Programmering og dybdelæring i fysikk

En kvalitativ studie av elevers arbeid med programmering i fysikk 1

Jonathan Brakstad Waters

Masteroppgave i fysikkdidaktikk
Lektorprogrammet
30 studiepoeng

Fysisk institutt
Det matematisk-naturvitenskapelige fakultet

UNIVERSITETET I OSLO
Mai 2020

Programmering og dybdelæring i fysikk

En kvalitativ studie av elevers arbeid med programmering i fysikk 1

Programming and deep learning in physics

A qualitative study of students' work with programming in physics 1

Forfatter:

Jonathan Brakstad Waters

Veiledere:

Ellen Karoline Henriksen og Tor Ole Odden

Masteroppgave

Lektorprogrammet

30 studiepoeng

Fysisk institutt

Det matematisk-naturvitenskapelige fakultet

UNIVERSITETET I OSLO

Mai 2020

2020

Programmering og dybdelæring i fysikk

Jonathan Brakstad Waters

http://www.duo.uio.no/

Trykk: Reprosentralen, Universitetet i Oslo

Sammendrag

Med innføringen av programmering i nye læreplaner for norsk skole, kommer det mange spørsmål om hvordan programmeringsaktiviteter skal planlegges og gjennomføres i fysikkfaget. Det er ennå ikke umiddelbart klart hva som er målet med programmeringsaktiviteter, hva man mener de skal tilføre faget, eller hvor stor plass de skal få i undervisningen.

Problemstillingen for denne oppgaven er: «*Hvordan kan en programmeringsaktivitet i fysikk 1 bidra til dybdelæring for elevene?*» Utgangspunktet for oppgaven er et undervisningsopplegg for en fysikk 1-klasse, hvor elevene bruker programmering til å simulere et vertikalt kast med og uten luftmotstand. Datainnsamlingen ble gjort i to klasser, og er tredelt: Det er tatt skjerm- og lydopptak av flere elever under gjennomføringen av opplegget, elevene har sendt inn skriftlige svar på to spørsmål om læring og motivasjon, og det er gjennomført fire fokusgruppeintervjuer i etterkant av undervisningsopplegget. Det er gjort kvalitativ, tematisk analyse av datamaterialet.

For å belyse problemstillingen, tar jeg utgangspunkt i fire elementer som har innvirkning på elevenes dybdelæring: Hva elevene lærer, hvilke utfordringer de har i møte med programmeringsaktiviteten, hvordan elevene arbeider med de ulike representasjonene i opplegget, og hvordan aktiviteten påvirker elevenes eierskapsfølelse. Jeg argumenterer for at programmeringsaktiviteter bør benyttes for å bidra til elevers dybdelæring av fysikkfaglige konsepter og fysikkfagets natur, heller enn omfattende programmeringsferdigheter. Resultatene indikerer at:

- Programmeringsaktiviteten hjelper elevene med å se sammenhenger mellom ulike størrelser i formlene de bruker, og hvordan disse påvirker bevegelse. De lærer noe om både programmering, fysikkfaglige konsepter og hvordan fysikkforskning foregår.

- For at elevene skal oppleve dybdelæring av fysikkfaglige konsepter, må særlig utfordringer knyttet til programmerings- og modelleringssyntaks reduseres.

- Elevene synes det er utfordrende å se sammenhengen mellom de ulike representasjonene i opplegget. For å legge til rette for dybdelæring, trenger de omfattende støtte i å bruke representasjonene på hensiktsmessige måter.

- Elevene føler på eierskap til programmeringsaktiviteten fordi problemstillingen de jobber med oppleves som virkelighetsnær og nyttig. Men det er avgjørende for eierskapsfølelsen at elevene får nok støtte i å forstå programkoden de skriver, ellers opplever de at de ikke kommer frem til svaret selv, noe som reduserer eierskapsfølelsen og potensialet for dybdelæring.

I tillegg til disse resultatene, er også et revidert undervisningsopplegg lagt ved oppgaven. Dermed gir denne oppgaven to viktige bidrag til norsk skolefysikk og fysikkdidaktisk forskning. For det første gir den en enkeltstående programmeringsaktivitet som er klar til bruk. For det andre bidrar den med generelle anbefalinger for gjennomføringen av programmeringsaktiviteter i norsk skolesammenheng, basert på analyse og resultater. Oppgaven viser at programmering kan være et verdifullt bidrag til det norske fysikkfaget.

Abstract

The introduction of computational programming into the new curriculum of Norwegian schools raises many questions about how programming activities should be implemented into physics instruction. It is still unclear what the goal of such programming activities is, what their contribution to the subject is perceived to be, or how big of a role they should play in the teaching of physics.

The main question for this thesis is: "*How can a programming activity in a physics 1 course contribute to students' deep learning?*" The thesis is based on an activity designed for a physics 1 class (year 12 in the Norwegian school system); wherein the students use programming to simulate a vertical throw with and without air resistance. Data was collected from two classrooms and consists of three elements: Recordings of students' screens and dialogue as the activity proceeded; students' written responses to two questions about learning and motivation; and four focus group interviews, conducted after the activity. Data was qualitatively, thematically analysed.

To address the thesis question, I discuss four elements that have an effect on students' deep learning: What the students learn, what challenges they face throughout the programming activity, how the students work with the different representations in the activity, and how the activity affects the students' feeling of ownership. I argue that programming activities should be used to foster deep learning of physics related concepts and the nature of science, rather than extensive programming skills. The results indicate that:

- The programming activity helps the students to see connections between different entities in the formulas they use, and how these entities affect motion. The students learn something about programming, physics related concepts and how physics research is done.

- For the students to experience deep learning of physics related concepts, challenges with programming and modelling syntax must be reduced.

- The students find it challenging to see connections between the different representations in the activity. To foster deep learning, they need extensive support in using the representations in a meaningful way.

- The students feel ownership of the programming activity because the problem they work with feels realistic and useful. But it is crucial for the feeling of ownership that the students receive support in understanding the code they are writing. If not, they risk feeling that they did not find the answer themselves, which reduces the feeling of ownership and potential for deep learning.

In addition to these results, a revised teaching activity is attached to the thesis. Hence, the thesis offers two important contributions to Norwegian physics education research and practice. Firstly, it yields a standalone programming activity ready for use. Secondly, it provides general recommendations for the implementation of programming activities in the Norwegian school setting, based on empirical results. The thesis shows that programming can be a valuable contribution to Norwegian physics education.

Forord

Denne masteroppgaven markerer slutten på mine fem år som lektorstudent med fysikk og matematikk ved Universitetet i Oslo. Underveis har jeg lært mye om både fysikk og skole og elever og pedagogikk, og jeg fullfører som en bedre og klokere utgave av meg selv. Masteroppgaven er på mange måter «kronen på verket», og sammenfatter mye av det jeg har lært om skolefysikk, fysikkdidaktikk, pedagogikk og min rolle som lærer. Jeg håper at oppgaven kan bli lest og satt pris på av mange som er tilknyttet det norske fysikkdidaktikkmiljøet, og at den tilfører noe nyttig til fysikkfaget i norsk skole. Om ikke annet har jeg lært mye mens jeg har lest og analysert og skrevet, og jeg gleder meg til å omsette all teorien til praksis når jeg nå er ferdig utdannet fysikklærer.

I arbeidet med masteroppgaven er det mange som har bidratt underveis. Jeg vil særlig takke mine veiledere Ellen Karoline Henriksen og Tor Ole Odden, som har vært mer tilgjengelige og har brukt mer tid på å lese, diskutere og gi innspill enn det noen kunne forvente. Jeg har fått mailsvar tidlig på morgenen og sent på kvelden, fått konstruktive tilbakemeldinger, ris og ros, og har følt meg meget godt ivaretatt.

Videre vil jeg takke alle som har vært med på å forme undervisningsopplegget som har vært utgangspunktet for oppgaven. Takk til Dan Weller, Bruce Sherwood, Ruth Chabay og John Burk, som har vist interesse for prosjektet og kommet med verdifulle innspill om å undervise med programmering. Takk til Paul Irving, som har lært meg mye av det jeg kan om programmering i Glowscript. Takk også til lærerne ved deltagerskolene som var med på å diskutere og forme ideen bak undervisningsopplegget.

En stor takk rettes til lærerne og elevene som har vært med på prosjektet. Jeg har utelukkende møtt positivitet og samarbeidsvilje – arbeidsomme og hyggelige elever, og hjelpsomme, engasjerte lærere, som alle gjorde at datainnsamlingsprosessen gikk «som smurt». Takk til min medstudent Andreas Fagerheim, som har vært en god samarbeidspartner og uvurderlig ressurs i datainnsamlingsprosessen.

Takk også til hele fysikkdidaktikkseksjonen på UiO. Takk for samtaler i lunsjen, seksjonsmøter, et «hei» i gangen, egen kontorplass, og for all kaffen jeg har drukket. Jeg har stortrivdes som del av en liten, men veldig hyggelig gjeng!

Innhold

Liste over figurer og tabeller

1. Introduksjon

En student som går et bachelor- eller masterløp i fysikk i Norge, vil på et eller annet tidspunkt møte programmering som et effektivt og nyttig verktøy for å regne på en rekke problemstillinger. For eksempel har Universitetet i Oslo sterkt fokus på programmering i fysikkutdanningen (Malthe-Sørenssen, Hjorth-Jensen, Langtangen & Mørken, 2015), og det er flere og flere som tar til orde for at dette fokuset på programmering – både i utdanning og i fysikeryrket – bør reflekteres i skolen. For eksempel skrev Senter for IKT i utdanningen i 2016 at «kunnskapsdomenet programmering kan ha en plass i norsk skole, gjennom hele skoleløpet (…) Det er mange fag i skolen der programmeringskompetanse kan være nyttig og bidra til å gjøre faget mer relevant og motiverende (…)» (Sevik, 2016, s. 7). Aiken (2013) hevder at amerikansk K-12 utdanning (omtrent tilsvarende vår grunnskole og videregående skole) ikke forbereder elever på videre studier og arbeid, fordi elevene ikke lærer de rette ferdighetene, for eksempel programmering. Haraldsrud & Tellefsen (2018) skriver at programmering har mulighet til å endre fysikkfaget med hensyn til innhold, oppgaver, og inkludering.

Når denne oppgaven blir skrevet, foregår det en fornyelse av læreplanverket i den norske skolen, kalt *fagfornyelsen*. Denne prosessen innebærer også en ny læreplan i fysikk, som skal foreligge i løpet av 2020. Som del av fagfornyelsen ser vi at begrepet «programmering» dukker opp både i den nye læreplanen for naturfag, og i kjerneelementene i fysikkfaget (Utdanningsdirektoratet, 2019c, 2020). I utkastet for nye kompetansemål i fysikkfagene (februar 2020) er programmering nevnt eksplisitt i to mål. Elevene skal

- bruke numeriske metoder og programmering til å modellere og utforske bevegelse i situasjoner der akselerasjonen ikke er konstant
- bruke numeriske metoder og programmering til å utforske og modellere bevegelse i to dimensjoner og vurdere resultatene (Utdanningsdirektoratet, 2020)

Gitt den tilsynelatende sterke fremmarsjen av programmering i norsk skole de neste årene, er det behov for å studere nærmere hvordan man kan se for seg at dette skjer, samt hvilken betydning det kan komme til å ha for elevers læring. Den mest umiddelbare måten å introdusere programmering i skolen på, er å la elevene programmere i fagene de allerede har, for eksempel i programfagene (Caballero, Fisler, Hilborn, Romanowicz & Vieyra, 2020; Haraldsrud & Tellefsen, 2018). Et av poengene til Caballero et al. (2020) er at vi vet for lite

om hvilken effekt programmering i realfagene har på elevenes læring. Kan vi se at elevene lærer mer av det tradisjonelle faglige innholdet enn de gjorde før, eller lærer de mindre? Er det hensiktsmessig at elevene bruker tid på å lære seg programmering, eller tar det for mye av fokuset som skulle vært brukt på det tradisjonelle faglige innholdet? Er det kanskje sånn at elevene lærer mer om naturfagenes egenart enn de lærer om begreper og fenomener i naturfagene? Hva gjør programmering med elevenes motivasjon og interesse for faget? Og hvordan skal vi sørge for at lærere har nok kompetanse til å undervise elever med programmering, når mange av dem aldri har programmert før?

Parallelt med økt fokus på programmering, ser vi også at begrepet *dybdelæring* er i vinden, og til dels veiledende i arbeidet med nye læreplaner (Kunnskapsdepartementet, 2016). Dybdelæring handler om å se sammenhenger, fordype seg i problemstillinger over tid, og å reflektere over egen læringsprosess. Det er på generell basis enighet om at dybdelæring er bra, men det er ikke umiddelbart gitt hvordan (eller om) man skal endre hverdagspraksisen i skolen for å oppnå dybdelæring hos elevene.

I denne oppgaven argumenterer jeg for at programmeringsaktiviteter i fysikkfaget kan bidra til at elevene opplever dybdelæring, og sånn sett er en nyttig ressurs for både elever og lærere. Jeg viser til flere aspekter ved slike aktiviteter som kan ha en innvirkning på elevenes dybdelæring, og har særlig fokus på *elevenes utfordringer*, *eierskap*, og *representasjonsformer*. Det viser seg at programmeringsaktiviteter ikke uten videre fører til mer dybdelæring, men at et godt gjennomtenkt undervisningsopplegg har stort potensial for å gi elevene godt læringsutbytte. Problemstillingen for oppgaven er følgende:

Hvordan kan en programmeringsaktivitet i fysikk 1 bidra til dybdelæring for elevene?

For å besvare problemstillingen, har jeg laget og gjennomført et undervisningsopplegg med programmering. Undervisnignsopplegget lar elevene bruke programmering til å simulere et vertikalt kast, både med og uten luftmotstand. Datamaterialet består hovedsakelig av skjerm- og lydopptak av noen elever fra gjennomføringen, samt fire fokusgruppeintervjuer og elevenes skriftlige tilbakemeldinger. I oppgaven diskuterer jeg styrker og svakheter ved opplegget, med utgangspunkt i følgende forskningsspørsmål:

- Hva lærer elevene av å jobbe med programmeringsaktiviteten?
- Hvilke utfordringer møter elevene i arbeidet med programmeringsaktiviteten?
- Hvordan påvirker programmeringsaktiviteten elevenes eierskap til egen læringsprosess?

- Hvordan jobber elevene med de ulike representasjonsformene i programmeringsaktiviteten?

