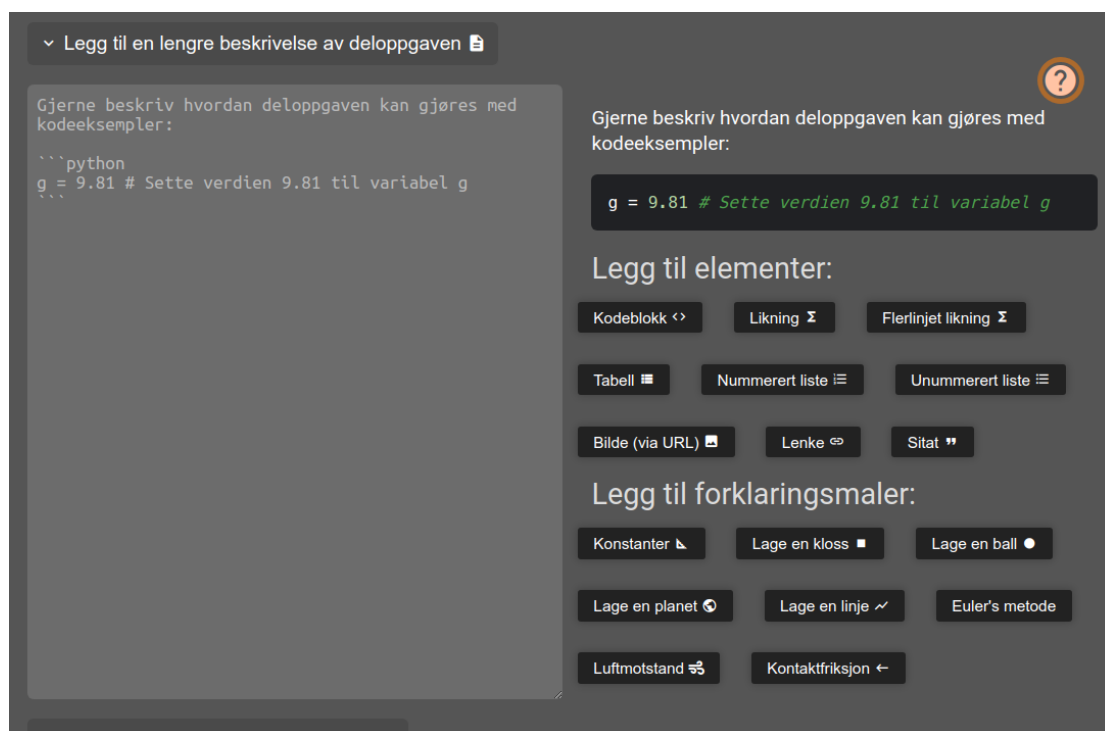


Figure 8.4: The descriptions of the initial task creation UI.

The screenshot shows a task creation interface with the following elements and callouts:

- 8**: A trash icon in the top right corner.
- 9**: A text input field for a short description of the task.
- 10**: A button to add a longer description.
- 11**: A button to add hidden code for the task.
- 12**: A section titled "Kode til eleven" containing two code editors: "Kode som kjører en gang" and "Kode som kjører hvert tidssteg, dt".
- 13**: A section titled "Løsning på deloppgaven" with a text area for the solution and a "Kopier inn fra koden til eleven" button.
- 14**: A "Tester" section containing buttons for "Sjekk om 'y' er definert", "Sjekk om 'y' er lik 0", "Simuler 1 sekund", and "Simuler et steg". Below these is a "Legg til tilbakemelding" button and a code editor with the following code:


```

1 assert y == 0, "Du må sette verdien 0 til variabel 'y'."
2
3 simulate(time=1)
4
5 # Alle tester bør skje før du sier om oppgaven ble gjennomført
6 print(f"Du klarte deloppgave {section}. {subgoal}")
7

```

At the bottom of the interface, there are two buttons: "+ Legg til ny deloppgave" and "+ Legg til ny seksjon".

Figure 8.5: The middle of the initial task creation UI.

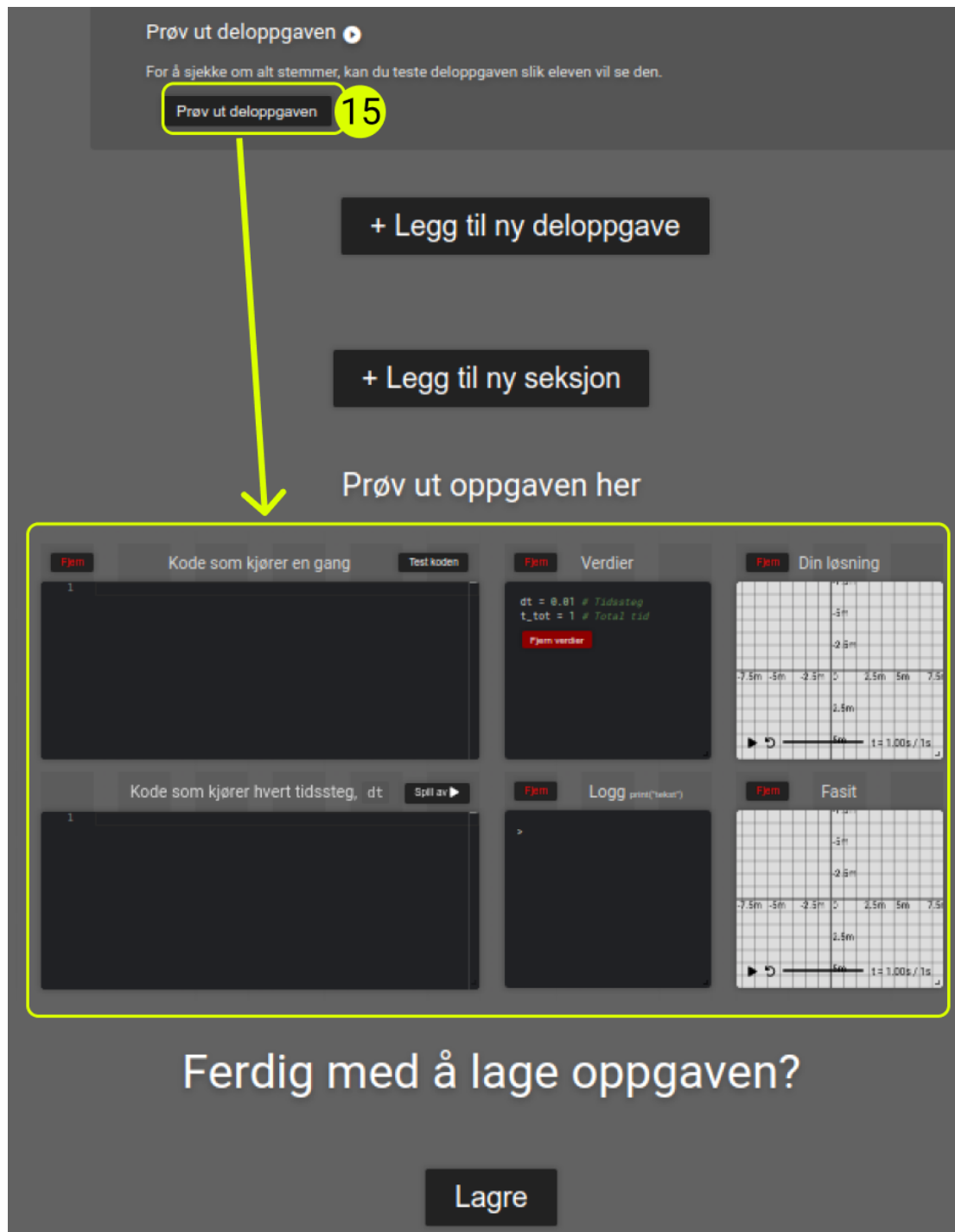


Figure 8.6: The bottom of the initial task creation UI.

8.4 Alternative Task Creation UI

To create a discussion with the participants, an alternative task creation UI was made. This new UI used an approach based on how typical developers work: using text editors only. By developing a language that could build a task using the same structure as in the first task creation UI, it was possible to create tasks using this technique. The UI also had buttons for generating typical elements, like hidden code sections, such that the user did not have to write everything and remember the syntax. The actual implementation can be seen in [Figure 8.7](#).

More specifically, the language uses Markdown¹³ with a special structure. Markdown is a lightweight markup language that is used by many professional developers to document their code. Thus, by interpreting the structure of the Markdown code in realtime, the UI generates the task elements side by side with the actual code.

The benefit of using this alternative UI is that it is transferrable to any PE. By using a Markdown editor with syntax colors on the code sections, it is possible to get close to the same experience as one does with this UI. The structure is also close to how tasks are made using CEs, which may make CE users familiar with this UI.

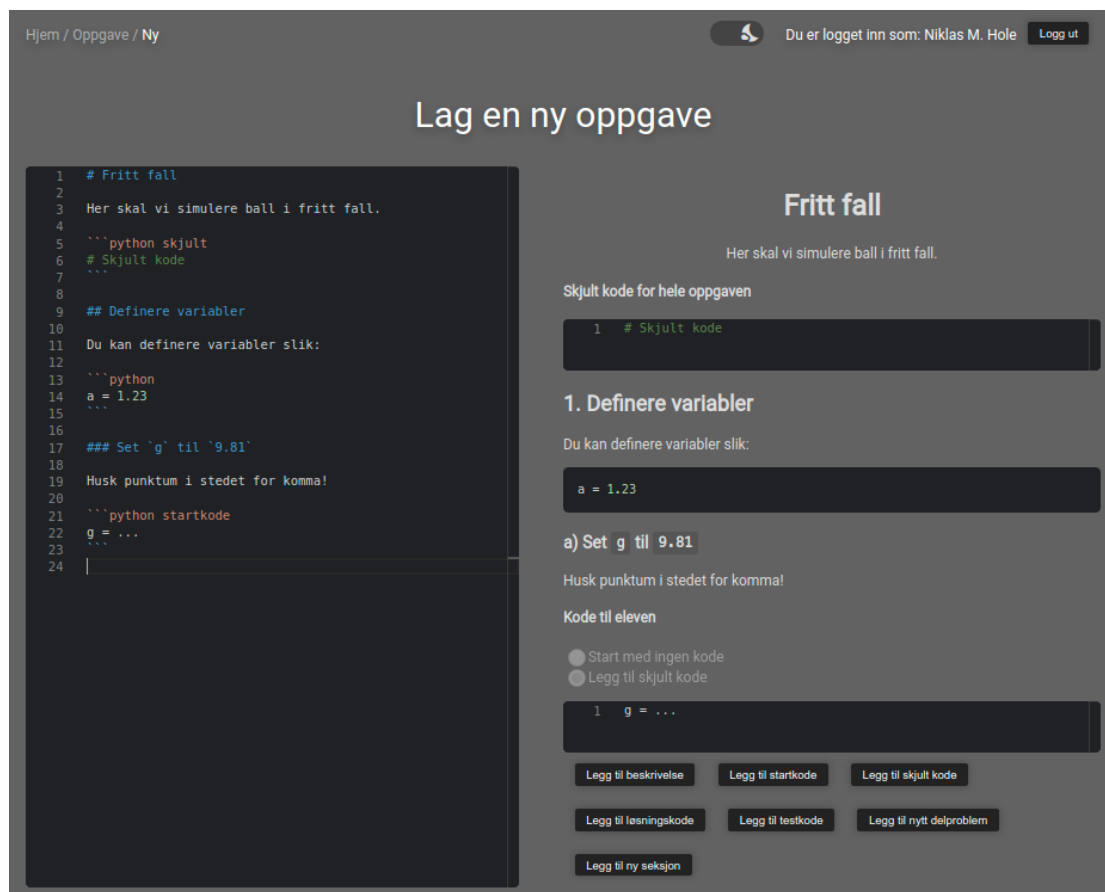


Figure 8.7: Alternative task creation UI based on Markdown syntax.