Med nye læreplaner ser vi en økende interesse for programmering i norsk skole, og denne masteroppgaven er enda en brikke i det store puslespillet. I tillegg til å bidra med innsikt i noen viktige aspekter ved programmeringsaktiviteter i fysikkfaget, er resultatet av oppgaven et revidert undervisningsopplegg som kan brukes av norske fysikklærere. Slik utkastet til læreplanen i fysikk ser ut nå, er det ikke utenkelig at programmering i fysikkfaget kan løses av mange lærere ved å gjøre ett eller to enkeltstående opplegg i løpet av året. Programmering "gjennomsyrer" ikke nødvendigvis faget, særlig når lærere ikke har bred kompetanse på feltet. Det er heller ingen grunn til å tro at fysikklærere vil bruke store deler av fysikkundervisningen på å lære opp elevene i programmering, slik at de enkelt kan håndtere mange forskjellige programmeringsoppgaver. Det er derfor nærliggende å tenke at behovet for undervisningsopplegg som dekker kompetansemålene, gir elevene begrenset programmeringserfaring på kort tid, samt er enkle å gjennomføre uten for mye opplæring, kan være nyttige for mange lærere. Om noen år er det godt mulig at både elever og lærere kommer til fysikkfaget med langt mer programmeringskompetanse, men enn så lenge er opplegg som mitt et godt alternativ.

Videre følger en gjennomgang av relevant teori og tidligere forskning, før jeg kort beskriver noen begreper knyttet til programmering, og gir en lengre beskrivelse av det gjennomførte undervisningsopplegget med begrunnelser for valgene jeg gjorde. Deretter beskriver jeg datainnsamlingen og -analysen, og gjør rede for kvaliteten på forskningsdesignet. Til slutt deler jeg resultatene fra analysearbeidet, og diskuterer disse i lys av teori og tidligere forskning.

2. Teori

I dette kapittelet presenteres teori som er relevant for oppgaven – både bakgrunnsstoff som rammer inn oppgaven, og de teoretiske rammeverkene og begrepene som er brukt i analysen.

2.1 Læring

Det finnes mange perspektiver på læring. I denne oppgaven lener jeg meg på teori som kommer fra individualkonstruktivistiske og sosialkonstruktivistiske tradisjoner (Säljö, 2016). Disse tradisjonene tar utgangspunkt i at mennesker *konstruerer* sin egen kunnskap i møte med nye begreper og forestillinger, enten alene (individualkonstruktivisme) eller i samhandling med andre (sosialkonstruktivisme). Videre presenteres noen aspekter ved læring som er relevante for denne oppgaven.

Kognitiv belastning

I likhet med datamaskiner, har den menneskelige hjernen to lagringsenheter; langtidsminnet og arbeidsminnet. Når vi prosesserer ny informasjon, benytter vi oss av arbeidsminnet, som konstruerer *skjemaer*. Disse skjemaene er kognitive strukturer som systematiserer informasjonen slik at den kan lagres i langtidsminnet (Sweller, Van Merrienboer & Paas, 1998). Det er konstruksjonen av slike skjemaer som fører til læring. Fordi arbeidsminnet har begrenset med kapasitet, er det et poeng å utforme læringssituasjoner slik at elevene kan bruke mest mulig kapasitet på å konstruere nye skjemaer, og minst mulig kapasitet på andre ting.

Sweller et al. (1998) beskriver tre typer belastning som finner sted i læringsaktiviteter. *Intern* (intrinsic) belastning er belastning som ligger i materialet som skal læres. Det kan for eksempel være krevende å huske nye begreper, og vite hva de betyr. Denne typen belastning kan læreren ikke gjøre noe med. *Ekstern* belastning er belastning som påføres utenfra, og som ikke bidrar til læring. Dette kan for eksempel være at elevene må bruke mye kapasitet på å tolke en tvetydig oppgavetekst, eller at undervisningen er full av digresjoner som ikke er relevante for det som skal læres. Mengden ekstern belastning avgjøres av undervisningsdesignet, og regnes som unødvendig (Sweller et al., 1998). *Relevant* (germane) belastning forekommer når eleven må bruke ekstra kapasitet på aktiviteter som bidrar til læring. Dette er altså belastning som kreves av undervisningsdesignet, men som kan ha positiv effekt dersom intern og/eller ekstern belastning er lav. For eksempel kan relevant

belastning forekomme dersom elevene tvinges til å reflektere rundt eller fullføre et eksempel som er gitt i læreboken (Sweller et al., 1998).

Den proksimale utviklingssonen

I følge Vygotsky, Cole, John-Steiner, Scribner og Souberman (1978) foregår læring i det som kalles den *proksimale* (eller *nærmeste*) *utviklingssonen*. Modellen går kort forklart ut på at man deler opp en persons kunnskaper og ferdigheter i tre soner. Én sone med det personen kan, én sone med det personen ikke kan, og én sone imellom de to, den proksimale utviklingssonen, med det som personen kan dersom vedkommende får støtte fra en annen, mer kompetent person. Hver gang vi lærer noe nytt, utvides denne midterste sonen, slik at alt vi lærer «peker fremover mot ny kunnskap og nye ferdigheter som nå kommer innenfor rekkevidde.» (Säljö, 2016, s. 118)

Når vi skal håndtere ferdigheter og kunnskaper i den proksimale utviklingssonen, er vi avhengige av støtte fra en annen som er mer kompetent enn oss. Denne andre er ofte en lærer eller annen voksenperson, men det kan også være en medelev, et dataprogram eller kanskje også en bok. Støttestrukturer som hjelper elever med å tilegne seg kompetanse i den proksimale utviklingssonen, kalles gjerne *scaffolding* (Säljö, 2016). *Scaffolding* kan defineres som:

> «Støtte som lar en student nå et mål eller gjennomføre en handling som ikke ville vært mulig uten den støtten, og som bidrar til at studentene kan nå det samme målet eller gjennomføre den samme handlingen uten støtten i fremtiden» (Guzdial, 1994, s. 3, min oversettelse)

Guzdial (1994) skriver at når elever gjør programmeringsaktiviteter i skolen knyttet til modellering og simulering, er de avhengige av scaffolding. Scaffoldingen skal både la elevene gjennomføre programmeringsaktiviteten, og gjøre dem i stand til å lære noe «om og gjennom programmeringsaktiviteten» (Guzdial, 1994, s. 3, min oversettelse). I denne oppgaven blir læreren, oppgaveteksten og dataprogrammet elevene jobber med sett på som slike støttestrukturer, som har til hensikt å hjelpe elevene med gjennomføring og læring av programmeringsaktiviteten.

Dybdelæring

Dybdelæring er et av de sentrale begrepene som brukes i ordskiftet rundt fagfornyelsen og nye læreplaner. Voll, Øyehaug og Holt (2019) understreker at dybdelæring ikke er et nytt

fenomen i den norske skolen, men at begrepet er med på å aktualisere arbeidet med endringer i læreplanene. For at begrepet ikke bare skal være et «motebegrep», er det viktig å være tydelig på hvordan det brukes (Voll et al., 2019). Dybdelæring er definert som «det å gradvis utvikle kunnskap og varig forståelse av begreper, metoder og sammenhenger i fag og mellom fagområder.» (Utdanningsdirektoratet, 2019b) Dette innebærer at elever får arbeide med fagstoff over tid, og at de får utfordringer i takt med sin egen utvikling. Sentralt for dybdelæring er evnen til å overføre kunnskap og forståelse fra kjente situasjoner til ukjente, og at eleven kan bruke det den har lært i nye sammenhenger. Dette står i motsetning til overflatelæring, som i større grad handler om å kunne reprodusere faktaopplysninger, uten å se større sammenhenger (Kunnskapsdepartementet, 2016). Dybdelæring er en prosess, og ikke et resultat. Denne prosessen består av tre deler: Elevene må organisere kunnskaper som fakta, modeller og lover, de må automatisere ferdigheter som å løse ligninger, tegne grafer eller programmere, og de må utvikle positive holdninger som motivasjon, en følelse av kontroll, og tro på egne evner (Voll et al., 2019).

For at elever i størst mulig grad skal oppleve dybdelæring heller enn overflatelæring, må undervisningsaktiviteter utformes slik at læringsmål og oppgaver «dytter» elevene i den retningen. Dersom aktivitetene elever deltar i oppfordrer til overflatelæring og mekanisk oppgaveløsning, vil dette også være utbyttet. Dersom de heller oppfordrer til dybdelæring, refleksjon rundt egen læringsprosess og fokus på sammenhenger mellom ulike deler av faget, kan dette bli utbyttet. Det er altså mulig å designe undervisningsopplegg som bidrar til elevenes dybdelæring, og undervisningsopplegg som ikke gjør det (Smith & Colby, 2007).

Et forslag til indikatorer på dybdelæring er gitt av Chin og Brown (2000). De foreslår at dybdelæring kan måles i fem kategorier:

- I hvilken grad elevene kommer opp med nye tanker og ideer de ikke har fått presentert
- Hva slags forklaringer elevene kommer med
- Hva slags spørsmål elevene stiller
- Elevenes metakognitive aktivitet
- Elevenes tilnærming til oppgaver/aktiviteter

Når elever tar del i dybdelæring, deler de ofte nye ideer, og de benytter seg av mer presis og utfyllende begrepsbruk enn elever som ikke gjør det. De forsøker også å gi teoretiske forklaringer, og produserer gjerne egne teorier som forklarer det de ser på et mikronivå eller

ved hjelp av tidligere kunnskap og erfaringer. For eksempel er forklaringen «vannet blir hardt fordi molekylene ordner seg i et gitter» et tegn på dybdelæring, mens forklaringen «vannet blir hardt fordi det fryser» ikke er det. Dybdelæring er også synlig i elevenes spørsmål. Mens spørsmål som fokuserer på årsak og virkning, hypoteser og uoverensstemmelser mellom gammel og ny kunnskap er tegn på dybdelæring, er spørsmål som handler om prosess («hva skal vi gjøre nå?») eller fakta («hvilken verdi har *g*?») ikke det. Når elever bruker tid på å reflektere over sin egen læringsprosess, resultatene sine og veien videre, eller de evaluerer sin egen arbeidsinnsats, sier vi at de har høy grad av metakognisjon, som også er et uttrykk for dybdelæring. I sin tilnærming til oppgaver er dybdelærende elever mer aktive og engasjerte, og diskuterer spørsmål, mulige svar og hypoteser på et høyere konseptuelt nivå enn elever som ikke deltar i dybdelæring (Chin & Brown, 2000). Disse kjennetegnene på dybdelæring gir et verktøy som hjelper oss å undersøke om dybdelæring faktisk skjer i klasserommet.

Tabell 1: *KI-rammeverket for læringsprosesser, med definisjoner og operasjonaliseringer hentet fra Taub, Armoni, Bagno og Ben-Ari (2015).*

KI-Prosess	**Definisjon**	**Operasjonalisering**
Elisitere ideer	Elever blir gjort oppmerksomme på kunnskap de har fra før	Elever uttrykker forkunnskaper ved å f.eks. huske fakta og formler
Legge til nye ideer	Elever blir introdusert for ideer som er nye for dem	Elever blir introdusert for nye ideer fra programmet, medelev eller lærer
Utvikle kriterier for å evaluere ideer	Elever utvikler og bruker spørsmål og tester for å vurdere om nye ideer er akseptable eller ikke	Elever vurderer informasjon og resultater som gyldige eller ugyldige
Sortere og reflektere	Elever reflekterer over og sorterer forskjeller og likheter mellom ny og gammel kunnskap, basert på de kriteriene de har utviklet	Elever uttrykker fysikkforståelse ved å organisere, tolke og forklare nye og gamle ideer

Kunnskapsintegrasjon

Taub et al. (2015) har forsket på hvordan bruken av programmering bidrar til elevers læring, og tok i den sammenheng i bruk et teoretisk rammeverk kalt *Knowledge Integration*, kunnskapsintegrasjon (KI). Teorien er hentet fra Linn og Eylon (2011), og sier at kunnskapsutvikling foregår i fire faser eller prosesser (Taub et al., 2015, min oversettelse): *Elisitere ideer, legge til nye ideer, utvikle kriterier for å evaluere ideer,* og *sortere og reflektere*. De fire prosessene forekommer ikke nødvendigvis kronologisk i den presenterte rekkefølgen, og flere av prosessene kan skje samtidig. Når jeg har sett etter disse prosessene i datamaterialet, har jeg lenet meg på operasjonaliseringen gitt av Taub et al. (2015), som vist i tabell 1.

2.2 Eierskap til egen læringsprosess

Eierskap er knyttet til det engelske begrepet *epistemic agency*. En «epistemisk agent», eller en som tar eierskap over sin egen læringsprosess, defineres av Stroupe (2014 s. 488, min oversettelse) som «individer eller grupper som tar, eller får, ansvar for å forme kunnskapen og praksisene i et fellesskap.» Når elever får mer ansvar for sin egen læringsprosess, øker motivasjonen, og dybdelæring kan foregå (Enghag & Niedderer, 2008; Voll et al., 2019). Dette henger tett sammen med elevenes *autonomi*, eller *følelse av selvbestemmelse* (Stefanou, Perencevich, DiCintio & Turner, 2004; Voll et al., 2019), samt en oppfatning av at fagstoffet som læres er *relevant* (Enghag & Niedderer, 2008; Voll et al., 2019). Denne følelsen av relevans og selvbestemmelse bidrar til en følelse av eierskap og ansvar for kunnskapsutvikling i klasserommet, og er med på å fostre dybdelæringsprosesser.

Inspirert av Enghag og Niedderer (2008) og Stefanou et al. (2004), beskriver jeg tre ulike former for eierskapsfølelse som elever kan oppleve:

- Elever kan kjenne på *organisatorisk* eierskap i den overordnede organiseringen av skolearbeid – f.eks. hvis de får bestemme hvor mange som jobber sammen i hver gruppe, når elevarbeid skal leveres inn, hvilket tema man skal gå gjennom først, eller hvilken oppgave de skal gjøre.
- Elever kan kjenne på *proseduralt* eierskap i prosessen der en oppgave skal gjennomføres – f.eks. hvis de får bestemme hvordan resultatene skal presenteres og hvilket kildemateriale de bruker for å belyse en problemstilling.

- Elever kan kjenne på *kognitivt* eierskap i sin egen læringsprosess – f.eks. hvis de får jobbe med relevante problemstillinger, svare på sine egne spørsmål, kan finne sine egne løsninger, får omformulere oppgaver så de passer med egne interesser, og får oppgaver tilpasset sine egne styrker og svakheter.

Videre argumenterer Stefanou et al. (2004) for at de to første formene for eierskapsfølelse (organisatorisk og prosedural) ikke alene fostrer motivasjon og dybdelæring, men at det er elevenes kognitive eierskap til sin egen læringsprosess som er avgjørende. Slik eierskapsfølelse forekommer hovedsakelig når elever får lov til å arbeide med sine egne spørsmål over tid, for eksempel ved å komme tilbake til det samme spørsmålet flere ganger i løpet av en økt (Enghag & Niedderer, 2008). Dette skillet mellom organisatorisk/proseduralt og kognitivt eierskap er et effektivt verktøy for å se på elevenes eierskap i møte med programmering, og for å kunne si noe om hvorvidt programmeringsaktiviteten fostrer dybdelæring eller ikke.