¹³Markdown: <https://en.wikipedia.org/wiki/Markdown>

8.5 Limitations

8.5.1 Missing Graphs

Due to the lack of time to implement, it was hard to add graphs to the system. Graphs were initially added to the design of the task UI, and were supposed to look like in [Figure 8.8](#).

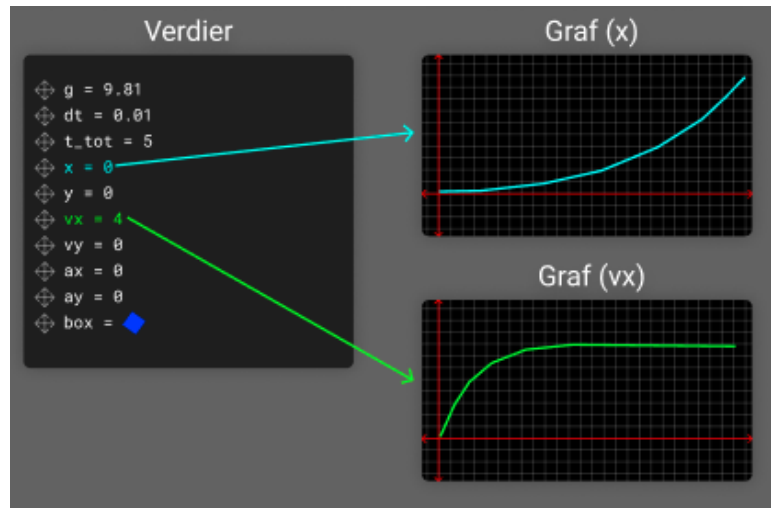


Figure 8.8: Conceptual design of graphs.

8.5.2 Slow Loading Time

As mentioned about *Pyodide* (see [subsection 8.1.4](#)), it has a slow startup time. Depending on the machine that is running the OLP, it can take from 1 second and up to 20 seconds. However, when *Pyodide* is loaded, it runs very fast and has no troubles. This loading time will hopefully be fixed in the future by the Mozilla team.

8.5.3 Incorrect Line Numbers on System Errors

When the user gets an error, there is a chance that the system does not find the correct line number. This is because it is hard to interpret the error message from Python 3. This could be fixed in the future by making the PE more systematically.

Chapter 9

First Evaluation: Pilot

To start the evaluation process of the artifact, a pilot evaluation was conducted. The main reason to use a pilot evaluation first was to test the evaluation itself and to find errors in the artifact that may remove focus from what is being evaluated. The evaluation's feedback can also be used to contribute to answering the research questions, but that is not the main goal.

9.1 Participants

In a pilot evaluation, the participants do not need to be in the target group, nor do they need to have the same qualifications. However, the closer the participants are to the target group, the more realistic feedback will be given.

To pick the participants for this evaluation, four students at NTNU were selected. They will be referred to as E, F, G, and H from now on. All of the students had some knowledge of physics and programming beforehand. Two of them, E and F, also had some experience with being teaching assistants.

9.2 Method

9.2.1 Preparation Routine

Before each interview, a new preparation routine was developed. This was to ensure that the routine was working as intended before executing it on the expert and main evaluation participants.

To start out, the plan was to first give the users a link to Zoom. Then, they were welcomed to the interview. Next, they were told that this was not being recorded as it was a pilot evaluation. After that, they would receive a link to the latest version of the artifact that was online. Here is the link that was sent to the participant over the Zoom chat: <https://master-thesis-artifact-e08k1qmx7.now.sh/>. At last, before the actual evaluation would start, they were told to think aloud and focus on what they think of the system — not the task content, as that was irrelevant.

9.2.2 Process Routine

A process routine for the evaluation itself was also developed before each evaluation.

The initial routine was to first tell that the evaluation consisted of two parts: task UI and task creation UI. They were also told that each of the parts would take around 30 minutes. However, that was just an estimation based on the previous expert interviews. Then, they were told to open their browser and paste in the link sent over Zoom.

In the first part, the task UI part, they were first told to choose one of the four tasks. When they opened a task, they were again told to think aloud and tell what they initially thought of the system. Next, they were instructed to do 2-3 subtasks. After some more experience with the system, they were again asked to tell what they thought about the system. At last, they could choose to continue to solve the task or move forward to the next part.

In the second part, the task creation UI part, they were first instructed to log in. Then, they were instructed to create a task. When the task creation UI was opened, they were again going to tell about their first impressions. They then were told to create a subtask. After creating a subtask, they were supposed to tell about what they thought about the system. In the end, they could finish what they had started on, or just save the task and leave.

At the end of the second part, an alternative version of the task creation UI was shown. As they had been exposed to creating tasks already, they were now familiar with what was needed to do so, however, exposing them to yet another way to create tasks could make them more aware of the possibilities. Then, the plan was to get a discussion resulting in what they found to be important in such a system.

9.3 First Part: Task UI

9.3.1 Results

Feedback from the first part is described in [Table 9.1](#). Each feedback has an ID with an FPF (First Pilot Feedback) prefix, a participant ID labeled as PID, and a feedback description that may also have a suggestion.

Aside from the feedback that was listed, there were also a few comments on what that worked. E, F, and G thought it was easy to see where the code should be written. E and F also noted that the subtask instructions were good, e.g., with code examples. E thought it was simple to understand for non-programmers. F told that it was good to focus on only a small part of the code, as this system does. At last, F found it motivating to see their progress in the outline.

9.3.2 Discussion

The first part of the evaluation was supposed to find an answer to [RQ1.2](#). However, as this was a pilot evaluation, the main focus was to find errors in the artifact and the evaluation method.

ID	PID	Feedback	Fixed
FPF01	E,F,G	Suggestion: Remove “run loop code” button.	✓
FPF02	E,F,H	Did multiple subtasks before running the code. Suggestion: Either run code automatically or make “run code” button more visible.	✓
FPF03	E,F,H	Suggestion: Place code editor in the middle.	✓
FPF04	E,F	Loop code button did not run the new loop code.	✓
FPF05	E,G	Using “,” for decimal does not give an understandable error.	✓
FPF06	E,G	Suggestion: Make it possible to change variables in the variable listing.	✓
FPF07	F,G	Some section titles was too long resulting in a layout bug. Suggestion: Require short titles.	✓
FPF08	F,G	Outline looks more important than it should. Also too many elements in the text. Suggestion: Move the outline out of the main content.	✓
FPF09	G,H	Suggestion: Instructions and error messages about variables should first mention the name, then the value, not value, then name.	✓
FPF10	G,H	Task typos.	✓
FPF11	E	Hard to see when section instructions are changing. Suggestion: Use different colors or highlight changes for a second.	✓
FPF12	E	Confusion auto-zoom. Suggestion: Make zoom more static.	
FPF13	E	Suggestion: Do not change code between subtasks.	
FPF14	F	Hard to see where to write the code. Suggestion: Highlight where to input.	✓
FPF15	F	Suggestion: Make it possible to navigate between tasks.	✓
FPF16	F	Suggestion: Add hints.	
FPF17	F	Suggestion: Do not duplicate info in the task title and description.	
FPF18	F	Suggestion: Collapse outline sections.	✓
FPF19	F	Suggestion: Show hidden code. Use collapsible fields in the editor.	
FPF20	H	Hard to see where to look. Suggestion: More pointers and highlighters.	✓
FPF21	H	Suggestion: Have an initial tutorial for the task UI.	

Table 9.1: Feedback from the first part of the pilot evaluation.

Errors in the artifact

Given the feedback listed in [Table 9.1](#), a few of them pointed out errors in the artifact. Two participants found a logical error with the “run loop code” button (FPF04). However, as three participants already suggested that the “run loop

code” button should have been removed (FPF01), it got removed, also resolving FPF04. Two of the participant also tried to use “,” instead of “.” for decimals. This resulted in a not understandable error for the user (FPF05). Due to the nature of Python, this type of error is very hard to detect. To resolve this error, the code editor was modified to convert all numbers with “,” into numbers with “.” automatically. As this feedback was never brought up again after this change, it would seem to work. However, this change may cause side effects as the learners may get used to writing “,” and getting an automatic correction.

Method

The first interviews were more unstructured than the last ones. This was due to getting more experience with both the preparation and process routine. Thus, overall, the routines worked as they were supposed to.

Elements

To begin answering [RQ1.2](#), [Table 9.2](#) contains all the elements that were placed in the artifact on purpose as well as elements that were suggested or not implemented on purpose. Those who did not exist in the artifact at the time were marked with “-” in the “Used by” column. Each element has either been placed in the “Should be included” section or the “Optional” section, based on whether they are important to include in an OLP used to introduce programming in an introductory physics course. If they received a complaint, the complaint was noted in the “Feedback” column. As it is important to know if the element was useful for the tasks, they were either noted with “ALL” in the “Used by” column, for being used by all, or with “SOME”, for being used by one or more participants, but not all. If it was noted with “NONE”, none of the participants used it.

For an element to be included in the “Should be included” section, the element must be used by all and be told to be beneficial by more than one participant. However, if the element was not implemented in the artifact, it must have been suggested by more than one participant.

Changes

The main change that needed to be done was to make it the focus of the task clearer, including removing elements that proved to distract the participants from the task at hand. All the changes can be seen in [Table 9.3](#).

ID	Element	Used by	Feedback
Should be included			
E01	Task title	ALL	
E02	Task description	ALL	
E03	Subtask title	ALL	
E04	Code editor	ALL	Hard to find the first time
E05	Graphical result output	ALL	
E06	Section title	ALL	Too long
E07	Section level instructions	ALL	Hard to see changes
E08	Automatic tested subtasks	ALL	
E09	Preexisting code in editor	ALL	
E10	Understandable error feedback	ALL	
Optional			
E11	Subtask level instructions	ALL	
E12	Outline	SOME	
E13	See progress	SOME	
E14	Separate code editor for initial conditions and loop	ALL	
E15	Textual feedback from teacher trough tests	NONE	Hard to spot
E16	Underlines on syntax errors	SOME	
E17	Autocomplete in code editor	SOME	
E18	Textual output	SOME	
E19	Graphical solution output	SOME	
E20	Current variables	NONE	No use cases
E21	Graphs	-	-
E22	Get solution after X attempts	ALL	
E23	Navigate between tasks	-	Wanted
E24	See hidden code	-	Wanted
E25	Change variables in variable list	-	Wanted
E26	Highlight where to look	-	Wanted
E27	Initial tutorial for task UI	-	Wanted
E28	Save progress, continue later	-	Wanted

Table 9.2: Feedback on specific elements from the pilot evaluation. The yellow ones were added from suggestion.

ID	Change
FPF01	Removed “run loop code” button.
FPF02	“Run code” button only appears when a change is done as well as not listing the next subtask titles. See Figure 9.1a .
FPF03	Placed code editor in the middle.
FPF04	Removed “run loop code” button.
FPF05	Automatically correct “,” in numbers to “.” in the code editor.
FPF06	Make it possible to click a variable to change it. Display a text that it is changeable on mouse hover.
FPF07	Shorten long section titles as well as make the titles wrap.
FPF08	Move outline to the whole left side of the screen. See Figure 9.2 .
FPF09	Change all occurrences where value comes before the variables name. Also changed the task creator UI to generate feedback with this format too. See Figure 9.3 .
FPF10	Changed tasks.
FPF11	Highlight section for a second when it changes. See Figure 9.4 .
FPF12	Not fixed as it has benefits as well. Needs a better solution.
FPF13	Not fixed as teachers may want different dynamics of their tasks. Thus, it was a reasonable suggestion.
FPF14	Pulsating highlight around code editor when the user approaches a new task. See Figure 9.1b .
FPF15	Suggestion: Make it possible to navigate between tasks.
FPF16	Not fixed.
FPF17	Changed tasks.
FPF18	Collapse outline to only show subtasks belonging to the current section. See Figure 9.2 .
FPF19	Not fixed.
FPF20	Fixed by FPF02, FPF08, FPF11, and FPF14.
FPF21	Not added. Thus, it was a reasonable suggestion to add.

Table 9.3: Changes from feedback in [Table 9.1](#).