Eierskap og motivasjon

Eierskap til egen læringsprosess er tett knyttet til indre motivasjon (*intrinsic motivation)* (Deci, Vallerand, Pelletier & Ryan, 1991). Elever med indre motivasjon for en læringsaktivitet, deltar i den læringsaktiviteten fordi de opplever den som givende, lærerik, og meningsfylt. De deltar fordi de vil, ikke fordi de må. Når elever tar eierskap over sin egen læringsprosess, øker også den indre motivasjonen (Stefanou et al., 2004).

2.3 Representasjonsformer

Vi skiller mellom to former for representasjoner, nemlig *eksterne* og *interne*. De interne representasjonene er de modellene vi ser for oss mentalt for å forstå et fenomen. De eksterne representasjonene er figurer, tekst og andre visualiseringer (Opfermann, Schmeck & Fischer, 2017). I denne oppgaven vil all bruk av ordet *representasjoner* omhandle eksterne representasjoner. I fysikkfaget opererer vi med fem hovedkategorier av representasjonsformer (Angell et al., 2019):

- *Fenomenologiske*: det som «faktisk skjer»
- *Eksperimentelle*: data vi får fra et eksperiment
- *Grafiske*: grafer, tabeller og figurer
- *Matematiske*: ligninger og formler

- *Begrepsmessige*: bruk av definerte og konkrete begreper, muntlig og skriftlig

Det er gjort mye forskning på hvilken effekt representasjoner har på læring. Forskningen tar utgangspunkt i antagelsen om at hjernen har to informasjonskanaler – en auditiv/verbal, og en visuell/billedlig (Opfermann et al., 2017). Tanken er at hver av kanalene har begrenset med kapasitet eller arbeidsminne, og at det er bedre å stimulere begge kanalene samtidig for å utnytte hjernen maksimalt. Det finnes flere teorier for læring med flere representasjonsformer. Både Mayer og Mayer (2005) og Ainsworth (2006) hevder at arbeid med flere representasjonsformer kan bidra til læring på flere forskjellige måter, blant annet ved å legge til rette for dybdelæring. De to artiklene presenterer to litt ulike modeller for hvordan man kan snakke om bruk av flere representasjonsformer (Opfermann et al., 2017), og jeg velger å ta utgangspunkt i Ainsworth (2006) sitt, da det i større grad er vinklet mot en naturfagsutdannende setting, og eksplisitt diskuterer flere varianter av representasjoner.

Ainsworth (2006) hevder at bruken av flere ulike representasjoner (MER – multiple external representations) kan bidra til elevers læring, gitt at de brukes riktig. Ainsworth (2006) beskriver i sitt DeFT-rammeverk (Design-Function-Tasks) tre funksjoner bruken av MER kan tjene, enten sammen eller hver for seg.

- Flere representasjoner kan ha en *komplementerende* eller *utfyllende* funksjon. Det kan enten være ved at hver representasjon inneholder dels ulik informasjon (for eksempel kan et stillbilde med kraftvektorer og en animasjon som beskriver den resulterende bevegelsen gi ulik informasjon om det samme fenomenet), eller det kan være å belyse den samme informasjonen på flere måter (for eksempel inneholder en lineær graf og det tilhørende funksjonsuttrykket den samme informasjonen, men kan legge til rette for å jobbe på vidt forskjellige måter).
- Flere representasjoner kan ha en *begrensende* funksjon, ved at én representasjon legger begrensninger for tolkningen av en annen representasjon. For eksempel kan teksten «hunden står ved siden av huset» tolkes på flere måter, og et bilde av hunden ved siden av huset vil fortelle oss hvilken side hunden står på. På denne måten kan en kjent representasjon bistå en elev i å forstå en ny, ukjent eller tvetydig representasjon.
- Flere representasjoner kan ha en *konstruerende* funksjon, ved at de fostrer dypere forståelse og dybdelæring. Særlig skjer dette dersom elever får abstrahere eller

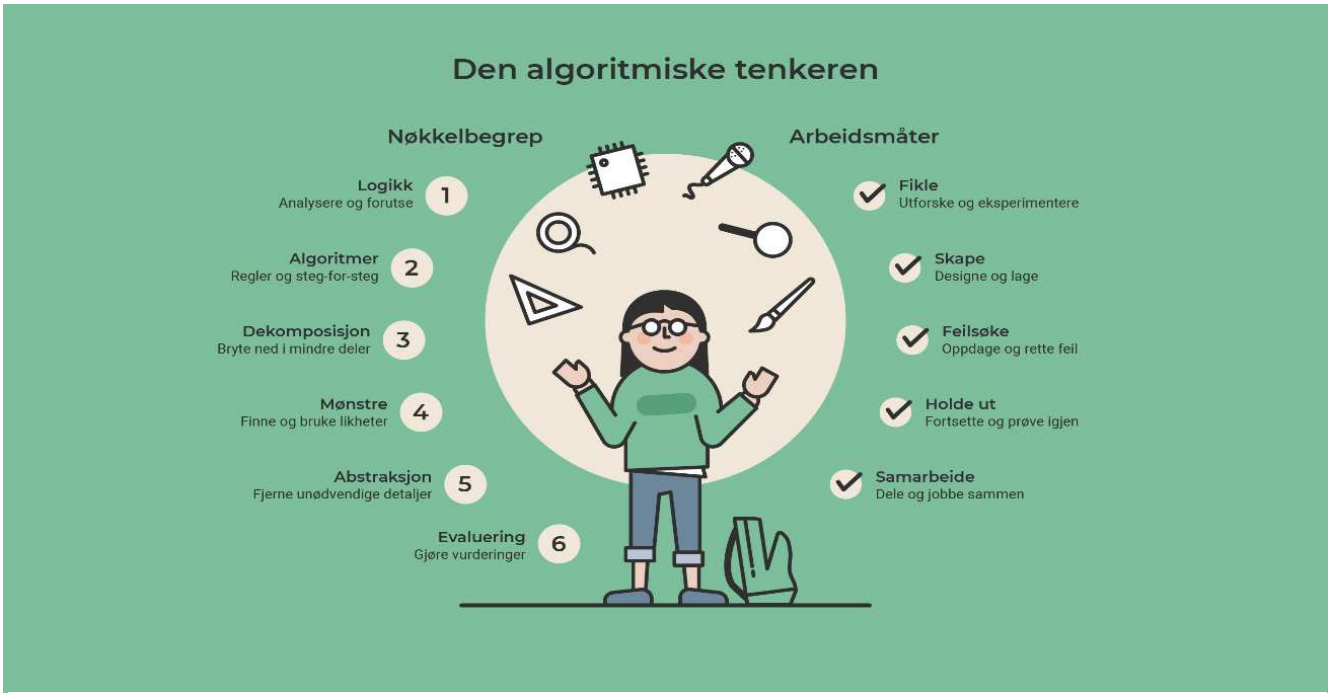
utvide representasjoner, eller om de får jobbe med relasjonen mellom ulike representasjoner.

Det er ikke gjort nok forskning på hvordan ulik bruk av MER bidrar til de ulike funksjonene, og i hvilken grad elever lærer mer eller mindre av ulike metoder (Ainsworth, 2006).

Mens bruken av MER har stort potensiale for økt læringsutbytte og dybdelæring (Opfermann et al., 2017), må det tas høyde for at å bruke flere representasjoner kan være krevende for elever. Særlig gjelder dette om representasjonene er nye. Da må elevene både lære hvordan representasjonen fungerer og skal tolkes, i tillegg til å lære det fagstoffet som representasjonen er ment å støtte opp om. I noen tilfeller er forståelse av en representasjon et mål i seg selv, for eksempel når man underviser grafer i matematikkfaget (Ainsworth, 2006).

2.4 Algoritmisk tenkning

Algoritmisk tenkning er en oversettelse av det engelske begrepet *Computational Thinking* (Utdanningsdirektoratet, 2019a). Dette er et vidt teoretisk begrep, som har flere tolkninger og operasjonaliseringer. Denne oppgaven skal ikke være en diskusjon av dette vide begrepet, og jeg henviser til Nordby (2019) for en god oversikt over hvordan begrepet er definert av flere



Figur 1: *Begreper og arbeidsmåter knyttet til algoritmisk tenkning, hentet fra Utdanningsdirektoratet (2019a).*

ulike forskere og autoriteter i Norge og internasjonalt. En kort presentasjon av begrepet algoritmisk tenkning er likevel på sin plass, da det danner en slags bakgrunn for programmering i fysikkfaget (og andre fag) i skolen.

Utdanningsdirektoratet (2019a) definerer algoritmisk tenkning slik:

«Å tenke algoritmisk er å vurdere hvilke steg som skal til for å løse et problem, og å kunne bruke sin teknologiske kompetanse for å få en datamaskin til å løse (deler av) problemet. I dette ligger også en forståelse av hva slags problemer/oppgaver som kan løses med teknologi og hva som bør overlates til mennesker.»

Algoritmisk tenkning er altså mer enn bare å programmere, det handler også om problemløsning og evaluering. Figur 1 viser begreper og arbeidsmåter som er viktige i sammenheng med algoritmisk tenkning og problemløsing, ifølge Utdanningsdirektoratet (2019a).

Tabell 2: *Weintrop et al. (2016) sin taksonomi for algoritmisk tenkning i realfagene, med oversettelse hentet fra Nordby (2019).*

Datahåndtering	**Modellering og simulering**	**Digital problemløsning**	**Systemtenkning**
Samle data	Bruke digitale modeller for å forstå et begrep	Forberede problemer til å løse dem digitalt	Undersøke et komplekst system som en helhet
Generere data	Bruke digitale modeller til å finne og teste løsninger	Programmere	Forstå sammenhenger innad i et system
Behandle data	Vurdere digitale modeller	Velge effektive beregningsverktøy	Tenke i nivåer
Analysere data	Utforme digitale modeller	Vurdere ulike tilnærminger/løsninger til et problem	Kommunisere informasjon om et system
Visualisere data	Implementere digitale modeller	Utvikle modulære digitale løsninger	Definere systemer og håndtere kompleksitet
		Lage digitale abstraksjoner	
		Feilsøking og feilretting	

Modellen fra Utdanningsdirektoratet er ikke spesiell for realfagene. Weintrop et al. (2016) har utviklet en taksonomi for å beskrive hvilke algoritmiske praksiser som finner sted i matematikk- og naturfagklasserom. Tabell 2 er en norsk oversettelse gitt av Nordby (2019).

Det er tydelig at Weintrop et al. (2016) sin taksonomi i større grad er sentrert rundt det digitale aspektet av algoritmisk tenkning, men vi ser likevel at mange av begrepene fra Utdanningsdirektoratet sin plakat går igjen. Noe av det vi ønsker når vi lar elevene våre programmere, er at de skal lære å bli gode problemløsere, som kan analysere og plukke fra hverandre avanserte problemer, og så løse dem ved hjelp av forskjellige teknikker, inkludert programmering. Vi vil at de skal lære seg å vurdere og evaluere resultatene sine, at de skal kunne visualisere og presentere data, og at de skal bli gode på å lete etter og rette opp feil. Programmering kan også bidra til skaperlyst og utforskertrang, og elevene kan oppmuntres til å lage noe eget.

2.5 Elevers utfordringer i møte med programmeringsaktiviteter

For å finne ut hva slags støttestrukturer vi skal bruke for å hjelpe elever med programmering i fysikkfaget, trenger vi å vite hvilke utfordringer de støter på. Det er begrenset med forskning på hva slags utfordringer elever møter når de skal kombinere programmering, modellering og fysikk, og derfor også få utarbeidede rammeverk for å studere dette. I kapittelet om tidligere forskning er det samlet noen mer konkrete eksempler på hva ulike forskere har funnet av utfordringer hos elever og studenter i møte med programmering i fysikk, men disse har til felles at de ikke er systematisert. Når jeg leter etter og diskuterer elevers utfordringer knyttet til programmering i fysikkfaget, forholder jeg meg til kategoriseringen gjort av Basu et al. (2016), se tabell 3.

Basu et al. (2016) deler elevers utfordringer med programmering i fysikk i tre store kategorier. *Fagspesifikke utfordringer* er knyttet til de fysikkfaglige konseptene elevene skal lære. Dette innebærer bruk av Newtons andre lov, hvordan en strøm påvirkes av et magnetfelt, hvordan man kan integrere hastighet for å få posisjon, også videre. Man kan si at fagspesifikke utfordringer er knyttet til det «tradisjonelle» fysikkfaget. *Utfordringer knyttet til programmering* er rent programmeringstekniske utfordringer, som innebærer hvordan en kodesnutt må skrives for å fungere. Her faller ting som rekkefølgen på ulike kodelinjer, hvordan ulike kommandoer fungerer, og hvordan man feilsøker et program. *Utfordringer*

knyttet til modellering og simulering kan sies å være skjæringspunktet mellom de to andre kategoriene, og handler om utfordringer elever møter når de skal bruke programmering til å utføre fysikk. Kategorien innebærer å overføre fysikkunnskap til programkode, velge riktige initialbetingelser og parametere, og analyse og evaluering av kode og resultater.

Tabell 3: *En kategorisering av elevers utfordringer når de programmerer i fysikk, hentet fra Basu et al. (2016).*

Type utfordring	**Konkrete utfordringer**
Fagspesifikke utfordringer	F.eks. problemer med å tolke bevaringsloven for mekanisk energi eller andre fysikkrelaterte utfordringer
Utfordringer knyttet til modellering og simulering	Identifisere relevante enheter, prosesser og sammenhenger
	Velge passende initialbetingelser
	Systematiske utfordringer – når eleven utforsker retningsløst heller enn å gjøre en strategisk analyse
	Spesifisere parametere for en modell
	Validere en modell
Utfordringer knyttet til programmering	Semantiske utfordringer – hva betyr kommandoene, og i hvilken rekkefølge skal de komme?
	Benytte programmeringstekniske konstrukter, f.eks. variabler, løkker og if-tester
	Feilsøking

3. Tidligere forskning

I dette kapittelet sammenfatter jeg noe tidligere forskning som er relevant for analysen og diskusjonen i denne oppgaven. Først kommer en oversikt over noen forsøk på å innføre programmering både på videregående- og universitetsnivå, og erfaringer derfra. Deretter kommer noen avsnitt som nærmere beskriver forskning gjort på programmering og læring, utfordringer for elever som programmerer i fysikk, og eksterne representasjoner i fysikk.

3.1 Noen forsøk på innføring av programmering i fysikkfaget

I dette avsnittet sammenfatter jeg noe av forskningen som har blitt gjort på implementering av programmering i fysikkundervisning, både på videregående- og universitetsnivå. Særlig har jeg ansett forskning på videregående skole-nivå de siste ti årene som relevant, og jeg har prøvd å inkludere det som måtte finnes, i sitt begrensede omfang. På høyere nivå er det gjort betraktelig mer forskning. Siden dette ikke først og fremst er en litteraturstudie, har jeg valgt noen eksempler fra de siste ti årene, men hevder på ingen måte å presentere alt som finnes. Utgangspunktet er Nordby (2019) sin litteraturstudie, men jeg har supplert med noen artikler skrevet etter (og samtidig) med den, samt enkelte artikler som er referert i tekstene Nordby henviser til. Underveis i gjennomgangen dukker det opp noen programmeringstekniske begreper. Flere av disse er forklart i neste avsnitt, «Et undervisningsopplegg med programmering».