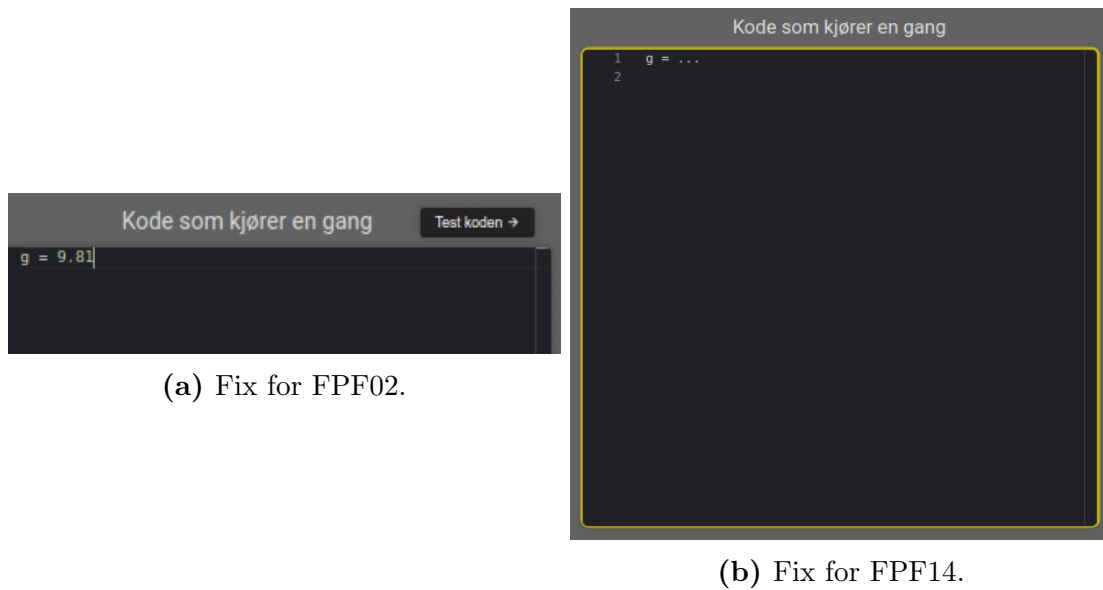


Figure 9.1: Fixes for code editor.

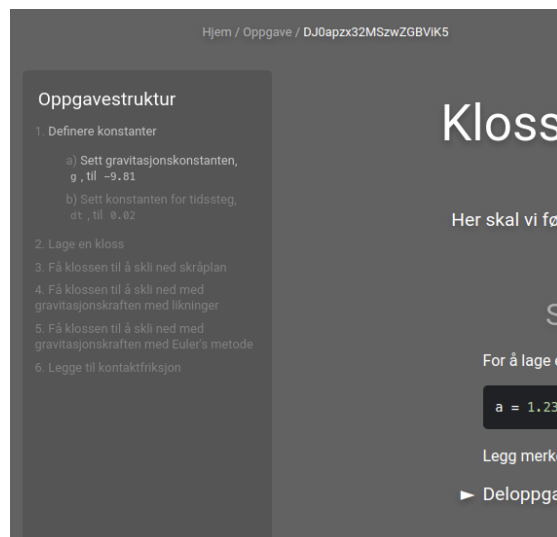


Figure 9.3: Fix for FPF09.



Figure 9.4: Fix for FPF11.

9.4 Second Part: Task Creation UI

9.4.1 Results

In the last part of the evaluation, there was a discussion about how a user interface (UI) for creating tasks could be designed. The participants were first presented with a potential example of how such UI could be, and then they had a look at another potential way of creating a task creation UI.

The first VS. the second task creator UI

In general, the participants had mixed thoughts of which UI was the best. E and G thought that the first UI they were presented to was the best, but F thought the second UI was better as it could make the users more efficient in the long run.

More specifically, the first UI made it possible to get help on all steps throughout the process, through help areas placed on each element. All participants found that having this option was essential to understand what content goes where and what their possibilities were.

On the contrary, the second UI had no help at all except for buttons that could generate elements in the task. F preferred this way of making a task as the participant had a lot of experience with coding in text editors. F also told that having the option to save it in a single file and having the option to modify the task in a human-readable format was a benefit.

Creating tests

All of the participants found that having buttons to generate tests made creating tests easier and more fun than writing them themselves. Also, having the option to compare the learner's results with their own solution after n seconds was also found useful by the participants.

Creating instructions

All of the participants agreed that having buttons for generating instructions was a useful tool. F pointed out that just having the buttons for generating elements in the instructions also helped with understanding what was possible to write. H suggested that having the option to generate a whole section, including instructions, code, and tests, would be beneficial as some sections would be repeated across many tasks.

Trying out the task

The last benefit that stood out had the option to test the task in a realistic environment while making the task. All of the participants found testing to be useful when creating a task. H also pointed out that expanding the option to test the whole task with titles, sections, and instructions could be even more useful as the teachers can then see the task in its full form.

9.4.2 Discussion

So far, even though the participants only had some level of understanding in physics, and no experience with making programming tasks in physics, it was possible to discuss a few interesting results.

Examples may be very important

Even though the task's layout was forcing the participants to follow a certain flow, the examples given in the help areas were told to be of much more help when shaping the task, according to the participants. This could indicate that examples are very important in a task creation UI.

Make testing simple may be important

From the results, the participants understood how to make tests in the first task creation UI. Since this feature stood out from many other features, this may be a key feature when designing a task creation UI.

Trying out the task is important

Being able to try out the task seemed to be important for all of the participants. Making it easy to test the task at any point may also make the process of creating tasks faster and weed out bugs in an early phase.

Changes

There were no notable errors in the task creation UI that made the discussion of the design of the UI harder. However, as H suggested that having the option to populate a whole section with a button automatically, this was added as an option. See [Figure 9.5](#).

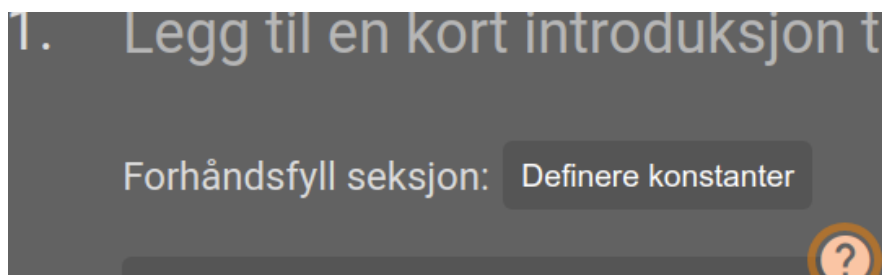


Figure 9.5: Automatically populate section button.

Chapter 10

Second Evaluation: Expert

The main reason for having an expert evaluation between the pilot and the main evaluation was to get realistic feedback from experienced physics teachers that already had experience with programming in the physics course. Feedback from this phase is then used to polish the final version of the artifact.

10.1 Participants

The participants had to be teachers with experience in programming. It was also important that they had experience with programming in a physics context.

To find the participants, contacts from previous interviews were used. Four participants wanted to participate as experts. Among these, two of the earlier interviewees were interested in joining: **A** and **B**, as well as two new participants: **I** and **K**. **A** and **I** were both physics teachers and had worked with programming in ProMod X on their own schools. This means they had a lot of experience with teaching Python programming to pupils. As ProMod X is an elective course aside with physics, most pupils who choose ProMod X also choose physics, according to **A**, which makes feedback from teachers of ProMod X very valuable. **K** was also a teacher, but did not teach programming to the pupils, however, **K** had a lot of programming experience from being an engineering student at NTNU. **B** was not currently a teacher, but had a lot of experience with programming as well as having knowledge from being a researcher researching how to use programming in more advanced physics courses.

All of the participants that were interviewed had also consented to a non-disclosure agreement from NSD¹ telling that everything that they say in the interview may be used this master thesis, but also that they are free to remove their consent or some of the information later. The consent is added to [Appendix D](#).

¹<https://nsd.no>

10.2 Method

10.2.1 Preparation Routine

The routine from the pilot evaluation was working great, so no elements were removed, changed, or added to the routine as of errors in the previous one, but as this evaluation was being recorded, sending a consent document aside with the Zoom link on a mail was added.

10.2.2 Process Routine

The process routine was the same as the one that was developed through the pilot evaluation, but there is a minor change in the process. As the artifact was being developed in between each interview, a new version of the artifact was tested on each of the participants. To be clear, these links were sent to **I**, **B**, **A** and **K** (in that order):

Participant I: <https://master-thesis-artifact-8gva387m9.vercel.app>

Participant B: <https://master-thesis-artifact-np1y158al.vercel.app>

Participant A: <https://master-thesis-artifact-5y1plth7u.vercel.app>

Participant K: <https://master-thesis-artifact-r0pd8q18m.vercel.app>

Each of the links has a different version of the artifact, thus the changes are minor and does not introduce big changes to the system.

10.3 First Part: Task UI

10.3.1 Results

Feedback from the first part of the evaluation is listed in [Table 10.1](#). Each feedback ID is prefixed by FEF (First Expert Feedback) denoting it is feedback from the first part of the expert evaluation.

Again, aside from the feedback in the table, there were some positive comments about the task UI. **I** found the outline with checkmarks to be beneficial for seeing progress and to get an overview. **B** noted that automatic code completion in the code editors were nice to have as well as having to press “run code” instead of the code running automatically once in a while. **B** also noted that it was good to get a big green goto button when completing a task, as “(...) that made me feel like I did something right.” **A** were positively surprised by a pulsating “run code” button, as it made it clear that the code needed to be run.

Observations

There were a few interesting observations to note from this evaluation. As opposed from the previous evaluation, none of the participants had any complaints on the outline (FPF08), changing section instructions (FPF11), using comma in decimal

numbers (FPF05) anymore. None of the participants continued doing subtasks before running them as well.

Observations of changes in evaluation

As some changes were added between each of the participants in this evaluation, it was possible to observe the effect of these changes on the participants that were left. This included FEF03, that was changed between **I** and **B**, where a green area was highlighted around where the user should input (see [Table 10.3](#)). None of the participants had trouble knowing where to input the code after that change. Between **B** and **A**, FEF13 was changed by flipping the coordinate system to point the y-axis upwards. This change made none of the participants react to the coordinate system as they had earlier.

10.3.2 Discussion

They did not go that much into the details of the artifact as the participants in the pilot evaluation. However, they was

Get data from various sources

Interviewee **I** and **K** wanted to get data from various sources. This option was not included intentionally as it does not simulate a phenomenon. However, the interviewees said this might be a demanding area in introductory physics as it is highly relevant, and can only be solved by computers. This also teaches modeling skills to the learners.

Do not call it Euler's method

Interviewee **B** and **A** said that using a name on Euler's method that was more descriptive could be more understandable, as well as more correct. As Euler's method is not a part of the curriculum at Norwegian USS and the general definition does not include time steps — which may lead to confusion of the method, it should not be used. Thus, adding a link to Euler's method would be fine, as then the most curious ones could explore. However, describing the method as frames in a movie, or using the motion equations, could be better for the understanding.