Det er etter hvert gjort flere og flere forsøk på å implementere programmering fysikk i høyere utdanning, både i Norge og i andre land. I Norge kan vi trekke frem Universitetet i Oslo. Der har det over mange år vært satset på programmering og beregninger i fysikkundervisningen. Studentene får programmeringsopplæring i et eget emne i første semester, hvor de lærer seg å bruke Python. Programmeringen de lærer tas hyppig i bruk i andre emner. Implementeringen er særlig tydelig i mekanikkemnet i andre semester, men det jobbes også med utnyttelsen av programmering i andre emner (Malthe-Sørenssen et al., 2015). Malthe-Sørenssen et al. (2015) hevder at det er viktig å ha et eget introduksjonsemne hvor studentene lærer programmering. I tillegg peker de på «mangel på godt læringsmateriell og gode eksempler» som en stor utfordring.

I USA er det flere eksempler på implementering av programmering i fysikkundervisning på universitetsnivå. Chabay og Sherwood (2015) har skrevet læreverket *Matter and Interactions*,

som tar for seg fysikk på introduksjonsnivå. I forordet til læreverket skriver de: «Computational modelling is now as important as theory and experiment in contemporary science and engineering. We introduce you to serious computer modelling right away to help you build a strong foundation in the use of this important tool» (Chabay & Sherwood, 2015, s. x). Programmering og programmeringsoppgaver går igjen i læreverket, og legger opp til bruk av VPython for beregninger i tre dimensjoner.

Caballero, Kohlmyer og Schatz (2012) innførte programmering som del av et introduksjonskurs i mekanikk ved Georgia Tech. Forfatterne har mye erfaring med *Matter and Interactions*, og har derfor tatt utgangspunkt i den erfaringen når de har designet kurset i mekanikk. Kurset fungerer slik at studentene lærer å lage et VPython-program i en «datalab» på universitetet, før de får i lekse å modifisere programmet de har laget for å løse et nytt problem. I motsetning til Malthe-Sørenssen et al. (2015), hevder Caballero et al. (2012) at store studentgrupper kan lære nok programmering i et fysikkemne til å bruke den til noe nyttig.

Dette ser vi også ved University of Toronto, hvor Serbanescu, Kushner og Stanley (2011) bruker VPython med studenter på første års mekanikkurs, og Python med fysikkstudenter på andre året. De hevder at studentene som bruker VPython setter pris på visualiseringsmulighetene programmet gir, og at det å se utviklingen av et fysisk system sammen med «kvantitativ output» var med å utvikle intuisjon om fysiske konsepter. Et annet interessant aspekt ved denne artikkelen, er at Serbanescu et al. (2011) virker å være opptatt av koblingen mellom laboratorieforsøk og programmering. For eksempel virker det å være sterkt fokus på behandling av innsamlede data med Python sine plottemoduler.

I fysikk på videregående skole-nivå er det færre erfaringer å støtte seg på. Det skrev Grover og Pea (2013), og seks år senere skriver Nordby (2019) det samme. Likevel finnes det noen eksempler. I skoleåret 2013-2014 gjorde en stipendiat ved University of Massachusetts Lowell forsøk på å integrere programmering i tre fysikklasser på en videregående skole (High school) (Aho, Chandra & Roberts, 2014). Elevene programmerte i språket R, og Aho et al. (2014) hevder at mange klarte seg ganske bra, gitt at de ikke hadde programmert før. Det mest effektive hjelpemiddelet for elevene var en liste over de vanligste kommandoene R tilbyr, og eksempler på hvordan disse kommandoene kan brukes.

I 2015 undersøkte Taub et al. (2015) hvordan programmering i fysikkundervisningen påvirker elevenes læring. Studien ble gjort på elever på videregående skole-nivå, i en klasse

for «flinke» elever fra flere forskjellige skoler, og forskerne benyttet seg av skjermopptaksverktøy med mikrofon, og observasjonsnotater. Resultatene fra studien tilsier at programmering kan fremme andre læringsprosesser enn «vanlige» fysikkoppgaver. Taub et al. (2015) tar utgangspunkt i rammeverket *Knowledge Integration* (se kapittel 2.1), og hevder at den læringsprosessen som fremkommer hyppigst i datamaterialet, er når elever «utvikler kriterier for å evaluere ideer». Dette skjer når elevene stiller spørsmål om gyldigheten eller kvaliteten til noe de får presentert (eller har laget), og når de sammenligner flere ideer. Potensialet til programmering i fysikkundervisningen understrekes, og det pekes blant annet på at programmering tvinger elevene til å bevege seg mellom ulike abstraksjonsnivåer, som er en ettertraktet ferdighet. På den annen side advarer de om at programmeringssituasjonen er kompleks for elevene, og det er viktig å begrense den kognitive belastningen. Det er krevende å lære både fysikk og programmering samtidig, og undervisningen må være slik at elevene kan fokusere på fysikken, ikke bare programmeringen. Som et forslag peker Taub et al. (2015) på at man kan gjøre oppgaveteksten så tydelig som mulig for elevene, slik at de bruker tid på de læringsprosessene og metodene man vil de skal bruke tid på. De foreslår også å lære noe grunnleggende programmering isolert fra fysikken før man går i gang med fysikkorienterte oppgaver.

Så sent som i 2019 gjorde Bott, Weller, Irving og Caballero (2019) et arbeid for å utvikle et spørreskjema om elevers holdninger til programmering. De har gjennomført flere programmeringsopplegg med Glowscript i tre klasser på videregående skole-nivå, og gjort en pilotundersøkelse for å kartlegge skjemaets kvalitet. Som en bonus har de skrevet noe om hva elevene svarte på undersøkelsen. For eksempel uttrykker elevene at programmering er nyttig fordi det er raskere og mer nøyaktig enn å regne for hånd. Elever skriver også at å kunne *se* resultatet av arbeidet sitt (som en animasjon eller graf på skjermen) er en fordel ved programmering. Forskerne påpeker at dette kan være fordi Glowscript er visualiseringsbasert av natur, og det er ikke sikkert man hadde fått samme respons om man hadde brukt et annet miljø med mer numerisk output. Bott et al. (2019) sitt arbeid er nyttig, fordi de peker på noen ting man kan undersøke videre. Det kommer blant annet frem at man bør se nærmere på hvordan elever forholder seg til programmering som et «prøv og feil»-verktøy for utforskning og eksperimentering. Et annet interessant poeng de trekker frem, er at man må være bevisst forskjellen mellom elever som mener programmering er et verktøy for å lære fysikk, og de som mener at man må kunne fysikk for å programmere.

I den norske skolen er det lite forskning på integrasjon av programmering i fysikkfaget. Fra 2018 har flere skoler tilbudt programfaget «Programmering og modellering X», som skal «gi en innføring i algoritmisk tankegang og gjøre elevene i stand til å bryte natur- og samfunnsvitenskapelige problemer ned i delproblemer. (…) Faget skal legge til grunnlag for (*sic*) kreativitet, kritisk sans og metodeinnsikt i realfagene.» (Utdanningsdirektoratet, 2017) Det er ennå ikke publisert en evaluering av faget.

I 2019 gjennomførte Nordby (2019) tre undervisningsopplegg med programmering i fysikk 1 og 2, som del av sin masteroppgave. Klassene som deltok kom fra den samme videregående skolen, i utkanten av en norsk storby. Gjennomføringen av oppleggene er ikke systematisk analysert, men forfatteren gir likevel en omfattende erfaringsdeling basert på observasjonsnotater. Programmeringen foregår i Python, ved hjelp av programmeringsmiljøet Spyder (fra Anaconda-pakken)[1]. Det poengteres hvordan tekniske ting som installering, lagring, filbehandling og lignende potensielt utgjør utfordringer for mange elever. Det er også gjennomgående at elevene synes det er selve programmeringen som er utfordrende, og at de synes det er krevende å forstå all koden, heller enn å forstå fysikken. Elevene trenger ikke forstå all koden for å bruke den, men flere uttrykker at de ikke liker denne måten å jobbe på, fordi de ønsker å forstå alt. Til tross for disse utfordringene, ser Nordby (2019) positive trender også. I likhet med Aho et al. (2014) opplever han at mange elever klarer oppgavene de får, og at man kan introdusere relativt avanserte programmeringsaktiviteter ganske fort, slik at man kan begynne med fysikkorienterte problemer. Mange av utfordringene knyttet til programmeringskoden kan modereres ved at læreren er tydelig i sine instruksjoner og undervisning, samt at elevene får god støtte og tett oppfølging underveis. Man ser også at programmeringen stimulerer til fysikkdiskusjoner mellom elevene, og at elevene uoppfordret spør hverandre om hjelp og sammenligner resultater. Nordby (2019) foreslår å følge elever nærmere gjennom undervisningen, for eksempel med skjerm- og lydopptak, for å gjøre en grundigere analyse av «elevenes arbeid og diskusjoner».

For å oppsummere, ser vi at det er et klart behov for mer forskning på hvordan innføring av programmering i det norske fysikkfaget skal fungere. Det er nærliggende å tro at eventuell forskning om læringspotensial og nødvendig støtte har god overføringsverdi fra universitetsnivå til lavere nivåer. Et annet felles trekk vi ser i forskningen på ulike nivåer, er at de fleste omfattende forsøk på innføring av programmering tilsynelatende orienterer seg

[1] https://www.spyder-ide.org/

rundt mekanikk, noe som også gjenspeiles i utkastet til ny læreplan (Utdanningsdirektoratet, 2020). Men det er ikke gitt at all forskning på fordeler og utfordringer ved programmering i fysikk i høyere utdanning er overførbart til norsk videregående skole. For eksempel vet vi at fysikklærere konkurrerer med mange andre fag om elevenes tid og oppmerksomhet, og tiden man kan sette av til programmeringsaktiviteter er potensielt kortere enn man kan tillate seg på universitetsnivå. Det er heller ikke utenkelig at elevsammensetningen er ganske annerledes i en fysikk 1-klasse enn på et bachelorkull. Det kan for eksempel resultere i at forskning på holdninger og motivasjon i sammenheng med programmering gir ulike resultater på videregående- og universitetsnivå.

3.2 Et utvalg av forskning relevant for denne oppgaven

Programmering og læring i fysikk

Selv om det er mange anerkjente forskere som tar til orde for at programmering og algoritmisk tenkning kan bidra til fysikkæring (f.eks. Guzdial, 1994; Malthe-Sørenssen et al., 2015; Wing, 2006), er det ennå begrenset med forskning som gir empirisk evidens for hvilke effekter programmering har på læring av tradisjonelle fysikkfaglige konsepter (Caballero et al., 2020; Nordby, 2019). I gjennomgangen over er det illustrert ulike erfaringer knyttet til programmeringsaktiviteter, som alle peker på styrker og svakheter ved programmering som et læringsverktøy, men det er kun Taub et al. (2015) som eksplisitt leter etter teoretiske indikatorer på læring mens elever arbeider med en programmeringsaktivitet.

Taub et al. (2015) bruker KI-rammeverket beskrevet i kapittel 2.1, til å lete etter tegn på at elevene faktisk lærer når de programmerer. De finner at programmeringsaktiviteten består av fire domener som alle bidrar til læringsprosessene i KI-rammeverket. De fire domenene er

- *Det strukturelle domenet*, hvor elever må bruke fysikkunnskap for å sette opp en fornuftig struktur i programkoden sin, f.eks. ved å velge hensiktsmessige variabler og objekter, og sette fornuftige navn på disse.
- *Det prosedurale domenet*, hvor elever må bruke fysikkunnskap for å beskrive de underliggende mekanismene som fører til bevegelse i programmet. Dette inkluderer håndtering av inn- og utdata, å benytte seg av riktige fysiske lover, og oversette disse til programkode.

- *Det systemiske domenet*, hvor elever må bruke fysikkunnskap for å beskrive og forstå sammenhengen mellom ulike objekter i et system, og særlig hvordan objekter i animasjonsvinduet påvirker hverandre.
- *Det iverksettende domenet*. Når elevene kjører koden sin, får de både en refleksjon av sin egen fysikkunnskap, og kanskje også ny informasjon om situasjonen de modellerer. Elevene tvinges til å evaluere simuleringen, og se etter eventuelle feil.

Med dette som bakgrunn, hevder Taub et al. (2015) at programmering i fysikk kan fremme læring, og at programmering fremmer andre læringsprosesser enn tradisjonell, algebrabasert undervisning.

Elevers utfordringer i møte med programmering i fysikk

Som allerede nevnt, er det en fare for at elever opplever kognitiv overbelastning når de skal jobbe med både programmering og fysikk samtidig (Taub et al., 2015). Selv om det etter hvert er gjort mange forsøk på å integrere programmering i fysikkundervisning, er det få systematiske gjennomganger av hva elever opplever som krevende og utfordrende når de jobber på denne måten. I dette avsnittet prøver jeg å oversiktlig presentere noen av funnene som er gjort, og sortere dem under de tre kategoriene til Basu et al. (2016). Også her er mange av referansene hentet fra gjennomgangen til Nordby (2019). Jeg presenterer hver artikkel for seg, og en inndeling etter kategori er å finne i tabell 4.

I tillegg til Basu et al. (2016), har også Caballero et al. (2012) gjort et omfattende analysearbeid av feil som gjøres av studenter i introduksjonskurs i mekanikk. I dette prosjektet er det kun studentenes besvarelser som er analysert, og sånn sett er det deres *feil*, heller enn deres *utfordringer* som er indikert. Men feilene som er funnet lar seg plassere i de samme tre kategoriene som Basu et al. (2016) bruker. Studentene har fagspesifikke feil knyttet til å regne ut krefter og akselerasjon med Newtons andre lov. De har også programmeringstekniske utfordringer knyttet til feilsøking og tolking av feilmeldinger, men det viser seg at det er få syntaksfeil. Av feil knyttet til modellering og simulering er det særlig fire eksempler som gjentar seg. Studentene har problemer med å oversette Newtons andre lov til et uttrykk som kan brukes i koden. Dette kan for eksempel være at de skriver loven som en ligning i koden, uten å først ha løst ligningen for én variabel. Studentene har også vanskeligheter med å skrive løkken i kodeteksten på en iterativ måte. Det vil si at en variabel gjerne får en ny verdi, som ikke er avhengig av variabelens gamle verdi. Eksempelvis kan de skrive at v = a*dt, heller enn v = v + a*dt. Eller de kan ha skrevet kodelinjer som skulle vært

inne i en løkke, på utsiden av løkken, slik at variabler ikke oppdateres i det hele tatt. Til sist ser Caballero et al. (2012) at mange studenter bruker feil initialbetingelser i koden sin.