Elements

Besides from the previous list of elements, the updated version (see [Table 10.2](#)) contains five new elements: E29-E33. It has also moved five elements into the “Should be included” section from the “Optional”. These included E11, as subtask instructions was told to be useful by multiple participants. The outline, E12, was told to make it easier to see the progress as well as what is going to be done next. Graphs, E21, were suggested by almost all the participants and was clearly important for learning purposes. As **I** said, graphs are important for interpreting non-constant acceleration as an animation makes it difficult to tell the exact

ID	PID	Feedback	Fixed
FEF01	I,B,A,K	Issues with tasks as well as bad formulations.	
FEF02	I	Suggestion: Be consistent when telling the user to run the code. E.g. Do not use “test the code”.	✓
FEF03	I	Hard to see where to write the code. Suggestion: Highlight exactly where to input.	✓
FEF04	I,K	Suggestion: Use $X*10**Y$ instead of XeY .	✓
FEF06	I	Test failed with comparing e.g. $2*10**24$ and $2e24$.	
FEF07	I	Wrote looping code in the initial condition code editor. Suggestion: Make initial condition code not writable when it should not be used.	
FEF08	I	Hard to understand when to use objects and when to use variables. Suggestion: Use objects only.	
FEF09	I	Suggestion: Have an initial tutorial for the task UI.	
FEF10	I	Hard to see changes in non-constant acceleration. Suggestion: Add graphs.	
FEF11	I,A	Using “section” for sections and “subtasks” for subtasks was confusing. Suggestion: Use “subtask” for sections and “step” for subtasks (from I), or “task” for sections and “a,b,c...” for subtasks (from A).	
FEF12	I,K	Want to get data from various sources. Suggestion: Make it easy to import datasets.	
FEF13	B	Learners are used to having the y-axis pointing upwards. Suggestion: Flip the y-axis to point upwards.	✓
FEF14	B	Too many visible elements at the same time. Suggestion: Make variable view, text output and solution graphics optional.	
FEF15	B,A	Need to explain time steps better. Avoid naming it Euler’s method. Suggestion: Use intuitive explanations, like frames in a movie like Tycho’s does (from B). Or use the motion equations (from A).	
FEF16	B	Setting a variable to itself in a loop may be confusing as the equal sign behave different in programming and mathematics. Participant I disagrees.	
FEF17	B	Add free body diagrams.	
FEF18	A	Avoid using the absolute time variable. Suggestion: Use delta time only.	
FEF19	K	Learners may not understand “ax” and “vx”. Suggestion: Explain “ay” and “vy”.	
FEF20	K	Hard to see the flow of the variable values. Suggestion: Add state of each variable in code editor.	

Table 10.1: Feedback from the first part of the expert evaluation.

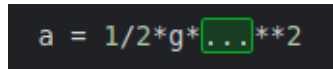
acceleration at any given time. E24 were moved as of the participants were missing seeing the hidden code behind the scenes. After testing E26, the participants that

did not have any pointer to where to write the code were suggesting a highlight of where to write the code. However, when the element was implemented, none of the participants had any complaints regarding finding where to write — suggesting that it worked. This moved E26 up. E29 was moved up as two of the participants suggested it was better to have the coordinate system to point upwards. The rest had no strong opinions regarding that. Thus, an argument for pointing it upwards, unlike what it normally used in computer graphics, is that trigonometric functions does not need to be inverted as the angles then goes in the correct mathematical direction. And, as physics are prioritizing mathematics over computer graphics, pointing the y-axis upwards was a reasonable choice.

Changes

Many changes that were suggested by the participants needed a lot of changes that required too much effort to implement. However, the conclusion is taking these into account when the research questions are answered.

Anyway, a few changes were made in the task UI. This included being consistent with using “run the code” (FEF02), not attempting any variations, like “test the code” - which is correct from a technical perspective, but not necessarily understandable for a learner. All the tasks were also modified to use $X*10**Y$ instead of XeY , which is the same, but could confuse learners that are used to the first form.



```
a = 1/2*g*...**2
```

Figure 10.1: Fix for FEF03.

ID	Element	Used	Feedback
Should be included			
E01	Task title	ALL	
E02	Task description	ALL	
E03	Subtask title	ALL	
E04	Code editor	ALL	
E05	Graphical result output	ALL	
E06	Section title	ALL	
E07	Section level instructions	ALL	
E08	Automatic tested subtasks	ALL	
E09	Preexisting code in editor	ALL	
E10	Understandable error feedback	ALL	
E11	Subtask level instructions	ALL	
E12	Outline	ALL	
E21	Graphs	-	Important for learning
E24	See hidden code	-	Highly wanted
E26	Highlight where to look	ALL	
E29	y-axis pointing upwards	ALL	Should be consistent with what is taught in physics
Optional			
E13	See progress	SOME	
E14	Separate code editor for initial conditions and loop	ALL	Should be optional
E15	Textual feedback from teacher through tests	SOME	Hard to spot
E16	Underlines on syntax errors	SOME	
E17	Autocomplete in code editor	SOME	
E18	Textual output	SOME	
E19	Graphical solution output	SOME	
E20	Current variables	SOME	
E22	Get solution after X attempts	ALL	
E23	Navigate between tasks	SOME	
E25	Change variables in variable list	NONE	
E27	Initial tutorial for task UI	-	Wanted
E28	Save progress, continue later	-	
E30	Call sections for “subtask” or “task”, and call subtasks for “steps” or “a,b,c...”	-	Wanted
E31	Get data from various datasets	-	Wanted
E32	Optionally show variables, textual output and graphical output	-	Wanted
E33	See state of each variable in code editor	-	Wanted

Table 10.2: Feedback on specific elements from the expert evaluation. The green rows contains elements that was moved into the inclusion area. The yellow are new ones added by suggestions

ID	Change
FEF01	Changed tasks.
FEF02	Changed tasks.
FEF03	Highlighted three dots or three questionmarks in a row with a green area, making them very visible. See Figure 10.1 .
FEF04	Changed tasks.
FEF05	Not fixed.
FEF06	Not fixed.
FEF07	Not fixed.
FEF08	Not fixed.
FEF09	Not fixed.
FEF10	Not fixed.
FEF11	Not fixed.
FEF12	Not fixed.
FEF13	Flipped the y-axis to point upwards.
FEF14	Not fixed.
FEF15	Not fixed.
FEF16	Not fixed.
FEF17	Not fixed.
FEF18	Not fixed.
FEF19	Not fixed.
FEF20	Not fixed.

Table 10.3: Changes from feedback in [Table 10.1](#).

10.4 Second Part: Task Creation UI

10.4.1 Results

The results in this part were slightly different from the previous result as the experts were more focused on the use cases of the artifact in actual teaching.

The first VS. the second task creation UI

All participants found the first UI to be more suited for beginners, and the second to be better for advanced users. This is the same result as with the previous result. However, they did not find any of the choices to be the best choice. This was due to the lack of how those who made programming tasks in physics were normally working. This led to the development of a third UI.

The third task creation UI

Interviewee I, which was the first expert participant, thought the first and the second task creation UI were not made to think with:

“I felt that the UI that you currently have is what you need when you already have created the task. It did not feel like a tool that I was able to think in.” (quoted by I, translated from Norwegian)

The interviewee was then eager to show how they usually created tasks for programming in physics. To summarize, they made a file, either in a computational essay (e.g., Jupyter) format or in a Python file format. Then they added skeleton code, which was not necessarily working but had fields, or lines, that had to be filled out. They also used functions to abstract complex mathematics. However, they were visible at all times, as none of the formats supported hiding them. To plot or to visualize behavior over time, an array was appended with updates in either a for or a while loop, then plotted onto a graph with time being on the x-axis. The learners were then supposed to expand the skeleton code to explore and solve tasks.

From all of the suggestions by interviewee I, a third UI was created. It was not a working prototype, but the essential UI elements were placed, including the editor, title, and the tool section on the right side. See [Figure 10.2](#). The design’s essence was to make the experience mimic the experience the teachers had when they created programming tasks in physics. In the third UI, they were able to input code in a single file like they were used to, but they could also take snapshots of the code along the way, dividing the task into several sections and subtasks. The code editor also contained areas with hidden code such that the user could hide any distracting code, as this was a suggestion by I as well (see next section).

This way of creating tasks has also been suggested by Kojouharov et al. (2004), Zhang et al. (2009) and Zhang et al. (2010), but was not considered before the feedback from interviewee I as it was targeting programming tutorials only and therefore may not be relevant.

The prototype was finished before the next evaluations and was therefore tested on participant B, A, and K. To summarize their feedback, they preferred this way of working over the two others. The UI was familiar with what they had

experience with before, but it also made intuitive sense. More specifically, [K](#) told that it was beneficial to not having to justify the code to the tool.

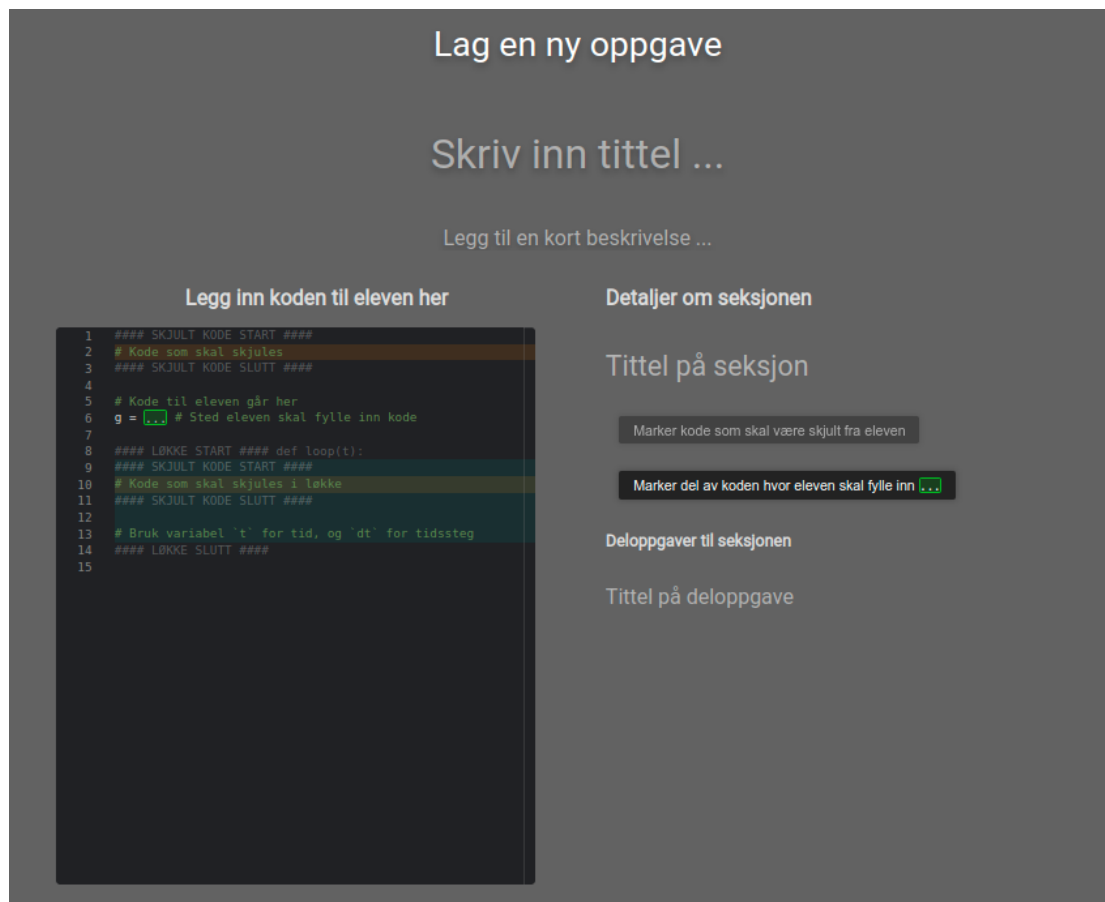


Figure 10.2: Third task creation UI.

Hiding any distracting code

Interviewee [I](#) was first to point out that hiding code was one of the main advantages with a system like this. However, it should have been possible to hide code before and after both initial condition code and loop code, as well as in the middle of them. [B](#) also found that to be a useful feature and told that tasks that were made in Trinket, which is an editor where all code is visible, had to contain: “Do not look at this” or “Do not play with this,” in code comments to avoid learners to modify code that should not have been visible. Thus [B](#) also added that Tychos had this as one of their main features when you create tasks there as a teacher.

No participants created a subtask

None of the participants in this evaluation was interested in finishing creating a task. As all of the participants in the pilot evaluation created at least one subtask, as opposed to this evaluation, this was considered as a result.

10.4.2 Discussion

Hiding code is essential

As noted by I and B, hiding code is an essential part of a good learning tool. However, it is also important that the code should be visible to the learner if they want, but that should not be the default behavior.

First UI was hard to think with

As said by I, the first UI was “hard to think with.” This may be because the UI was forcing the user to come with information in a specific order, and in portions that the user was not expecting. It may have a beneficial structure but was not good to develop a task with.

If this was found earlier on, the third UI would have been developed so that the participants could test if they were thinking better with the new UI. However, that was not possible. Anyway, the concept of the third UI was preferred over the first UI by the experts.

No participants created any subtasks

None of the experts finished creating a subtask, and two of them did not even start creating a subtask. This may be because the experts were detailed in their tasks, which the pilot group was not, leading to using too much time at the beginning of the task. This could also be because the experts found it hard to create tasks in the system as it was hard to think with, as noted previously. It may be hard to determine why they did not go further, but it may be a sign that the first UI did not meet the needs of the more experienced users.

Changes

No notable changes were made to the task creation UI except for creating the third task creation UI, described in [section 10.4.1](#).

Chapter 11

Third Evaluation: Main

This is the final evaluation where the artifact will be tested on the typical physics teacher who does not know much about programming beforehand. As this is the final evaluation, the artifact will not be changed based on feedback. Thus the feedback is being accounted for in the conclusion chapter.

11.1 Participants

The participants are physics teachers that have not worked with this topic earlier, and therefore does represent the typical user of the system that is designed in this thesis.

Earlier contacts were used to find participants. One participant was found through an earlier interview. This was interviewee [D](#). The last participant, [J](#), was found through suggestions by earlier contacts. Both [D](#) and [J](#) had some programming experience from before, but that was mostly from ProFag initiatives that prepared the teachers for the new curriculum. They were not as experienced as experts. However, they were both current physics teachers and had experience creating physics tasks, but not programming tasks.

All of the participants that were interviewed had also consented to a non-disclosure agreement from NSD¹ telling that everything that they say in the interview may be used this master thesis, but also that they are free to remove their consent or some of the information later. The consent is added to [Appendix D](#).

11.2 Methods

11.2.1 Preparation Routine

There were no change in the preparation routine.

¹<https://nsd.no>

11.2.2 Process Routine

Again, as the artifact was being developed in between each interview, a new version of the artifact was tested on each of the participants. To be clear, these links were sent to **D** and **J** (in that order):

Participant D: <https://master-thesis-artifact-c1215tvry.vercel.app>

Participant J: <https://master-thesis-artifact-5ylplth7u.vercel.app>

Each of the links has a different version of the artifact. Thus the changes are minor and do not introduce big changes to the system.

11.3 First Part: Task UI

11.3.1 Results

The final feedback from the first part of the evaluation is listed in [Table 11.1](#). Feedback that were considered as a success was not included. However, interviewee **D** found the auto-fix of comma misspelling and autocompletion to be beneficial, as well as the outline. Interviewee **D** also found having not to deal with loops to be a beneficial as well, but **J** disagreed and said that it could be essential in some cases. On the other side, **J** found the instructions to be easy to follow and the current variables to be beneficial. The interviewee also thought the use of objects and not just variables was good, as well as highlighting of where to input the code.

11.3.2 Discussion

Abstracted loop code (E14) should be optional

J argued that it could be necessary to run for-loops and while-loops when teaching specific topics. Especially when using Euler's method. **B** also mentioned that loops could be beneficial for understanding of physics, and could make it easier in the future. **A** and **I** have both introduced loops to pupils enrolled in physics class. All this points to is that loops may be needed. However, **J** and **B** also argues that it depends on the material that is taught. If loops are not directly relevant; it could be abstracted.

Thus, as almost all participants found it hard to find the loop code when it first appeared, this suggests that the abstracted loop may need a better design.

Tasks should go forward faster

Both **D** and **J** found the pace of the tasks to be a bit slow. This means that the tasks used too many steps to come to the goal. This also makes sense from the literature, as Brown and Wilson (2018) suggested that the tasks are more enjoyable if they come to the point fast. The tasks that were tried out in the OLP did not come to the point fast. They even made a little detour when attempting to describe the problem analytically.

ID	PID	Feedback
FMF01	D,J	Suggestion: Have an initial tutorial for the task UI.
FMF02	D,J	Suggestion: Graphs
FMF03	D,J	Too slow, move to the actual material faster.
FMF04	D,J	Want to get data from various sources. Suggestion: Make it easy to import datasets.
FMF05	D,J	Suggestion: Graphs.
FMF06	D,J	Fills in the initial condition code when it is loop code. Suggestion: Tell it in the instructions or block the initial condition field for writing.
FMF07	D	Coordinates looks wrong as the objects are above the numbers. Suggestion: Draw grid above the objects.
FMF08	D	Hard to test code. Suggestion: Want to automatically test code (Thus, participant B disagrees).
FMF09	D	Hard to understand "create gravitation constant and set it to 9.81". Suggestion: "define a constant g that equals 9.81"
FMF10	D	Suggestion: Show coordinates above elements in graphics.
FMF11	D	Suggestion: Give control to teachers, such that they can choose the pace while the pupils are in the classroom.
FMF12	J	Need to explain time steps better. Avoid naming it Euler's method.
FMF13	J	Add free body diagrams.
FMF14	J	Want to see the whole code. Suggestion: Make it possible to see the whole code. Maybe use collapsible fields in the code editor.
FMF15	J	Radians should not be included. There is a chance that the pupils have not learned it yet. Suggestion: Explain it better or use degrees.
FMF16	J	Too much programming in explanation. Suggestion: Explain using physics, and show examples using code. Avoid code until it is necessary.
FMF17	J	Abstracted loops are a problem when teaching Euler's method. Suggestion: Depending on what needs to be learned, abstract the main loop.

Table 11.1: Feedback from the first part of the main evaluation.

According to D, the tasks should focus on the problems that are not solvable using pen and paper. However, the task was solvable using pen and paper about 80% of the time (4 of 5 sections). To minimize the gap between the pen-and-paper-enabled tasks and the programming-only enabled tasks, the tasks should rather jump directly on the core of the problem. J also supports this argument by saying that the learners should get a skeleton code that just needs

to fill in the gaps.

What can be taken away from this argument is that the code should jump-start into the code of the problem, avoiding slow starts.

Update acceleration-first or position-first

When it comes to which order the physical variables should be run, it was different answers. Interviewee **J** argued that acceleration should come first as it is usually the first analytically. Interviewee **I** agreed, but on a different basis: the mathematically correct answer is to use position-first, as the position updates based on the previous velocity, but the corrected version of Euler's method makes it right anyway. This means that it actually may be more correct to use the acceleration-first approach, even though it is not mathematically correct.

Another argument that was for acceleration-first was in Orban et al. (2018). In their work, they argued that learners found it difficult to understand variables that continued into the next iteration of the loop. This means that if the learners use the position-first approach, they will get lost when the previous velocity is used on the position as it has not been set yet. This means that it is also the best option in a learning context. However, it was a more advanced course. The position-first approach could be better if the mathematical correctness mattered.

Thus, all three arguments point to having the acceleration-first approach, meaning that acceleration is updated first. Then the velocity is updated based on the acceleration that was just set. At last, the position is updated based on the velocity that was just updated.

11.3.3 Implications For The Design

As this was the final evaluation, the artifact was not changed. However, it is possible to describe the implications that the feedback would have on the design.

First of all, graphs (E21) would have been added such that learners could see changes in variables over time. This would, in return, make it possible to interpret non-constant acceleration that is currently not possible.

Next, an initial tutorial (E27) should have been made. This would ensure that the first time users were suited for using the task UI.

An element that was requested by many interviewees was making it possible to get data from various sources. This could also make it possible to expand the variance of tasks. However, this could cause major changes as the current UI is focused on phenomena, where the goal is to simulate based on equations, and not on modeling based on reality. This means that the UI would need more elements, like tables. Thus, this was the only focus of the UI; it could be that it had fewer elements than the current selection.

One last modification that would have been reasonable to do is to enable looping with code and to use the abstracted loop (E14). This way, if the point of the task is to learn using loops, the learners are free to do so.

ID	Element	Used	Feedback
Should be included			
E01	Task title	ALL	
E02	Task description	ALL	
E03	Subtask title	ALL	
E04	Code editor	ALL	
E05	Graphical result output	ALL	
E06	Section title	ALL	
E07	Section level instructions	ALL	
E08	Automatic tested subtasks	ALL	
E09	Preexisting code in editor	ALL	
E10	Understandable error feedback	ALL	
E11	Subtask level instructions	ALL	
E12	Outline	ALL	
E21	Graphs	-	Important for learning
E24	See hidden code	-	Highly wanted
E26	Highlight where to look	ALL	
E27	Initial tutorial for task UI	-	Wanted
E29	y-axis pointing upwards	ALL	Should be consistent with what is taught in physics
E31	Get data from various datasets	-	Highly wanted
Optional			
E13	See progress	SOME	
E14	Separate code editor for initial conditions and loop	ALL	Should be optional
E15	Textual feedback from teacher through tests	SOME	Hard to spot
E16	Underlines on syntax errors	SOME	
E17	Autocomplete in code editor	SOME	
E18	Textual output	SOME	
E19	Graphical solution output	SOME	
E20	Current variables	SOME	
E22	Get solution after X attempts	ALL	
E23	Navigate between tasks	SOME	
E25	Change variables in variable list	NONE	
E28	Save progress, continue later	-	
E30	Call sections for “subtask” or “task”, and call subtasks for “steps” or “a,b,c...”	-	
E32	Optionally show variables, textual output and graphical output	-	
E33	See state of each variable in code editor	-	

Table 11.2: Feedback on specific elements from the main evaluation.

11.4 Second Part: Task Creation UI

11.4.1 Results

As the participants has a lack of interest in creating tasks, it were not many results to get with regard to the task creation UI.

No participants created a subtask, again

None of the participants in this evaluation was interested in even starting to create a subtask this time, as opposed to the expert evaluation.

Make it possible to modify templates

Both interviewees suggested that it should be possible for task creators to create tasks from finished templates.

11.4.2 Discussion

Only a few teachers want to create programming tasks

During the evaluations of the task creation UI, it was possible to note that the teachers were uncomfortable with creating tasks, even when they were presented with the third UI – which **J** also preferred. However, interviewee **D** found it hard to understand why a teacher should create programming tasks in the first place. Further suggesting that the teachers should find tasks, as they usually do. And if they should create programming tasks, they should start from an existing template where it is easy to make modifications.

The first task creation UI is still not wanted

A hypothesis that most of the experts had was that the first task creation UI was suited for beginners. However, it was now shown that the beginners did not want to start, making the first UI less useful. As mentioned above, the teachers wanted to modify tasks. This indicates that having an easy UI is not good enough; it also must contain templates of working tasks.

Chapter 12

Conclusion

This research has attempted to support teachers with introducing programming in physics courses through an Online Learning Platform (OLP). This has made a few contributions to the scientific community: (a) A list with programmable phenomena that are well suited for introductory physics courses, (b) a list of elements an OLP needs to teach programming in the physics course, and (c) how a user interface (UI) for creating programming tasks in physics could be designed.

12.1 Answering The Research Questions

To wrap up the conversation, a concluding answer to each of the research questions has been included, starting with the main research question.

Research Question 1 (RQ1) *How can we design an OLP that supports teachers with teaching programming to pupils in the introductory physics course at Norwegian USS?*

To support teachers with teaching programming in an introductory physics course, it is important to remove any unnecessary distractions from their teaching. This includes avoiding spending time on the setup of computers in the classroom and not having to think about different operating systems or folder structures.

Next, the OLP should enable active learning in programming. It has been found that tools that engage the learners with programming tasks is more efficient than lecturing about the same content. It also is more memorable. It does not matter if the code is written, or dragged and dropped, by making the learners active, they learn programming better.

By using an OLP, there are also a few advantages over traditional teaching methods. Among these, the tasks can be shared, the teacher gets full control over their class, and it is possible to achieve automatic assessment of tasks. More benefits can be found in [Table 4.1](#). By taking advantage of these features, it is possible to support the teachers such that they can prioritize their time on their learners.