Sørby og Angell (2012) har analysert studenters arbeid med programmering i et mekanikkurs på Universitetet i Oslo. I artikkelen sin argumenterer de for at studentene opererer i ulike moduser – en matematikkmodus, en fysikkmodus og en programmeringsmodus. Det viktigste funnet herfra er at studenter i programmeringsmodusen i større grad «prøver og feiler», det som Basu et al. (2016) kaller systematiske utfordringer forbundet med modellering og simulering. Studentene som Sørby og Angell (2012) observerer, programmerer gjerne uten en klar plan, og i stedet for å fokusere på fysikken som er utgangspunktet for koden, endrer de ulike verdier og sammenligner koden med tidligere, lignende programmer de har skrevet.

Tabell 4: *Noen utfordringer som er funnet blant elever som programmerer i fysikk, sortert i kategoriene til Basu et al. (2016).*

Fagspesifikke utfordringer	Problemer med å regne ut summen av krefter og sette dette inn i koden	Caballero et al. (2012)
Utfordringer knyttet til modellering og simulering	Systematiske utfordringer: Prøving og feiling	Sørby og Angell (2012)
	Elever har vanskelig for å velge riktige initialbetingelser	Caballero et al. (2012)
	Vanskeligheter med oppdatering av variabler	Caballero et al. (2012)
	Vanskeligheter med å oversette N2L til et uttrykk datamaskinen forstår	Caballero et al. (2012)
Utfordringer knyttet til programmering	Feilsøking: Er det kodefeil eller fysikkfeil?	Caballero et al. (2014)

Slike systematiske utfordringer henger tett sammen med evnen til å feilsøke. Caballero et al. (2014) har gjennomført flere programmeringsaktiviteter i et fysikkurs i en amerikansk

niendeklasse. De peker på at elevene strever med feilsøking. Særlig synes elevene at det er vanskelig å vite hvilke feil som skyldes fysikkforståelse, og hvilke feil som skyldes programmeringstekniske ting. I tillegg til dette poengterer Caballero et al. (2014) at elevene glemmer mye av programmeringen mellom hver gang de gjennomfører en aktivitet, og at tiden mellom de ulike programmeringsaktivitetene er for lang til at elevene får bygget en forståelse for de programmeringstekniske konseptene.

Vi ser altså at de utfordringene som tidligere er funnet når man analyserer elevers arbeid med programmering i fysikk lar seg kategorisere i de tre kategoriene til Basu et al. (2016). Det tyder på at kategoriseringen er et hensiktsmessig utgangspunkt for å analysere slike utfordringer.

Eksterne representasjoner i fysikk

Kohl og Finkelstein (2008) har forsket på hvordan fysikkstudenter forholder seg til flere eksterne representasjoner, og hvordan de bruker disse i problemløsning. For å studere dette, har de sammenlignet nybegynnere med viderekomne studenter («eksperter»). Forskerne så både på hvor lang tid studentene brukte på å jobbe med ulike representasjoner i problemløsningsprosessen, og hvilke aktiviteter de tok del i da de jobbet med representasjonene. Seks forskjellige aktiviteter ble kodet: Planlegging, oversetting, implementering, utforsking, analysering og verifisering.

Kohl og Finkelstein (2008) fant at både studenter med mye og lite fysikkerfaring brukte omtrent det samme antallet representasjoner, og brukte omtrent tilsvarende med tid på hver representasjon. Forskjellen på de to gruppene i arbeidet med eksterne representasjoner, var hovedsakelig i hvilken grad de analyserte (strategisk og målrettet arbeid med problemløsing) eller utforsket (prøving og feiling, ustrategisk arbeid med problemløsing). Det er kanskje ikke overraskende at mer erfarne fysikere brukte mer tid på å analysere, mens de med mindre erfaring brukte mer tid på retningsløs utforsking.

Til tross for at kompetanse i å forstå og bruke flere ulike representasjoner er anerkjent som en fundamental del av fysikkfaget (f.eks. Angell et al., 2019), mangler det forskning på hvordan representasjoner brukes av elever, og hvordan man best kan bruke ulike representasjoner for å fostre fysikklæring og god representasjonskompetanse (Kohl & Finkelstein, 2017). Det er for eksempel uklart om elever trenger eksplisitt undervisning i hvordan de skal bruke ulike representasjoner, eller om elever lærer å bruke representasjoner kun ved å måtte håndtere dem i forbindelse med fagspesifikt innhold. Kohl og Finkelstein (2017) presenterer resultater

som tyder på at elever presterer veldig ulikt på oppgaver med ulike representasjoner, selv om de inneholder det samme faglige stoffet, og argumenterer for at kunnskap ikke er uavhengig av representasjonsform. Eksempelvis betyr det at selv om elever kan regne på Newtons andre lov og vise at akselerasjonen er null om kraftsummen er null, så er det ikke sikkert at de kan tegne et fri legeme-diagram som viser et tilsvarende tilfelle. Det er derfor et poeng å undervise de samme konseptene med mange forskjellige representasjonsformer.

Det er ikke gitt hvordan et programmeringsmiljø med kode og/eller simuleringer og animasjoner skal passe inn med de representasjonsformene vi allerede er vant med i skolen. Gravel og Wilkerson (2017) hevder at digitale miljøer kan tjene tre ulike funksjoner som representasjoner. De kan enten brukes som simuleringer eller eksperimenter elever kan manipulere, de kan brukes til å presentere eller formidle kunnskap, og de kan brukes som et verktøy for å lære mer om fysikkfaglig innhold. Dette er i samsvar med Taub et al. (2015) sine resultater om læring, og passer godt med elementer fra Weintrop et al. (2016) sin taksonomi for algoritmisk tenkning.

3.3 Oppsummering av tidligere forskning

Det utvalget av forskning som er presentert, om enn noe begrenset, indikerer at det er behov for forskning som kan si noe om læringseffektene av programmering, samt hvordan programmering kan (eller ikke kan) fasilitere elevers læring om og med ulike representasjoner. Det er tydelig at programmering tilfører noe nytt og annerledes til fysikkfaget, og det er behov for mer forskning som kan si noe om hvordan programmering kan (eller skal) brukes, og hvilke læringseffekter det har. Den observante leser vil også registrere at det ikke presenteres noe forskning her som sier noe om hvordan programmering i fysikk påvirker elevers eierskapsfølelse. Det skyldes den enkle grunn at jeg ikke har klart å finne noe.

Jeg ønsker å bygge videre på forskningsgrunnlaget rundt programmering i fysikkutdanning, ved å finne spesifikke deler av en programmeringsaktivitet som er utslagsgivende for elevenes dybdelæring i fysikk. Målet er å peke på noen helt konkrete aspekter ved programmering som bidrar til dybdelæring, og hvordan vi kan tilnærme oss disse aspektene når vi designer undervisningsopplegg. Dette er resultater som kan være relevante for programmering i fysikkutdanning på mange nivåer og i mange settinger, men særlig er de

relevante for den norske videregående skolen, hvor både elever og lærere mangler erfaring med programmering. I møte med nye læreplaner er det svært aktuelt å se på hvordan programmeringsaktiviteter kan være noe mer enn «bare en ekstra ting».

4. Et undervisningsopplegg med programmering

I dette kapittelet vil jeg først beskrive Glowscript og Trinket.io[2], før jeg beskriver prosessen og vurderingene som er gjort i utvikling av programmeringsaktiviteten jeg har gjennomført i klasserommet.

4.1 Glowscript og Trinket.io

Det finnes mange ulike *programmeringsspråk*, og mange ulike *programmeringsmiljøer*. Et programmeringsspråk er et sett med kommandoer, funksjoner og operasjoner, som settes opp etter spesifiserte syntaktiske regler.[3] Eksempler på slike språk er Python, C++, Fortran og Java. Tradisjonelt skrives koden i en teksteditor, før den kjøres fra datamaskinens terminalvindu. Et programmeringsmiljø (på engelsk IDE – integrated development environment) er et program som lar deg skrive og kjøre kode i et gitt språk.[4] Fordelen med slike miljøer er at man har alt på et sted, og det er kort vei mellom koden man skriver og resultatene den gir når man kjører den. Miljøene har ofte støtte for autokorrektur, innebygde feilsøkingsmekanismer, egne grafikkinnstillinger og systematisering av store prosjekter. Eksempler på slike miljøer er Spyder (for Python), Qt Creator (for blant annet C++) og Microsoft sitt Visual Studio (som støtter mange forskjellige språk). Det finnes også programmeringsmiljø på nett, som enkelt lar deg programmere uten å installere software på datamaskinen din.

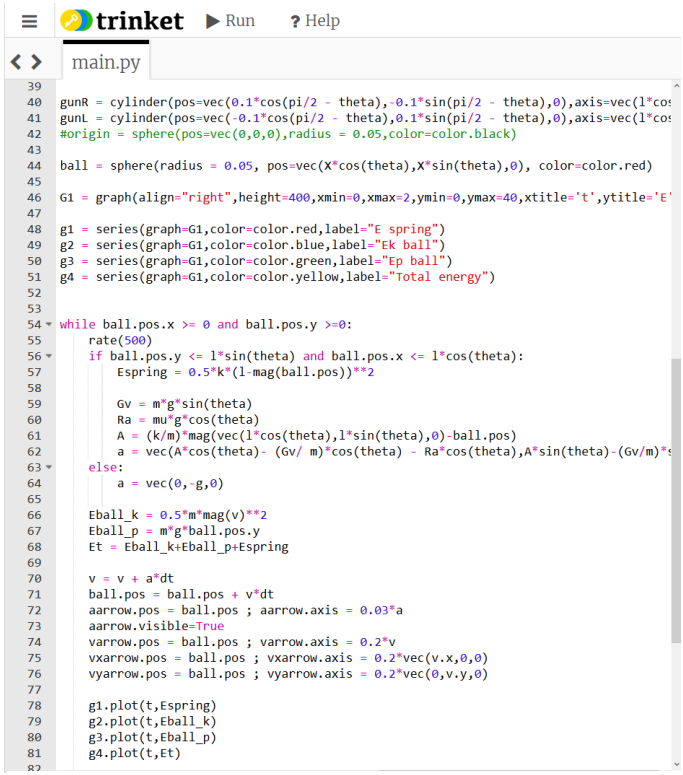
Programmeringsaktiviteten som er beskrevet i denne oppgaven, tar i bruk Trinket.io, som er et slikt nettbasert programmeringsmiljø for programmeringsspråket Glowscript. Glowscript finnes også under navnet VPython (Visual Python), og består av Python og en tredimensjonal grafikkmodul. Resultatet er et miljø hvor elevene kan skrive kode med Python-syntaks, som lett lar seg animere.

Kort oppsummert inneholder Glowscript et sett med objektklasser, for eksempel bokser, piler og kuler. Hver gang man kaller på et objekt i disse klassene, ved å for eksempel definere en kule, dukker dette objektet automatisk opp i et grafikkfelt. Dersom koordinatene til objektet endres i koden, flyttes objektet i grafikkfeltet. Resultatet er at man med svært få endringer

[2] http://trinket.io
[3] https://snl.no/programmeringsspråk
[4] https://en.wikipedia.org/wiki/Integrated_development_environment

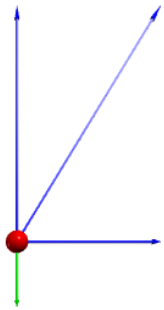
kan skrive koden sin slik man ville skrevet et vanlig Python-program, og likevel se resultatene sine som en animasjon.

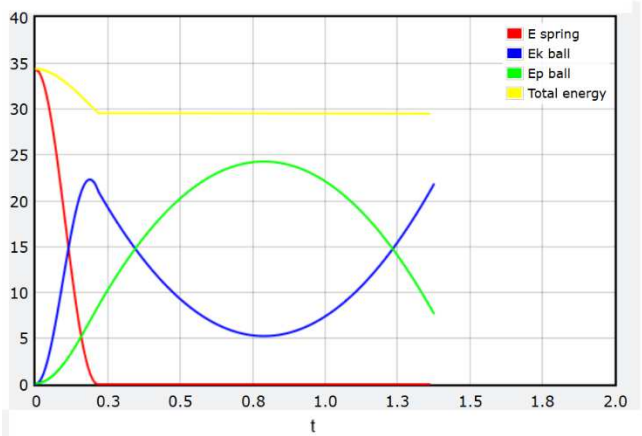
En av fordelene med å bruke Glowscript fremfor vanlig Python, er at animasjonen gir umiddelbar og tydelig tilbakemelding til elevene. Dersom man skal lage tilsvarende animasjoner i Python, krever det mye tid og et relativt høyt programmeringsnivå. Det finnes flere løsninger for å programmere i Glowscript, for eksempel ved å installere VPython-modulen, eller å logge inn på Glowscript.org. Jeg har valgt å bruke Trinket.io.



Figur 2: *Kodevinduet i Trinket.io, med en eksempelkode.*

Trinket.io er en nettside som legger til rette for å skrive og dele Glowscript-kode raskt og effektivt. Brukergrensesnittet består av to vinduer: et kodevindu (figur 2) hvor man skriver koden sin, og et resultatvindu (figur 3) som viser animasjonen, eventuelle grafer, og utskrevet tekst. En programkode med tilhørende resultater kalles gjerne en *Trinket*.





Figur 3: *Animasjonsvindu og grafvindu hentet fra Trinket.io, produsert med koden i figur 2. Standardinnstillingene i Trinket viser grafen under animasjonen, ikke ved siden av. Figuren viser en ball som skytes diagonalt opp i luften. De blå pilene er fartsvektoren og dens x- og y-komponenter, og den grønne pilen er akselerasjonsvektoren.*

På Trinket.io er det også mulig å lage egne *kurs*, som tar brukeren gjennom flere sider med tekst, formler og kode. Kursene kan være interaktive, slik at brukeren kan kjøre og endre på kodene som presenteres. For å bruke funksjonaliteten til Trinket.io kreves ingen installasjon av noen programvare. Nettsiden kjører i alle nettlesere, og krever kun enkel innlogging med e-post og passord. Sånn sett er det godt egnet for bruk i klasserom og i andre enkeltsituasjoner.

4.2 Undervisningsopplegg: Vertikalt kast med og uten luftmotstand

Undervisningsopplegget som er utgangspunktet for denne oppgaven ble laget i løpet av høsten 2019. Valgene knyttet til denne prosessen er tatt dels på grunnlag av min egen erfaring fra lektorstudiet, og dels på grunnlag av verdifulle innspill fra veiledere, medstudenter, og forskere fra både Universitetet i Oslo og andre utdanningsinstitusjoner nasjonalt og internasjonalt. Flere av disse har mye erfaring med programmering for elever, studenter og lærere, for eksempel Dan Weller (Michigan State University), Ruth Chabay, og Bruce Sherwood (utvikler av Glowscript). Ideen til oppleggets utgangspunkt kom fra lærere ved den ene deltagerskolen.