Research Question 1.1 (RQ1.1) *What phenomena in physics can be programmed in the introductory physics course?*

It is also important to teach relevant material. In the new curriculum at the Norwegian upper secondary school, they have specified how programming is going to be used in the physics courses. More specifically, they have stated that the focus should be on tasks with non-constant acceleration. After studying eight phenomena with non-constant acceleration, four of them were found to be fit to introduce in an introductory physics course. These included falling objects with air resistance, block down a slope, ball collisions, and planet orbits. These were further tested in the evaluations, and it was found that falling object with air resistance and block down a slope was good tasks to include. Ball collisions and planet orbits, on the other hand, did not get enough data as they were only attempted by one user each. Thus, they did not receive any complaints about being too difficult to solve, meaning they may be good candidates to include in an introductory physics course as well.

However, it is important to note that this is not a complete list of programming tasks to teach in the introductory physics course. This list does only include phenomena that can be simulated through time and space based on an existing model. It does not include tasks that are creating models based on patterns. E.g., programs that find a model through a dataset. In retrospect, this should have been included as it is relevant for the introductory physics course, but it could also be that the artifact would be too general if modeling from datasets would have been accounted for.

Research Question 1.2 (RQ1.2) *What elements does an OLP need to teach pupils to use programming in the introductory physics course?*

The final list of elements can be found in [Table 11.2](#). The list contains all the elements that were either found through literature or suggested by the participants. Not all elements were implemented, but by discussing potential elements and their importance, it was possible to note their importance in the artifact. Anyway, the list is divided into two parts; the ones that should be included and the ones that should be optional.

The ones that should be included were found to be important for doing the tasks that were given in [RQ1.1](#). The ones that should be optional were not seen as a necessity by the majority. However, all of the elements were good to include in the artifact, but it is important to prioritize the space on the screen to the ones that are important for introducing programming in introductory physics.

The list is not complete. More elements could be added. For example, having interactive videos could be a great tool or being able to choose multiple endings of the task based on interest. However, these were too time-consuming to implement and would drastically change the whole concept. More research in this area would be interesting to see.

Research Question 1.3 (RQ1.3) *How can we design a user interface for creating programming tasks in the introductory physics course that supports the teachers?*

Teachers are an essential resource when it comes to guiding, teaching, and motivating pupils. Research has shown that the engagement of the teacher is

essential for engaging the pupils as well. As an attempt to make teachers be more engaged in introducing programming in the physics course, a tool for creating programming tasks was designed such that the teachers would want to create programming tasks for their pupils. Hypothetically, they would then also be more engaged in introducing programming to their pupils as they would invest time in creating tasks.

However, that was not the case. The teachers who had not created programming tasks earlier, or the target group, was not interested in creating anything new. They would instead want to pick and choose from a list of already existing programming tasks in physics. They would also like to spend time going through the material, but not create it.

On the other side, the teachers who already had some experience creating programming tasks were interested in a user interface (UI) that made it easier to create programming tasks, even though they already had a working workflow. After testing the different UI concepts, the third UI, which mimics the work environment of a professional programmer, was preferred by all the participants who tested it. This was because it had a single code editor where the users could hide any distracting code, which was a problem that was not solvable with the workflow they had currently. It also had a tool section on the right side of the editor that could add instructions and specific elements to the task. To divide the task into sections and subtasks, which is a typical format to divide a task into, it was conceptually possible to take snapshots of the code while working with the task as they were used to. This concept was also proposed in JTCreator (Kojouharov et al., 2004, p. 30), which focused on creating programming tutorials as they found them to be cumbersome and time-consuming to create. JTCreator was not developed and finished. However, tools like this could be a great idea for creating programming tasks in physics or even all fields that can be combined with programming.

12.2 Future Work

This thesis did not cover all the research that should have been done to create a programming task UI specifically for physics. It also just scratched the surface of how a UI for creating programming tasks in physics could be made.

Concerning development, more concepts should be developed. Currently, there has only been developed one concept that is based on popular programming tutorial environments. Other concepts, like video tutorials, or interactive video tutorials, could be a good choice too as they may make it easier to follow while providing good examples. They may also be easier to make for the task creators. Tasks that can change goals based on interest would also be an idea. However, that would require much more effort to work.

With regard to testing, it could be interesting to test the task UI on pupils in a realistic classroom setting. Testing should also be done in groups, where each group gets a unique concept. Further, testing actual working prototypes and measuring their effect over a long time could also be useful in finding more information on this topic.

With technical advancements in computer software, it could be easier to create programming environments and programming languages specific to certain tasks, including introductory physics. This could potentially lead to discoveries, motivating research on this topic.

Bibliography

- Albashaireh, R., Ming, H., dec 2018. A survey of online learning platforms with initial investigation of situation-awareness to facilitate programming education. In: Proceedings - 2018 International Conference on Computational Science and Computational Intelligence, CSCI 2018. Institute of Electrical and Electronics Engineers Inc., pp. 631–637.
- Allain, R., 2016. Modeling a Pendulum’s Swing Is Way Harder Than You Think — WIRED.
URL <https://www.wired.com/2016/10/modeling-pendulum-harder-think/>
- Basili, V. R., Carver, J. C., Cruzes, D., Hochstein, L., Hollingsworth, J. K., Shull, F., Zelkowitz, M. V., 2008. Understanding The High Performance Computing Community: A Software Engineer’s Perspective. Tech. rep.
URL <http://www.top500.org>
- Basu, S., Mcelhaney, K. W., Harris, C. J., 2018. A Principled Approach to Designing Assessments That Integrate Science and Computational Thinking. Tech. rep.
- Bensky, T. J., Moelter, M. J., mar 2013. Computational problems in introductory physics: Lessons from a bead on a wire. *American Journal of Physics* 81 (3), 165–172.
- Bernard, S., 2010. Science Shows Making Lessons Relevant Really Matters — Edutopia.
URL <https://www.edutopia.org/neuroscience-brain-based-learning-relevance-improves-engagement>
- Blank, D., Bourgin, D., Brown, A., Bussonnier, M., Frederic, J., Granger, B., Griffiths, T. L., Hamrick, J., Kelley, K., Pacer, M., Page, L., Pérez, F., Ragan-Kelley, B., Suchow, J. W., Willing, C., 2019. nbgrader: A Tool for Creating and Grading Assignments in the Jupyter Notebook Software • Review • Repository • Archive. *The Journal of Open Source Education*.
URL <https://doi.org/10.21105/jose.00032>
- Bouvier, D., Lovellette, E., Matta, J., Alshaigy, B., Becker, B. A., Craig, M., Jackova, J., McCartney, R., Sanders, K., Zarb, M., jul 2016. Novice programmers & the problem description effect. In: Proceedings of the 2016 ITiCSE Working Group Reports, ITiCSE 2016. Association for Computing Machinery, Inc, New

-
- York, New York, USA, pp. 103–118.
URL <http://dl.acm.org/citation.cfm?doid=3024906.3024912>
- Brown, N. C., Wilson, G., apr 2018. Ten quick tips for teaching programming. *PLoS Computational Biology* 14 (4).
- Caballero, M. D., Kohlmyer, M. A., Schatz, M. F., 2012. Implementing and assessing computational modeling in introductory mechanics.
- Cakir, M., 2008. Constructivist Approaches to Learning in Science and Their Implications for Science Pedagogy: A Literature Review. Tech. rep.
URL <http://www.ijese.com/>
- Cambridge Assessment International Education, 2018. Getting started with Active Learning.
URL <https://www.cambridge-community.org.uk/professional-development/gswal/index.html>
- Carver, J. C., Kendall, R. P., Squires, S. E., Post, D. E., 2007. Software development environments for scientific and engineering software: A series of case studies. In: *Proceedings - International Conference on Software Engineering*. pp. 550–559.
- Claypool, H. M., Mackie, D. M., Garcia-Marques, T., McIntosh, A., Udall, A., 2004. THE EFFECTS OF PERSONAL RELEVANCE AND REPETITION ON PERSUASIVE PROCESSING. Tech. Rep. 3.
- de Jong, T., aug 2010. Cognitive load theory, educational research, and instructional design: Some food for thought. *Instructional Science* 38 (2), 105–134.
- DiSessa, A. A., Abelson, H., sep 1986. Boxer: A reconstructible computational medium. *Communications of the ACM* 29 (9), 859–868.
- EduTech Wiki, 2020. Microworld.
URL <http://edutechwiki.unige.ch/en/Microworld>
- Flannery, W., oct 2019. The Coming Revolution in Physics Education. *The Physics Teacher* 57 (7), 493–497.
URL <http://aapt.scitation.org/doi/10.1119/1.5126834>
- Freeman, S., Eddy, S. L., McDonough, M., Smithb, M. K., Okoroafor, N., Jordt, H., Wenderoth, M. P., 2014. Active learning increases student performance in science, engineering, and mathematics 10 (23).
- Guzdial, M., 2018. Constructivism vs. Constructivism vs. Constructionism — Computing Education Research Blog.
URL <https://computinged.wordpress.com/2018/03/19/constructivism-vs-constructivism-vs-constructionism/>

-
- Guzdial, M., 2019. Why I say task-specific programming languages instead of domain-specific programming languages.
URL <https://computinged.wordpress.com/2019/05/27/why-i-say-task-specific-programming-languages-instead-of-domain-specific-programming-languages/>
- Guzdial, M., 2020a. Computing Education Lessons Learned from the 2010's: What I Got Wrong — Computing Education Research Blog.
URL <https://computinged.wordpress.com/2020/01/13/computing-education-lessons-learned-from-the-2010s-what-i-got-wrong/>
- Guzdial, M., 2020b. Teaching Computer Science to Reach A Broader Audience: Part 1 Mark Guzdial. Tech. rep.
- Guzdial, M., McCracken, W. M., Elliott, A., 1997. Task specific programming languages as a first programming language. In: Proceedings - Frontiers in Education Conference. Vol. 3. IEEE, pp. 1359–1360.
- Guzdial, M., Naimipour, B., nov 2019. Task-specific programming languages for promoting computing integration: A precalculus example. In: ACM International Conference Proceeding Series. Association for Computing Machinery, New York, New York, USA, pp. 1–5.
URL <http://dl.acm.org/citation.cfm?doid=3364510.3364532>
- Hamrick, J. B., 2016. Creating and Grading IPython/Jupyter Notebook Assignments with NbGrader. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE '16. Association for Computing Machinery (ACM), New York, New York, USA, pp. 242–242.
URL <http://dl.acm.org/citation.cfm?doid=2839509.2850507>
- Hannay, J. E., MacLeod, C., Singer, J., Langtangen, H. P., Pfahl, D., Wilson, G., may 2009. How do scientists develop and use scientific software? In: Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, SECSE 2009. IEEE, pp. 1–8.
URL <http://ieeexplore.ieee.org/document/5069155/>
- Hevner, A. R., 2007. A Three Cycle View of Design Science Research. Tech. Rep. 2.
- Hevner, A. R., March, S. T., Park, J., Ram, S., 2004. Design Science in Information Systems Research. Management Information Systems Quarterly 28, 75–105.
URL https://www.researchgate.net/publication/201168946_Design_Science_in_Information_Systems_Research
- Hutchins, N. M., Biswas, G., Maróti, M., Lédeczi, Á., Grover, S., Wolf, R., Blair, K. P., Chin, D., Conlin, L., Basu, S., McElhaney, K., 2019. C2STEM: a System for Synergistic Learning of Physics and Computational Thinking. Journal of Science Education and Technology.
-

-
- Kitagawa, M., Fishwick, P., Kesden, M., Urquhart, M., Guadagno, R., Jin, R., Tran, N., Omogbehin, E., Prakash, A., Awaraddi, P., Hale, B., Suura, K., Raj, A., Stanfield, J., Vo, H., Raj, A., 2019. Scaffolded Training Environment for Physics Programming (STEPP): Modeling High School Physics using Concept Maps and State Machines.
URL <https://doi.org/10.1145/3316480.3325513>
- Kofod-Petersen, A., 2015. How to do a Structured Literature Review in Computer Science.
URL https://www.researchgate.net/publication/265158913_How_to_do_a_Structured_Literature_Review_in_computer_science
- Kojouharov, C., Solodovnik, A., Naumovich, G., 2004. JTutor: An Eclipse Plug-in Suite for Creation and Replay of Code-based Tutorials *. Tech. rep.
URL <http://cis.poly.edu/gnaumovi/jtutor>
- Kosar, T., Mernik, M., Carver, J. C., jun 2012. Program comprehension of domain-specific and general-purpose languages: Comparison using a family of experiments. *Empirical Software Engineering* 17 (3), 276–304.
- Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E., nov 2010. The scratch programming language and environment. *ACM Transactions on Computing Education* 10 (4), 1–15.
URL <http://portal.acm.org/citation.cfm?doid=1868358.1868363>
- Marciuc, D., Miron, C., 2017. Developing students' creativity by Physics lessons. The 12 th International Conference on Virtual Learning VIRTUAL LEARNING-VIRTUAL REALITY Phase II-Period 2010-2020: e-Skills for the 21st Century Phase III-Period 2020-2030: Intelligence Learning-Knowledge Society and Learning Culture The ICV and CNIV proj, 470–477.
URL https://www.researchgate.net/profile/Snejana_Dineva/publication/327557908_The_Benefits_of_Combining_Social_Media_and_e-learning_for_Training_Improving_in_FTT_Yambol/links/5b966f34a6fdccfd543a4e6d/The-Benefits-of-Combining-Social-Media-and-e-learning-for-T
- Marciuc, D., Miron, C., Barna, E. S., 2016. Using GeoGebra and Vpython software for teaching motion in a uniform gravitational field. Tech. Rep. 4.
URL <http://www.infim.ro/rrp/>.
- Margulieux, L., Guzdial, M., Catrambone, R., 2012. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In: ICER'12 - Proceedings of the 9th Annual International Conference on International Computing Education Research. ACM Press, New York, New York, USA, pp. 71–78.
URL <http://dl.acm.org/citation.cfm?doid=2361276.2361291>
- Martin, R. F., 2016. Undergraduate computational physics education: uneven history and promising future.

-
- Martin, R. F., mar 2017. Undergraduate Computational Physics Education: Uneven History and Promising Future. *Computing in Science and Engineering* 19 (2), 70–78.
- Morrison, B. B., Margulieux, L. E., Ericson, B., Guzdial, M., feb 2016. Subgoals help students solve Parsons Problems. In: *SIGCSE 2016 - Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. Association for Computing Machinery, Inc, New York, New York, USA, pp. 42–47.
URL <http://dl.acm.org/citation.cfm?doid=2839509.2844617>
- Morrison, B. B., Margulieux, L. E., Guzdial, M., jul 2015. Subgoals, context, and worked examples in learning computing problem solving. In: *ICER 2015 - Proceedings of the 2015 ACM Conference on International Computing Education Research*. Association for Computing Machinery, Inc, New York, New York, USA, pp. 21–30.
URL <http://dl.acm.org/citation.cfm?doid=2787622.2787733>
- On Purpose Associates, 2011. *Constructivism — Philosophy of Learning — Funderstanding: Education, Curriculum and Learning Resources*.
URL <https://www.funderstanding.com/theory/constructivism/>
- Orban, C. M., 2017. A novel approach for using programming exercises in electromagnetism coursework. Tech. rep.
- Orban, C. M., Smith, J. R. H., Brecht, 2017. A Game-Centered, Interactive Approach for Using Programming Exercises in Introductory Physics. Tech. rep.
URL <http://compadre.org/PICUP>
- Orban, C. M., Teeling-Smith, R. M., Smith, J. R. H., Porter, C. D., nov 2018. A hybrid approach for using programming exercises in introductory physics. *American Journal of Physics* 86 (11), 831–838.
- Papert, S., 1980. *Mindstorms: children, computers, and powerful ideas*. Basic Books, Inc. Division of HarperCollins 10 E. 53rd St. New York, NY United States.
URL <https://dl.acm.org/doi/book/10.5555/1095592>
- Resnick, M., 2012. Reviving Papert’s dream — MIT Media Lab. *Educational Technology*, Vol. 52.
URL <https://www.media.mit.edu/publications/reviving-paperts-dream/>
- Rich, K. M., Strickland, C., Andrew Binkowski, T., Moran, C., Franklin, D., mar 2018. John henry AWARD k–8 learning trajectories derived from research literature: Sequence, repetition, conditionals. *ACM Inroads* 9 (1), 46–55.
URL <http://dl.acm.org/citation.cfm?doid=3105726.3106166>
- Segal, J., oct 2005. When software engineers met research scientists: A case study. In: *Empirical Software Engineering*. Vol. 10. pp. 517–536.
- Simon, H. A., 1996. *The Sciences of the Artificial* Third edition. Tech. rep.

-
- Steven P. Reiss, 1994. The Field Programming Environment: A Friendly Integrated Environment for ... - Steven P. Reiss - Google Bøker.
URL https://books.google.no/books?hl=no&lr=&id=xg-e-YR90o4C&oi=fnd&pg=PP11&dq=%22programming+environment%22+tools&ots=d6H8SjgwZt&sig=Zd73yGGYEKIT1-RTm6DX8pU4TaU&redir_esc=y#v=onepage&q=%22programmingenvironment%22tools&f=false
- Taub, R., Armoni, M., Bagno, E., Ben-Ari, M., 2015. The effect of computer science on physics learning in a computational science environment. *Computers and Education*.
- UDIR, 2019a. Algoritmisk tenkning.
URL <https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/>
- UDIR, 2019b. Innspillsrunde 1 Fysikk.
URL <https://hoering-publisering.udir.no/793/uttalelser>
- UDIR, 2019c. Læreplan i fysikk (Utkast).
URL <https://hoering.udir.no/Hoering/v2/793>
- UDIR, 2020a. Innspillsrunde 2 Fysikk.
URL <https://hoering-publisering.udir.no/960/uttalelser>
- UDIR, 2020b. Læreplan i fysikk (Utkast).
URL <https://hoering.udir.no/Hoering/v2/960>
- UDIR, 2020c. Skisser til nye læreplaner på studieforbredende – Vg2 og Vg3.
URL <https://www.udir.no/laring-og-trivsel/lareplanverket/fagfornyelsen/gi-innspill-pa-skisser-til-nye-lareplaner-pa-studieforbredende--fag-elevene-velger-pa-vg2-og-vg32/>
- UDIR, 2020d. Slik ble læreplanene utviklet – fag i grunnskolen og gjennomgående fag i vgo.
URL <https://www.udir.no/laring-og-trivsel/lareplanverket/fagfornyelsen/slik-ble-lareplanene-utviklet/>
- Wing, J. M., 2012. Computational Thinking. Tech. rep.
URL https://www.microsoft.com/en-us/research/wp-content/uploads/2012/08/Jeannette_Wing.pdf
- Zargar, S. T., Joshi, J., Tipper, D., 2013. A survey of defense mechanisms against distributed denial of service (DDOS) flooding attacks. *IEEE Communications Surveys and Tutorials* 15 (4), 2046–2069.
- Zhang, N., Huang, G., Zhang, Y., Jiang, N., Mei, H., 2010. Towards Automated Synthesis of Executable Eclipse Tutorials. Tech. rep.
- Zhang, Y., Huang, G., Zhang, N., Mei, H., 2009. Smarttutor: Creating IDE-based interactive tutorials via editable replay. In: *Proceedings - International Conference on Software Engineering*. pp. 559–562.

Appendix

Appendix A

NDA for Expert Interviews

Vil du delta i forskningsprosjektet

”Introduce computational thinking in introductory physics courses: A custom python environment”?

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å introdusere algoritmisk tenking i fysikk på videregående. I dette skrivet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål

Algoritmisk tenking har fått mer fokus i den Norske skole og er muligens på vei inn i en rekke realfag, blant annet matematikk og fysikk. Dette gjøres fordi skolen skal være fremtidsrettet og det er mange jobber i dag som drives av denne tankegangen. Formålet med dette forskningsprosjektet er å undersøke hvordan den Norske skole kan innføre denne tankegangen i fysikkfaget på videregående uten at det går utover læringen av pensum.

Med dette intervjuet har vi som mål å få svar på hvordan dagens skole fungerer og hvordan man kan implementere denne tankegangen på en god måte. Samtidig vil vi også høre litt om dine tanker og ideer som du eventuelt har tenkt eller kommer på i farten. Svarene vi får fra intervjuet brukes så videre i forskningsprosjektet.

Hvem er ansvarlig for forskningsprosjektet?

Niklas Molnes Hole er ansvarlig for prosjektet.

Hvorfor får du spørsmål om å delta?

Dette er et eksperintervju hvor vi vil hente ut meninger fra forskere og lærere med en bakgrunn i fysikk.

Hva innebærer det for deg å delta?

Hvis du velger å delta i prosjektet, innebærer det at du deltar på et intervju over Skype eller ansikt til ansikt, om mulig. Dette intervjuet kan ta opp til ca. 45 minutter. Intervjuet inneholder blant annet spørsmål om din faglige bakgrunn, din erfaring med datamaskiner i klasserommet og hva slags tanker/ideer du eventuelt har om algoritmisk tenking i fysikkfaget. Dine svar på intervjuet blir tatt opp med lydopptaker og senere skrevet ned. Lydopptaket blir deretter slettet fra enheten.

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 fjernet. 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.

- Den eneste som har tilgang til lydopptaket er studenten som utfører intervjuet.
- Lydopptaket blir slettet permanent etter det har blitt skrevet ned. Svarene du har gitt blir videre assosiert med en ID i masteroppgaven og vil ikke på noen måte kunne peke tilbake til deg.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?

Prosjektet skal etter planen avsluttes 3. august 2020. Innen da skal alt av opplysninger om deg tilknyttet lydopptaket være borte.

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 Norges teknisk-naturvitenskapelige universitet (NTNU) 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:

- Student: Niklas Molnes Hole ([REDACTED])
- Veileder: Monica Divitini ([REDACTED])
- NSD – Norsk senter for forskningsdata AS, på epost ([REDACTED]) eller telefon: [REDACTED].

Med vennlig hilsen

Prosjektansvarlig
(Forsker/veileder)

Appendix B

Original Task UI Screenshot

Hjem / Oppgave / DJ0apzx32MSzwZGBVIK5 Du er logget inn som: Niklas M. Hole [+ Lag ny oppgave](#) [Logg ut](#)

Lag en ny oppgave ut ifra denne

Kloss ned skråplan med friksjon

Her skal vi først få en kloss til å skli ned et skråplan uten friksjon, så skal vi legge til friksjon.

-9.81

b) Sett konstanten for tidssteg, dt, til 0.02

2. Lage en kloss

a) Lag klossen, kloss, i origo (0,0)

b) Sett fargen på kloss til red

3. Få klossen til å skli ned skråplan

a) Plasser klossen på toppen av linja (-5, 5)

b) Roter klossen med linja (45 grader)

c) Sett kloss.x til t - 5 og kloss.y lik 5 - t

4. Få klossen til å skli ned med

✓ Seksjon 1: Definere konstanter (2/2)
✓ Seksjon 2: Lage en kloss (2/2)
Seksjon 3: Få klossen til å skli ned skråplan (1/3)

✗ Deloppgave a) Plasser klossen på toppen av linja (-5, 5)

▶ ✗ Deloppgave b) Roter klossen med linja (45 grader)

Husk at kloss.rot er skrevet i radianer. For å konvertere grader til radianer kan du gjøre slik:


```
radianer = grader * pi / 180
```

Deloppgave c) Sett kloss.x til t - 5 og kloss.y lik 5 - t

Kode som kjører en gang Test koden

```
1 g = -9.81
2 dt = 0.02
3 kloss = Kloss(x=5, y=-5, b=1, h=1, rot=0*pi/180,
4 color="red")
```

Verdier

```
dt = 0.02 # Tidssteg
t_tot = 1 # Total tid
r = -0.7071067811865476
linje = Linje(
  x1=-4.29,
  y1=-5.00,
  x2=5.71,
  y2=-5.00,
  w=3.00
)
g = -9.81
k1000 = k1000
```

Din løsning

Kode som kjører hvert tidssteg, dt Spill av ▶

```
1
```

Logg print("tekst")

```
Du må skrive 45 inn i 'rot'
parametere. Husk å legge
til *pi/180 for å konvertere
til radianer.
```

Fasit

Figure B.1: The task UI with no edits.