Undervisningsopplegget har en tidsramme på 5x45 minutter, og dreier seg om et vertikalt kast både med og uten luftmotstand. Tanken er at elevene skal få innblikk i hvordan luftmotstand påvirker bevegelse i tyngdefeltet, og at de skal lære hvordan man kan modellere bevegelse ved hjelp av programmering. Utgangspunktet var følgende kompetansemål fra utgående (men fortsatt gjeldende) læreplan i 2020 (Utdanningsdirektoratet, 2006):

- *identifisere kontaktkrefter og gravitasjonskrefter som virker på legemer, tegne kraftvektorer og bruke Newtons tre lover*
- *gjøre rede for situasjoner der friksjon og luftmotstand gjør at den mekaniske energien ikke er bevart, og gjøre beregninger i situasjoner med konstant friksjon*
- *bruke parameterframstilling til å beskrive rettlinjet bevegelse for en partikkel, og bruke derivasjon til å regne ut fart og akselerasjon når posisjonen er kjent, både med og uten digitale verktøy*
- *bruke matematiske modeller som kilde for kvalitativ og kvantitativ informasjon, presentere resultater og vurdere gyldighetsområdet for modellene*

- *samle inn og bearbeide data og presentere og vurdere resultater og konklusjoner av forsøk og undersøkelser, med og uten digitale verktøy*
- *bruke simuleringsprogrammer til å vise fenomener og fysiske sammenhenger*

Ut ifra disse, og knyttet til det konkrete opplegget, konstruerte jeg fem læringsmål:

1. Forstå prinsippene bak enkel programmering, og bruke disse til å utforske konsepter i mekanikk ved hjelp av VPython (Variabler, løkker, tidssteg)
2. Forstå akselerasjon som en fartsendring fra et lite tidssteg til et annet (Og tilsvarende fart som en posisjonsendring fra et lite tidssteg til et annet)
3. Kunne bruke en løkke til å undersøke bevegelse over tid, med og uten konstant akselerasjon
4. Kunne bruke programmering til å undersøke fenomener som friksjon og luftmotstand
5. Ha noe innsikt i hvordan moderne fysikere bruker programmering som verktøy

Deretter begynte arbeidet med utformingen av undervisningsopplegget. Den første avgjørelsen handlet om hvilket programmeringsmiljø som skulle brukes. Det ble fort bestemt at jeg ønsket at så lite tid som mulig skulle gå med til forberedelser som installasjon og lignende. Nordby (2019) erfarte at å begynne økten med slike forberedelser, som gjerne tar lang tid, reduserte kvaliteten på undervisningsopplegget. Derfor var det svært nærliggende å bruke et internettbasert programmeringsmiljø. Trinket.io ble valgt av flere grunner. For det første ligger det syntaksmessig nært opp til Python. Python er et utbredt språk for introduksjon til programmering, og egner seg godt til programmering av fysikkproblemer. Syntaksen er relativt enkel, og det er færre punktum, parenteser og semikolon enn i for eksempel C++ eller Matlab. Fordi elever som møter programmering senere kan ha nytte av å ha sett noe før som ligner, virker et Python-lignende språk å være hensiktsmessig. For det andre ble animasjonsverktøyet i Trinket.io ansett som positivt for elevenes læring og/eller motivasjon. For det tredje gjør Trinket det lett for elevene å finne koden som jeg skriver (siden den kan deles med en internettlenke), og lett for meg (eller en lærer) å se koden som elevene skriver. I tillegg er plattformen smidig og enkel å bruke, så elevene slipper å bruke mye tid på å lære å navigere verktøyet de skal benytte seg av.

Sengupta, Kinnebrew, Basu, Biswas og Clark (2013) hevder at visuell programmering (eller blokk-programmering) er mer egnet til undervisning enn tekstbasert programmering, fordi det senker terskelen for elevene. Visuell programmering kan beskrives som «dra og slipp» med

ulike bokser med ferdigprogrammerte funksjoner. Trinket.io har innebygget støtte for slik programmering, men jeg valgte likevel å benytte meg av tekstbasert programmering. Dette henger hovedsakelig sammen med argumentet over, nemlig at det er nyttig for elevene å se noe de kommer til å se igjen senere. For å kompensere for en noe brattere læringskurve, er oppgavesettet utformet slik at mye av koden elevene skal skrive er gitt på forhånd.

Inspirert av Serbanescu et al. (2011), tar undervisningsopplegget utgangspunkt i en laboratorieøvelse elevene hadde gjort på forhånd. Der skjøt de en liten stålkule opp i lufta med en fjærkanon, og målte både startfart og maksimal høyde. Den målte maksimale høyden ble sammenlignet med resultatet man får ved bevaring av mekanisk energi, nemlig $h = \frac{v^2}{2g}$. Forsøket viste at kulen ikke kom så langt som man skulle tro dersom den mekaniske energien var bevart, og ga elevene grunn til å tenke at luftmotstand bremser kulen på vei opp.

Dette er bakteppet for en programmeringsaktivitet hvor elevene skal simulere en slik oppskutt kule. En oversikt over de ulike bolkene er gitt i tabell 5. De første 45 minuttene brukes til å gå gjennom et Trinket-kurs[5] hvor elevene introduseres for grunnleggende programmeringsfunksjoner, i tråd med Taub et al. (2015) sitt forslag om noe isolert programmeringsundervisning. Dette inkluderer hvordan man skriver ut noe til terminalen, hvordan man kan utføre regnestykker og lagre verdier i variabler, hvordan en løkke fungerer, og hvordan man kan oppdatere en variabel ved å legge til en verdi. Jeg underviste, mens elevene fulgte med i kurset på hver sin datamaskin, og gjorde aktivitetene der. Deretter følger en undervisningsøkt på ca. 20 minutter, hvor læreren går gjennom hvordan man kan modellere bevegelse ved hjelp av while-løkker og formler for konstant akselerasjon og hastighet. Metoden som undervises heter «Euler-Cromer»-metoden, men elevene opplyses ikke om det.

Etter introduksjonskurs og undervisning, har elevene igjen ca. 2,5 klokketimer til å gjennomføre et oppgavesett. I oppgavesettet skal de først lage en simulering av et vertikalt kast uten luftmotstand. Denne sammenlignes med resultatet for bevaring av mekanisk energi, slik at elevene kan se at koden kjører, og gir riktige resultater. Deretter skal de legge til luftmotstanden. Ved å variere luftmotstandskoeffisienten *k*, er målet å finne en slik *k* som gjør at kulen i simuleringen går like høyt som kulen i laboratorieforsøket med fjærkanonen. Denne

[5] Det reviderte kurset ligger her: https://trinket.io/jonathan_b_waters-5946/courses/intro-til-programmering

verdien for *k* anses å være «den riktige», siden den stemmer med virkeligheten (oppgaven fokuserer ikke på feilkilder, men det er selvfølgelig ganske stor usikkerhet i de eksperimentelle resultatene fra forsøket). Helt til slutt i oppgavesettet får elevene i oppgave å variere initialhastigheten til kulen, for å se hvilken effekt luftmotstanden har på små og store hastigheter. De siste ti minuttene av økten ble bruk til at elevene besvarte to «læringsspørsmål» som inngår i datamaterialet.

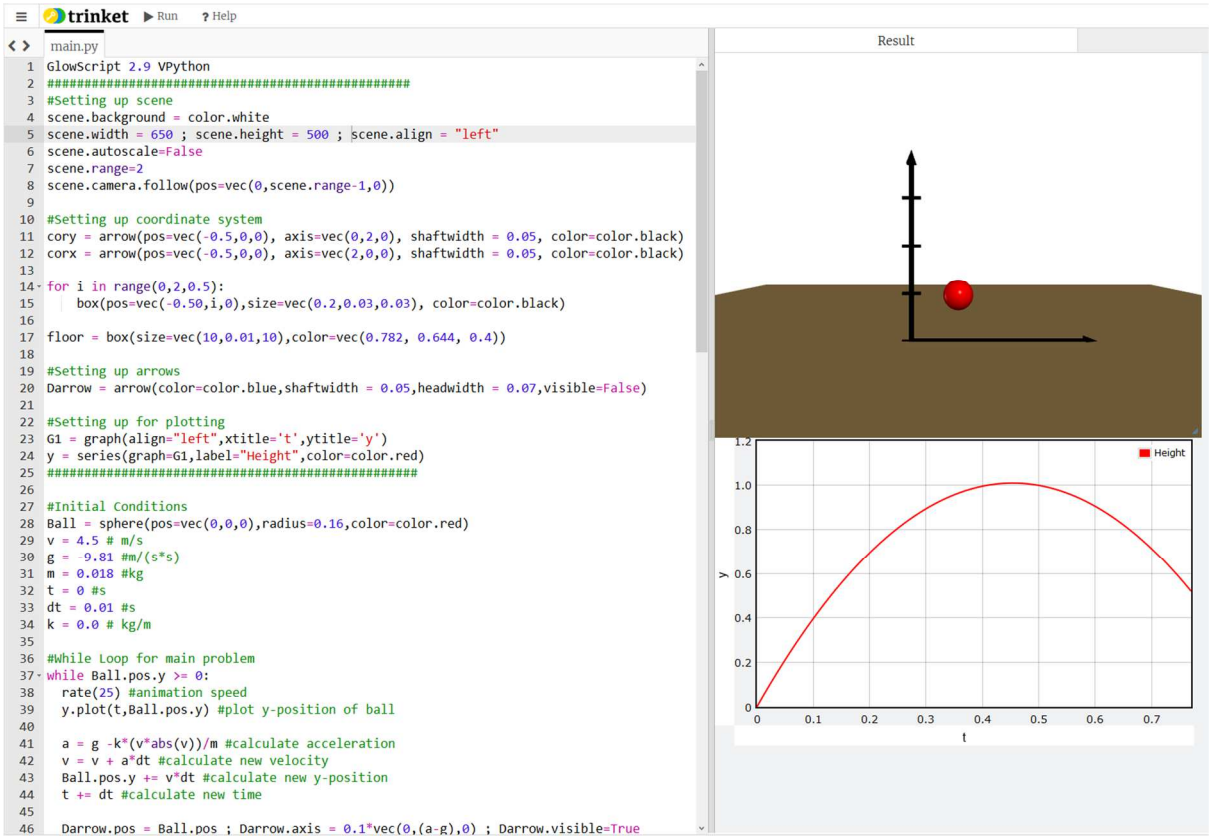
Tabell 5: *En oversikt over undervisningsopplegg om vertikalt kast og programmering, slik det ble gjennomført under datainnsamlingen.*

Tidsramme (anslagsvis)	**Aktivitet**
45 minutter	Undervisning i grunnleggende programmeringskonsepter og bruk av Trinket
20 minutter	Undervisning i hvordan man simulerer bevegelse i programmeringsmiljøet, ved hjelp av løkker og vei/fart/tid-formler
140 minutter	Elevene jobber med programmeringsaktiviteten «Vertikalt kast med og uten luftmotstand» og eventuelt ekstraoppgaver
10 minutter	Elevene besvarer læringsspørsmål

Det var tenkt at oppgavesettet skulle oppfylle følgende funksjoner. For det første ville jeg at elevene skulle oppleve det som motiverende å jobbe tett opp til et forsøk de hadde et forhold til, og få lov til å bruke det de kan til å lære noe «ekte» om verden. Jeg ville også at oppgavene skulle motivere til utforskning av ulike bevegelser, parametere og variabler, og at elevene skulle få tilstrekkelig støtte til å kombinere programmering med fysikk. Til sammen tenkte jeg at dette ville gi elevene gode forutsetninger for å nå læringsmålene.

Elevene jobbet to og to med å løse oppgavene. Basert på forslag fra Dan Weller, og med støtte fra McDowell, Werner, Bullock og Fernald (2002), valgte jeg å gå for parprogrammering (Pair Programming), hvor to elever programmerer sammen på én datamaskin. Tanken var at elevene skulle bytte regelmessig på hvem som programmerte, og hvem som skrev inn svar i oppgavedokumentet (på en annen datamaskin). Selv om Nordby (2019) erfarte at elever som jobbet sammen på én datamaskin ikke var like motiverte, tenkte jeg at en organisert variant av parprogrammering ville oppmuntre til samarbeid, og fungere bedre enn der hvor elever kun jobber sammen på grunn av tekniske utfordringer.

Oppgavearket ble delt ut til elevene via It's Learning. Da den siste økten var slutt, ble elevene bedt om å laste opp oppgavearket sitt på It's Learning, med link til programmet de hadde skrevet i Trinket. Det gjør at både jeg og elevenes lærer har tilgang til det de har gjort. Oppgavesettet inneholder også en hel del ekstraoppgaver, men disse er kun i liten grad relevante for resultatene og diskusjonen av prosjektet. Oppgavearkene som elevene arbeidet med, er gitt i vedlegg E. Et eksempel på elevenes ferdige kode, med beskrivende kommentarer, er tilgjengelig i vedlegg F. For å enkelt henge med i presentasjonen og diskusjonen av resultatene, anbefales leseren å skumme gjennom oppgavesettet, og sette seg inn i eksemplet (koden er også vist i figur 4, men lesbarheten er ikke optimal).




```
GlowScript 2.9 VPython
################################################
#Setting up scene
scene.background = color.white
scene.width = 650 ; scene.height = 500 ; scene.align = "left"
scene.autoscale=False
scene.range=2
scene.camera.follow(pos=vec(0,scene.range-1,0))

#Setting up coordinate system
cory = arrow(pos=vec(-0.5,0,0), axis=vec(0,2,0), shaftwidth = 0.05, color=color.black)
corx = arrow(pos=vec(-0.5,0,0), axis=vec(2,0,0), shaftwidth = 0.05, color=color.black)

for i in range(0,2,0.5):
    box(pos=vec(-0.50,i,0),size=vec(0.2,0.03,0.03), color=color.black)

floor = box(size=vec(10,0.01,10),color=vec(0.782, 0.644, 0.4))

#Setting up arrows
Darrow = arrow(color=color.blue,shaftwidth = 0.05,headwidth = 0.07,visible=False)

#Setting up for plotting
G1 = graph(align="left",xtitle='t',ytitle='y')
y = series(graph=G1,label="Height",color=color.red)
################################################

#Initial Conditions
Ball = sphere(pos=vec(0,0,0),radius=0.16,color=color.red)
v = 4.5 # m/s
g =  -9.81 #m/(s*s)
m = 0.018 #kg
t = 0 #s
dt = 0.01 #s
k = 0.0 # kg/m

#While Loop for main problem
while Ball.pos.y >= 0:
  rate(25) #animation speed
  y.plot(t,Ball.pos.y) #plot y-position of ball

  a = g -k*(v*abs(v))/m #calculate acceleration
  v = v + a*dt #calculate new velocity
  Ball.pos.y += v*dt #calculate new y-position
  t += dt #calculate new time

  Darrow.pos = Ball.pos ; Darrow.axis = 0.1*vec(0,(a-g),0) ; Darrow.visible=True
```

Figur 4: *Eksempel på et ferdig program skrevet av en elev i Trinket, etter å ha jobbet gjennom oppgavesettet «Vertikalt kast med og uten luftmotstand». Linjene 1-25 er skrevet av lærer og ligger i programmet når elevene åpner det første gang. Resten skrives inn av elevene.*

Slik undervisningsopplegget er utformet, legger det til rette for flere av prosessene beskrevet av rammeverkene for algoritmisk tenkning. Elevene bør få mulighet til å utforske, feilsøke og samarbeide, samtidig som de tenker logisk og ser større sammenhenger. Særlig legger opplegget til rette for at elevene kan sammenligne flere resultater og vurdere gyldigheten av dem. Hvis vi ser på Weintrop et al. (2016) sin taksonomi, er det flere av prosessene som bør

bli synlige når elevene arbeider. De samler og genererer data både ved å gjøre et laboratorieforsøk, og ved å kjøre kode de har skrevet. Dataene behandles og tolkes, og visualiseres i tall, grafer og animasjoner. De må også programmere for å konstruere en modell. Men mest relevant er det kanskje at elevene bruker modellen sin til «å forstå et begrep», og til «å finne og vurdere løsninger».