Appendix C

Original Task Creation UI Screenshots

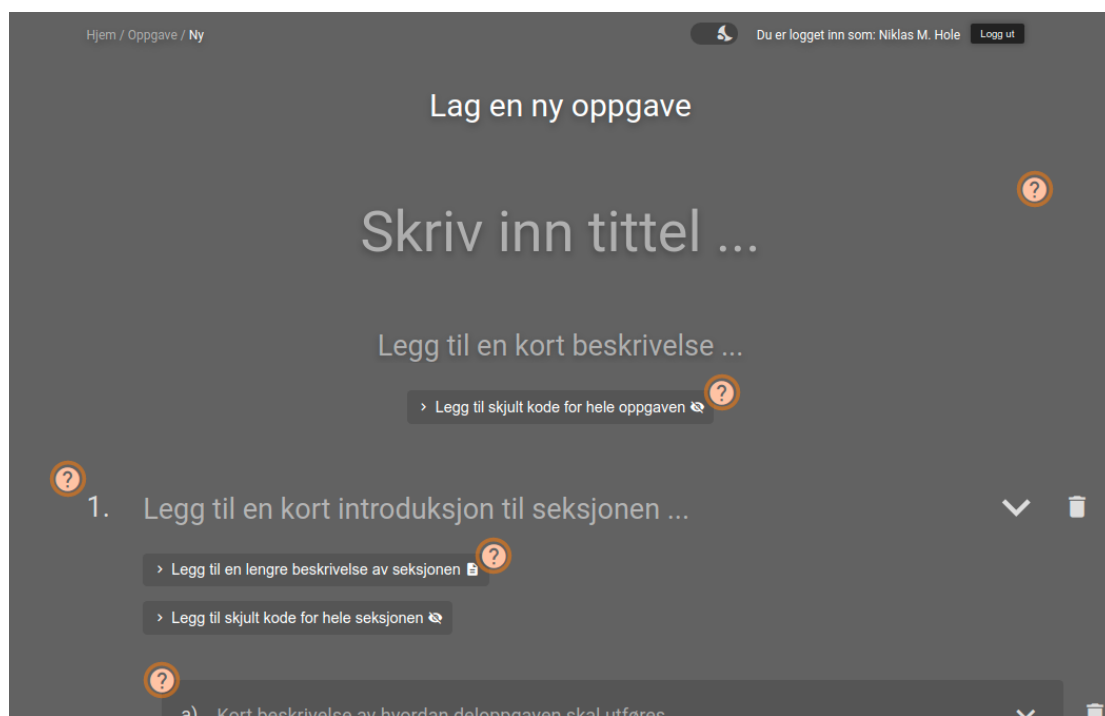


Figure C.1: The task creation UI with no edits. (Top)

a) Kort beskrivelse av hvordan deloppgaven skal utføres ...

> Legg til en lengre beskrivelse av deloppgaven

> Legg til skjult kode for deloppgaven

Kode til eleven <>

Kode som kjører en gang

```
1
```

Kode som kjører hvert tidssteg, dt

```
1
```

Løsning på deloppgaven

Eleven kan velge å se løsningen etter de har forsøkt 3 ganger. Prøv å gjør løsningen så lesbar som mulig.

Kopier inn fra koden til eleven

Kode som kjører en gang

```
1 y = 0
```

Kode som kjører hvert tidssteg, dt

```
1 y = y + 1
```

Tester

Disse testene skal sjekke om svaret er riktig. Her kan du også gi en tilpasset tilbakemelding til eleven om hva som er feil.

Sjekk om 'y' er definert ✓ Sjekk om 'y' er lik 0 Simuler 1 sekund Simuler et steg

Legg til tilbakemelding

```
1 assert y == 0, "Du må sette verdien 0 til variabel 'y'."
2
3 simulate(time=1)
4
5 # Alle tester bør skje før du sier om oppgaven ble gjennomført
6 print(f"Du klarte deloppgave {section}. {subgoal}")
7
```

Prøv ut deloppgaven

For å sjekke om alt stemmer, kan du teste deloppgaven slik eleven vil se den.

Prøv ut deloppgaven

+ Legg til ny deloppgave

+ Legg til ny seksjon

Figure C.2: The task creation UI with no edits. (Middle)

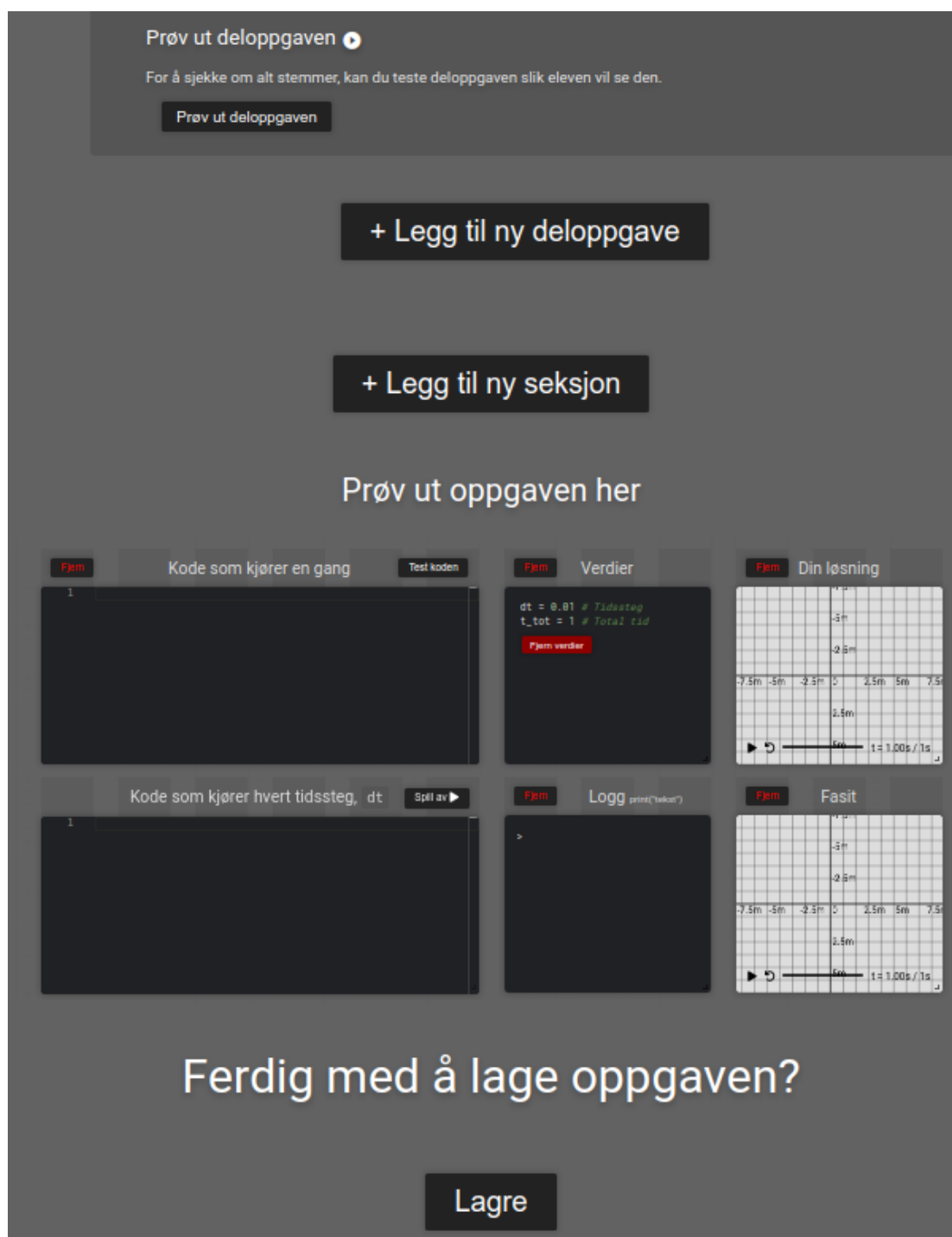


Figure C.3: The task creation UI with no edits. (Bottom)

Appendix D

NDA for Evaluation Interviews

Vil du delta i forskningsprosjektet

”Introduce programming in introductory physics courses through interactive learning platform”?

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å introdusere programmering i fysikk på videregående. I dette skrevet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål

Programmering har fått mer fokus i den Norske skole og er muligens på vei inn i en rekke realfag, blant annet matematikk og fysikk. Dette gjøres fordi skolen skal være fremtidsrettet og det er mange jobber i dag som drives av denne måten å jobbe på. Formålet med dette forskningsprosjektet er å undersøke hvordan den Norske skole kan innføre denne måten å jobbe på i fysikkfaget på videregående uten at det går utover læringen av pensum.

Under dette forskningsprosjektet har det blitt utviklet en prototype av en læringsplattform på internett som har som hensikt å hjelpe lærere med å både lage programmeringsoppgaver i fysikk samt gjøre oppgavene bedre for elevene. Med dette intervjuet har vi som mål å få svar på hvordan dette systemet fungerer for deg, og om det passer med dine forventninger av et slikt system.

Hvem er ansvarlig for forskningsprosjektet?

Niklas Molnes Hole er ansvarlig for prosjektet.

Hvorfor får du spørsmål om å delta?

Vi undersøker bruksmønstre fra fysikklærere, så om du er kommende, nåværende eller tidligere fysikklærer, er du kvalifisert.

Hva innebærer det for deg å delta?

Hvis du velger å delta i prosjektet, innebærer det at du deltar på et intervju over Zoom. Dette intervjuet kan ta opp til ca. 45 minutter. Under intervjuet vil du bli spurt om litt bakgrunn for å vite hva du kan om temaet fra før, resten handler om dine meninger om systemet. Dine svar på intervjuet blir tatt opp med lydopptaker og senere skrevet ned. Lydopptaket blir deretter slettet fra enheten.

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 fjernet. 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 skrevet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket.

- Den eneste som har tilgang til lydopptaket er studenten som utfører intervjuet.
- Lydopptaket blir slettet permanent etter det har blitt skrevet ned. Svarene du har gitt blir videre assosiert med en ID i masteroppgaven og vil ikke på noen måte kunne peke tilbake til deg.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?

Prosjektet skal etter planen avsluttes 3. august 2020. Innen da skal alt av opplysninger om deg tilknyttet lydopptaket være borte.

Dine rettigheter

Så lenge du kan identifiseres i datamaterialet, har du rett til:

- innsyn i hvilke personopplysninger som er registrert om deg, og å få utlevert en kopi av opplysningene,
- å få rettet personopplysninger om deg,
- å få slettet personopplysninger om deg, og
- å sende klage til 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 Norges teknisk-naturvitenskapelige universitet (NTNU) 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:

- Student: Niklas Molnes Hole ([REDACTED])
- Veileder: Monica Divitini ([REDACTED])
- Vårt personvernombud: Thomas Helgesen ([REDACTED]) eller [REDACTED].

Hvis du har spørsmål knyttet til NSD sin vurdering av prosjektet, kan du ta kontakt med:

- NSD – Norsk senter for forskningsdata AS, på epost ([REDACTED]) eller telefon: [REDACTED].

Med vennlig hilsen

Prosjektansvarlig
(Forsker/veileder)

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet *Introduce programming in introductory physics courses through interactive learning platform*, og har fått anledning til å stille spørsmål. Jeg samtykker til:

- å delta i intervju

Jeg samtykker til at mine opplysninger behandles frem til prosjektet er avsluttet

(Signert av prosjektdeltaker, dato)