5. Metode

I dette kapittelet presenteres metodevalgene for både datainnsamling og dataanalyse. Utgangspunktet for datainnsamlingen var utarbeidelsen – og gjennomføringen – av undervisningsopplegget beskrevet over.

5.1 Datainnsamling

Undervisningsopplegget ble gjennomført i to klasser på to ulike videregående skoler utenfor Oslo. Det var jeg som gjennomførte opplegget, og var fungerende lærer i klassene, men i alle øktene hadde jeg med meg én medstudent, samt elevenes vanlige lærer. Begge disse assisterte ved å gå rundt og hjelpe elever som ba om hjelp. Oppgavesettet ble ikke endret fra klasse til klasse, men ordlyden og formuleringen i undervisningsbolken ble justert mellom øktene, etter å ha observert en del vanskeligheter i den første klassen. Særlig ble simulering av bevegelse presentert annerledes, ved å koble Euler-Cromer-metoden tettere til vei-fart-tid-formlene for konstant fart.

Datamaterialet består i hovedsak av tre komponenter. Jeg har data fra klasserommet i form av fire skjermopptak med lyd. Elevene leverte også oppgavebesvarelsene sine, som inkluderte to spørsmål om motivasjon og læring. Den største delen av datamaterialet er hentet fra fire fokusgruppeintervjuer med elever, som ble gjennomført i etterkant av undervisningsopplegget i hver klasse. I tillegg til disse tre datakildene, skrev jeg også kortfattede observasjonsnotater etter hver time. Disse er ikke brukt som del av analysen, annet enn å bidra til «det store bildet». I dette avsnittet vil jeg beskrive utvalget mitt, samt hvordan datainnsamlingen foregikk.

Utvalg

Alle informanter er hentet fra de to klassene hvor opplegget ble gjennomført. I den ene klassen var det 14 jenter og 11 gutter. I den andre klassen var det 10 jenter og 11 gutter. Til sammen 12 gutter og 13 jenter deltok i fokusgruppeintervjuer. Det ble etterstrebet kjønnsbalanse i hvert enkelt intervju, samt blant elevene som gjorde skjermopptak i klasserommene.

Utvalget er helt klart et bekvemmelighetsutvalg, på alle fronter. Skolene er hentet via mitt nettverk, og ble valgt fordi det var lettvint. Elevene som har deltatt i skjermopptak og

fokusgruppeintervjuer har alle meldt seg frivillig. Det er vanskelig nok å finne elever som vil være med, så jeg har ikke stilt krav om spredning i ferdighetsnivå, fysikkfaglig kompetanse eller lignende. Klassene viste seg å være omtrent som man forventer en gjennomsnittlig fysikk 1-klasse å være. Elevene på begge skoler var jevnt over arbeidsomme, og fremsto på et middels til høyt fysikkfaglig nivå. Arbeidsmiljøet i timene var preget av engasjement og lærelyst. Også i fokusgruppene virker det å være både elever som har gode resultater i fysikk, og elever som har mindre gode resultater.

Troverdigheten til en studie kan være preget av hvordan utvalget er satt sammen, og det finnes flere teknikker for å sette sammen et strategisk utvalg som gir større grunnlag for generaliserbarhet (Firebaugh, 2018). Likevel anser jeg mitt bekvemmelighetsutvalg som både forsvarlig og hensiktsmessig for denne oppgaven. Undervisningsopplegg som det som er presentert her vil alltid variere i sin gjennomføring (og vellykkethet) fra klasse til klasse, og det er nærliggende å tro at det likevel kan danne grunnlag for en god diskusjon om aspekter ved programmering i skolefysikken.

Fokusgruppeintervjuer

Fokusgruppeintervjuene ble gjennomført i fire grupper – tre med seks elever, og én med syv elever. To av intervjuene ble gjennomført av meg, og to ble gjennomført av en medstudent. Selv om 25 elever er representert i intervjuene, er det gjennomgående at noen elever snakker mye, mens andre elever nesten ikke sier noen ting.

Intervjuene var semistrukturerte (Dalen, 2011), og intervjuguiden er vedlagt i vedlegg B. Spørsmålene er utarbeidet med støtte i flere ting. Først og fremst danner forskningsspørsmålene for masteroppgaven grunnlag for mye av intervjuguiden. I tillegg er det hentet spørsmål og inspirasjon fra Bott et al. (2019), og intervjuguiden til Ytterhaug (2015). Jeg har etterstrebet å stille spørsmål som er minst mulig ledende, men som likevel peker mot helt konkrete aspekter ved undervisningsopplegget elevene har vært med på. Elevene bes også forklare en programsnutt i Trinket, som ligner den de selv har skrevet i timen. Dette fungerer blant annet som en slags lakmustest på om elevene har forstått noe av det de har gjort.

Intervjuguiden begynner med en kort åpningsdel, med spørsmål om elevenes oppfatning av fysikkfaget generelt. Dette har til hensikt å få elevene i snakk, og å introdusere temaet vi skal snakke om (Krueger, 2014). Hoveddelen av intervjuguiden kan grovt deles i tre kategorier. Under overskriften «Elevenes opplevelse av programmering i fysikk» spør jeg om hva

elevene synes om opplegget, hva de opplever at de har lært, hva de husker best, hva som var lett og hva som var vanskelig. I tillegg spør jeg om hva de syntes om formatet på oppgaveteksten, om det var nok tekst, om denne var til hjelp underveis, og om hvordan parprogrammeringen fungerte. Under overskriften «VPython spesielt» spør jeg om hvordan det var å jobbe med Trinket. Her kommer spørsmål om hva elevene syntes om å få utdelt ferdig kode, hvor mye de forsto av koden de skrev, og hva slags støtte de fikk fra programmet og animasjonen. Under overskriften «Programmering i fysikk», spør jeg om hvordan opplegget har påvirket elevenes syn på fysikk generelt, og om hvordan de synes programmering påvirker motivasjonen deres. Helt til slutt spør jeg elevene om det er noe vi har glemt, eller som de har lyst å si (Krueger, 2014). Totalt sett er det tenkt at intervjuguiden skal favne tre overordnede temaer: Hva (og når) elevene lærte, hva de syntes var lett og vanskelig, og hva slags effekt opplegget hadde på motivasjonen deres. Fordi intervjuene skulle kodes induktivt, ble det lagt opp til å favne mest mulig av elevenes synspunkter og oppfatninger, for så å filtrere ut siden.

Intervjuene er tatt opp på lydfil, med to opptakere, plassert på hver sin side av bordet elevene satt rundt. Dette sørger for at alle elevene kommer tydelig med på opptakene, og det fungerer som en backup dersom én av opptakerne skulle slutte å fungere. Hverken jeg eller min medstudent tok notater under intervjuene, annet enn å krysse av på spørsmål i intervjuguiden etter hvert som de ble besvart.

Elevenes skriftlige svar

Elevene skrev inn svar i oppgavearket de fikk utdelt. Som del av oppgavesettet var to spørsmål jeg har valgt å kalle *læringsspørsmål*: «Hvordan likte du å jobbe med programmering i fysikkfaget?», og «Hva synes du at du har lært av å jobbe med programmering i fysikk?» (vedlegg C). Tanken var at dette ville la flere elever svare på to viktige spørsmål. På svararket skrev elevene også en link til programkoden sin, slik at jeg kunne se på den i etterkant. Det er totalt 28 skriftlige elevsvar. Elevene fikk beskjed om å levere ett svar pr. elevpar, men enkelte av elevene har levert individuelle svar. Det er umulig å si hvilke av elevene dette er.

Svarene på læringsspørsmålene er tatt med som en del av datamaterialet i analysen, og er analysert sammen med intervjutranskripsjonene. Elevenes svar på de øvrige oppgavene, samt programmene de har laget, er ikke tatt med som en del av datamaterialet. Dette skyldes hovedsakelig mangel på tid og kapasitet. En rask gjennomgang av oppgavesvarene ble

gjennomført, for å sjekke at ikke umiddelbart interessante funn ble oversett, men de er altså ikke analysert.

Skjerm- og lydopptak

Når elevene jobbet med det utdelte oppgavesettet, hadde jeg to par i hver klasse som tok opp både lyd og skjermbilde mens de jobbet. Begge deler ble gjort med programvaren Captura, et enkelt og anvendelig skjermopptaksverktøy.[6] Programmet var installert på datamaskiner jeg hadde med og som elevene lånte. Som backup brukte jeg også en diktafon med mikrofon på hver gruppe. Dette viste seg å være nyttig, ettersom Captura ved to tilfeller krasjet under økten, slik at noe av klasseromsmaterialet kun finnes på lydspor.

5.2 Analyse

Etter datainnsamlingen var ferdig i slutten av januar 2020, ble intervjuene transkribert i løpet av februar. Parallelt med transkribering av intervjuene ble skjermopptakene gjennomgått for å finne potensielle interessante sekvenser, og analysen av intervjuer og skjermopptak begynte i slutten av februar. Det tyngste analysearbeidet ble gjort i mars og april 2020. Jeg vil i det følgende beskrive analyseprosessen for fokusgruppeintervjuer, skriftlige elevsvar og skjermopptak.

Transkribering

Lydopptakene fra fokusgruppene ble transkribert i sin helhet. Alle elevuttalelser har i transkripsjonene blitt tillagt én av elevene, ved pseudonym. Det er tilstrebet at hvert pseudonym kun tilhører én elev, men på grunn av elevenes ofte like stemmer, og til dels lav kvalitet på lydopptakene, kan det forekomme at noen sitater er tillagt feil pseudonym. Dette er kanskje ikke ideelt, men for den videre analyseprosessen og de endelige resultatene, er det ikke avgjørende hvem av elevene som sa hva (Krueger, 2014). Jeg har valgt å ta med alle uttalelser, også de som lyder «mhm» og «ja» i bakgrunnen av en annen uttalelse. Når det gjelder pauser og nøling, har jeg vært noe pragmatisk med hensyn til flyt i setningene, og relevans for analysen. Alle setningsbrudd er markert med «…», men det er ikke indikert om disse bruddene er lengre eller kortere pauser. Lyder som «eh» og stamming er tatt med der det har opplevdes relevant, men er ikke nødvendigvis konsekvent behandlet gjennom alle fire

[6] https://mathewsachin.github.io/Captura/

intervjuene. Etter transkriberingen har jeg hørt gjennom lydopptakene flere ganger mens jeg har lest teksten, for å sortere ut eventuelle feil og unøyaktigheter. Jeg anser transkripsjonene for å være gode gjengivelser av det som ble sagt i intervjuene.

Skjerm- og lydopptakene fra elevene i klasserommet varer totalt tolv timer, og ble ikke transkribert, da dette ville føre til mye jobb jeg ikke anså som hensiktsmessig. I stedet ble de gjennomgått på dobbel hastighet, mens jeg noterte underveis hvor jeg kunne finne spennende situasjoner, sekvenser og interaksjoner, som jeg kaller «høydepunkter». Disse høydepunktene ble senere sett på nytt, mens jeg tok observasjonsnotater basert på teoretiske begreper fra litteraturen.

Tematisk analyse av intervjuer og skriftlige elevsvar

Etter at intervjuene var transkribert, ble de kodet induktivt ved tematisk analyse, som beskrevet av Braun og Clarke (2006), i programmet Atlas.ti[7]. Intervjuene ble kodet sammen med elevenes skriftlige besvarelser på læringsspørsmålene. Målet med den første analysen var å identifisere hvilke temaer som var gjennomgående i datamaterialet, og å kartlegge hva elevene snakker om når de snakker om programmering. Fordi det er begrenset med forskning på elevers oppfatning av programmering i fysikkfaget, virket det hensiktsmessig å være åpen for mange ulike typer resultater, i stedet for å benytte et teoretisk rammeverk fra start.

Etter jeg hadde lest gjennom datamaterialet flere ganger, og hadde gjort en første runde med koding, besto kodesettet av mellom 50 og 60 koder. Disse ble sortert i seks overordnede temaer: «Læring», «representasjoner», «motivasjon», «utfordringer & støtte», «undervisningsopplegg» og «diverse». Temaet «undervisningsopplegg» inneholdt koder som «parprogrammering», «tekst» og andre nøkkelord som beskriver utformingen av undervisningsopplegget og designet av oppgaveteksten. Temaet «diverse» inneholdt koder som «bra», «liker ikke», «andre programmeringsspråk», og andre nøkkelord som ikke umiddelbart passet inn i noen av de andre kategoriene. Det ble besluttet å ha fokus på de fire første temaene.

Videre fant jeg teori og tidligere forskning som er gjort på hvert av disse temaene, og hentet sentrale begreper og kategoriseringer som er funnet og konstruert av andre. Dette var utgangspunktet for en ny koderunde, hvor mange av de gamle kodene ble byttet ut med mer presise teoretiske begreper. Det endelige kodesettet består av fem overordnede kategorier,

[7] https://atlasti.com/

med totalt 20 koder, i tillegg til 7 koder som ikke nødvendigvis passer inn i de store kategoriene, men som likevel har vært nyttige for å analysere datamaterialet. En oversikt over alle kategoriene er gitt i resultatkapittelet (kapittel 6), og en detaljert beskrivelse av alle kodene ligger i vedlegg A. I det følgende beskriver jeg kort hver enkelt kategori og kodene den består av. Enkelte av kodene er *semantiske*, i den forstand at de ligger nær ordlydene fra intervjuet og er enkle å bruke. Andre koder er mer *latente*, i den forstand at de i større grad er brukt som en tolkning av datamaterialet (Braun & Clarke, 2006). Som et ledd i en gjennomsiktig analyse, presiserer jeg underveis hvilke koder som er mer og mindre tolkningsbaserte.

Kategorien «Representasjoner» er brukt på alle tilfeller der elevene omtaler de ulike representasjonene de møter i programmeringsaktiviteten: Forsøket med fjærkanonen, kodevinduet hvor de skriver kode, animasjonen og grafen som vises i Trinket, og formler og ligninger de skriver enten for hånd eller i koden. Hensikten med denne kodekategorien er å lett kunne identifisere hvordan ulike representasjoner henger sammen med de andre kategoriene. Kodene i kategoriene er jevnt over enkle å bruke, fordi det kommer tydelig frem av intervjuet når elevene snakker om hvilke representasjoner.

Kategorien «DeFT» er brukt på tilfeller der elevene uttrykker at ulike representasjoner hadde én eller flere av de tre funksjonene fra DeFT-rammeverket; utfyllende, begrensende, og konstruerende funksjoner (Ainsworth, 2006, se kapittel 2.3). Hensikten med kodekategorien er å identifisere på hvilke måter de ulike representasjonene støtter elevenes læring og arbeid med oppgavene. Fordi elevene aldri nevner disse funksjonene eksplisitt, innebærer kodene i denne kategorien en stor grad av tolkning fra min side. Noen steder er det tydelig at elevene får støtte av representasjonene, men det kan være vanskelig å vite akkurat hvilken funksjon representasjonene utgjør. Jeg har gjort mitt beste for å kun bruke disse kodene når jeg mener det er liten tvil om at de passer.

Kategorien «Motivasjon» er brukt på tilfeller hvor elevene uttrykker enten positiv eller negativ motivasjon knyttet til undervisningsopplegget. Kodene i kategorien er knyttet til elevenes opplevelse av eierskap, hvorvidt de oppfatter aktiviteten som virkelighetsnær, om de ser på programmering i fysikkfaget som nyttig, om de syntes aktiviteten var gøy og interessant, og hvordan de omtaler aktiviteten som variert i forhold til annen undervisning. Hensikten med kategorien er å identifisere virkningen av programmeringsaktiviteten på elevenes motivasjon. De fleste tilfellene hvor kodene er brukt er eksplisitte utsagn, for

eksempel om variasjon eller nytteverdi. Koden «eierskap» er brukt på flere implisitte utsagn, og er mer avhengig av forskerens tolkning. Fordi eierskap og motivasjon er tett knyttet sammen, var det naturlig å sortere eierskap i motivasjonskategorien. Siden eierskap ble sett på som et spissere og mer interessant konstrukt, vil hovedfokuset i presentasjonen av resultatene være på dette, heller enn på motivasjonskategorien som helhet.

Kategorien «Utfordring» er brukt på tilfeller hvor elevene uttrykker hva som var vanskelig i arbeidet med programmeringsaktiviteten. Kategorien består av fire koder: Fagspesifikke utfordringer, utfordringer knyttet til programmering, utfordringer knyttet til modellering & simulering, og kognitiv belastning. De første tre kodene er hentet fra Basu et al. (2016) (som beskrevet i teorikapittelet), mens den siste koden er brukt for å favne uttalelser som «det var veldig mye å tenke på». Hensikten med denne kodekategorien er å identifisere hva slags utfordringer elevene støter på, og når de støter på dem. I likhet med Basu et al. (2016), fant jeg det krevende å skille konsekvent mellom de tre kategoriene. Kodene er gjennomgått flere ganger for å etterstrebe en så klar inndeling som mulig, men det er også her mulig å stille spørsmål ved min tolkning.

Kategorien «Læring» beskriver hva elevene uttrykker at de har lært, og består av fysikk (som i begreper og konsepter, for eksempel hvordan *k* påvirker luftmotstand), programmering (som i syntaks og kommandoer), og syn på fysikk (syn på hva fysikk er og hva fysikere gjør). Hensikten med kodekategorien er å finne ut av hva elevene lærer av opplegget. De fleste tilfellene hvor kodene er brukt er elevenes eksplisitte uttalelser, men koden er også brukt dersom en elev for eksempel bruker et begrep som ble introdusert i undervisningsopplegget. Det er vanskelig å bevise læring, og jeg erkjenner at min tolkning ikke nødvendigvis er den eneste mulige eller riktige der koden er brukt på implisitte uttrykk for læring.

I tillegg er det en kategori som heter «Diverse». Her har jeg samlet koder som er brukt i analysen for å belyse de store kategoriene og sammenhengene mellom dem. Felles for kodene er at de ikke nødvendigvis passer inn i én kategori, men henger sammen med flere. Noen av kodene har vært viktige for å komme frem til enkelte resultater, noen er brukt for å undersøke enkelte ideer som har dukket opp underveis, mens andre er brukt for å navigere datamaterialet, og fungerer som «filter» for andre koder. Jevnt over er disse kodene ikke grunnet i teoretiske begreper, men ligger nærmere ordlyden i transkripsjonene.

Teoribaserte observasjonsnotater av Captura-opptak

Underveis i prosessen med å kode intervjuene, ble det klart at det ville være enkelte spesielt interessante ting å se etter i Captura-opptakene. Disse henger tett sammen med kodekategoriene fra intervjuene. For eksempel kan måten elever bruker representasjoner på være vel så tydelig når man observerer dem som den er når man snakker med dem. Det samme gjelder typen utfordringer som elevene støter på, som kanskje er enda tydeligere når man kan observere dem underveis. I tillegg er det interessant å se om det dukker opp indikatorer på læring underveis i opplegget, som kan supplere det elevene selv sier om sin egen læring.

Selv om mange av temaene og kodene går igjen i arbeidet med Captura-opptakene, er det ikke sånn at opptakene er kodet systematisk. Arbeidet med opptakene er gjort i tre omganger. Først ble alle opptakene gjennomgått på dobbel hastighet for å identifisere interessante øyeblikk. Deretter, etter at mye arbeid var gjort på intervjuene, kom jeg tilbake til opptakene og hørte dem på nytt, denne gangen på normal hastighet, men med enkelte «hopp» over uinteressante sekvenser. Målet med denne andre runden var å beskrive elevenes arbeid med programmeringsaktiviteten ved hjelp av teoretiske begreper, både for å belyse resultater fra intervjuene, men også for å finne helt nye resultater.

Når man skal se etter teoretiske konstrukter i observasjonsdata, er det viktig med en god operasjonalisering (Judd, Smith & Kidder, 1991). Derfor følger nå en oversikt over hvilke konstrukter som er brukt i arbeidet med observasjonsdataene, og hvordan de er operasjonalisert og identifisert. Oversikten er delt opp i de overordnede temaene jeg har tatt for meg.

Utfordringer. For å studere elevenes utfordringer, har jeg brukt inndelingen til Basu et al. (2016) – fagspesifikke utfordringer, utfordringer knyttet til programmering, og utfordringer knyttet til modellering og simulering. Jeg har valgt å kalle noe en utfordring dersom elevene bruker mer enn ett forsøk på å gjøre noe riktig, eller om de spør læreren eller medelever om hjelp.

Representasjoner. Elevenes representasjonsbruk er beskrevet med bakgrunn i DeFT-rammeverket. På samme måte som i intervjuet, er bruken av begrepene *utfyllende*, *begrensende* og *konstruerende* preget av tolkning. Jeg har valgt å si at representasjoner utfyller hverandre når elevene bruker forskjellige representasjoner til å få ulik informasjon. Jeg har valgt å si at en representasjon begrenser en annen dersom elevene virker å bruke den

ene representasjonen som en hjelp til å tolke den andre. Jeg har valgt å si at representasjoner virker konstruerende når elevene jobber aktivt med sammenhengen mellom to representasjoner. Jeg erkjenner at begrepene til dels er brukt overlappende, og at det er vanskelig å gjøre en klar inndeling.

Tabell 6: *Indikatorer på dybdelæring med tilhørende operasjonalisering, slik de er brukt i analysen av Captura-opptak. Inspirert av Chin og Brown (2000) og Taub et al. (2015).*

Indikator på dybdelæring	**Operasjonalisering**
Elever bruker ideer fra tidligere	Når elever uttrykker forkunnskaper ved å f.eks. gjengi formler og fakta eller referere til en hverdagssituasjon.
Elever presenteres for nye ideer	Elever blir utsatt for nye ideer fra programmet, oppgavesettet, lærer eller medelev, og gir uttrykk for at dette er overraskende, nytt, uventet, uforståelig e.l.
Elever setter sine egne hypoteser	Når elever kommer med forslag til hvilke resultater de får, før de kjører koden eller gjør en utregning.
Elever evaluerer nye ideer og egne hypoteser	Elever vurderer nye ideer som enten gyldige eller ugyldige, gjerne med begrunnelse. Elever velger strategiske metoder for å teste sine egne hypoteser.
Elever sorterer og reflekterer	Elever uttrykker fysikkforståelse ved å organisere, tolke og forklare nye og gamle ideer. Særlig ser jeg etter: - forklaringer som er teoribaserte og spontane - spørsmål om årsak/virkning, og sammenhenger mellom gammel og ny kunnskap (kan også være hypoteser) - diskusjoner med engasjement og på et høyt konseptuelt nivå

Læringsprosesser og dybdelæring. Captura-opptakene gir ikke umiddelbare indikatorer på hva elever lærer. I stedet har jeg valgt å bruke indikatorer fra KI-rammeverket (Taub et al., 2015) og indikatorer på dybdelæring (Chin & Brown, 2000) for å se på hvor læring skjer. Indikatorene jeg ser etter og den tilhørende operasjonaliseringen er gitt i tabell 6. Operasjonaliseringen er et forsøk på å sammenfatte de to lignende rammeverkene i ett analytisk verktøy.

5.3 Etiske vurderinger

I arbeidet med masteroppgaven har jeg forholdt meg til sentrale forskningsetiske prinsipper. Det metodiske opplegget er også godkjent av NSD, Norsk senter for forskningsdata. Særlig har jeg et ansvar overfor elevene i de to klassene som har vært med i prosjektet. Alle elevene har blitt informert både skriftlig og muntlig om hva opplegget går ut på, og alle har samtykket skriftlig. Informasjons- og samtykkeskriv ligger vedlagt i vedlegg D. For elevene er det noen ting som veier tungt i et etisk perspektiv. For det første er anonymitet viktig (Befring, 2015). Lydopptakene av elever er ikke delt med noen. Innad i forskningsgruppen er det kun delt anonymiserte transkripsjoner og notater, og det er ikke mulig for andre enn meg å vite hvilke transkripsjoner som kommer fra hvilke klasser. Det er også et poeng at deltakelse i forskningsprosjektet ikke skal ha noe å si for lærerens vurdering av elevene. Lærerne har tilgang til elevenes skriftlige besvarelser og kan bruke disse som de vil (i likhet med ting de samler inn i alle andre timer), men de har ikke tilgang til transkriberte sitater fra hverken intervju eller skjermopptak. Dette gjør at elevene kan være med i prosjektet uten risiko for å hverken bli gjenkjent eller vurdert på bakgrunn av hva de sier om undervisningsopplegget.

All denne informasjonen står i skrivet til elevene. Elevene fikk god tid til å lese gjennom skrivet i timen, og anledning til å stille spørsmål, før jeg ba de som ville være med om å skrive under. Deretter ble alle skrivene samlet inn. Lærerne gikk gjennom bunken, og kunne bekrefte at alle elevene var med. Ved å samle inn alle skrivene, gjorde jeg det lettere for elever å si nei uten å skille seg ut. Det ble også understreket at elever kunne snakke med enten meg eller læreren sin dersom de ønsket å trekke seg. Selv om jeg understreket at ingen måtte være med, prøvde jeg også å tydelig formidle at det ikke ville være farlig eller risikofylt for elevene å skrive under. Jeg argumenterer dermed for at elevene ga sitt samtykke frivillig og informert, uten tvang. Informasjonen om samtykke og hensikt ble gjentatt muntlig for elevene som stilte til intervju, og elevene som deltok i skjerm- og lydopptak.

En annen etisk vurdering har å gjøre med forskningens troverdighet og kvalitet. Det er viktig at det som blir gjort av forskning er godt håndverk og mulig å stole på, for alle som leser den. Hele forskersamfunnet er sammen med på å bygge opp forskningens troverdighet utad, og alle som publiserer forskning er ansvarlig for at det de publiserer er sant (Befring, 2015). Jeg vil derfor poengtere noen mulige svakheter ved forskningsdesignet og gjennomføringen, samt strategier jeg har brukt for å øke resultatenes troverdighet.

5.4 Troverdighet og gyldighet

Siden dette prosjektet er utelukkende av kvalitativ art, er det ikke hensiktsmessig å operere med begrepene «validitet» og «reliabilitet», som ofte brukes om gyldighet og generaliserbarhet i forbindelse med kvantitativ forskning. Fordi utvalget av informanter er lite, er begreper forbundet med statistikkbasert forskning til liten hjelp i bedømmelse av forskningens gyldighet. I stedet opererer jeg med begrepet «troverdighet», som handler om i hvilken grad forskningsresultatene er plausible, sannsynlige og følgelig forsvarlige (Johnson, 2013). Forskningens gyldighet er avhengig av i hvilken grad datamaterialet er korrekt fremstilt, og i hvilken grad analyseprosessen ikke tillegger datamaterialet mening som ikke er der. Siden lydopptak ikke kan deles av personvernhensyn, er det vanskelig for leseren å avgjøre om transkripsjonene er av tilstrekkelig kvalitet. Men jeg har valgt å gjøre intervjutranskripsjonene tilgjengelige ved forespørsel, slik at den som måtte være interessert kan kontrollere at det er sammenheng mellom datamaterialet mitt og resultatene mine. Captura-opptakene er ikke transkribert, men utfyllende beskrivelser er gitt tidlig i resultatdelen, slik at leseren kan sette sitatene som presenteres inn i en større kontekst.

Når det gjelder generaliserbarhet, er ikke denne studien umiddelbart generaliserbar til andre skoler og klasserom. Dette er heller ikke målet. Målet med oppgaven er å peke på muligheter og begrensninger ved programmering i fysikkfaget, og disse vil etter all sannsynlighet se noe forskjellige ut i forskjellige klasserom. Når det er sagt, så vil resultatene til en viss grad være overførbare til andre, lignende settinger (Johnson, 2013). Jeg håper at utvalget og metoden min er forklart slik at leseren selv kan gjøre seg opp en mening om hvorvidt forskningen som her presenteres er overførbar til den sammenhengen vedkommende befinner seg i. I tillegg håper jeg at denne studien kan sees sammen med andre studier, og på sikt være del av et bredere empirisk grunnlag som sier noe om programmering i fysikkfaget, kanskje særlig i den norske skolen (Johnson, 2013).

Før jeg fortsetter vil jeg igjen påpeke den hyppige dialogen jeg har hatt med både veileder, medstudenter og andre forskere fra innland og utland. Disse har alle kommet med verdifulle innspill, og jeg anser derfor forskningsdesignet som godt informert, ikke bare av meg, men av andre. Her følger ulike aspekter som har innvirkning på oppgavens troverdighet.

Det første aspektet med forskningsdesignet mitt som jeg vil peke på, er antallet lærere som var med i timene med programmeringsaktiviteten. I alle timene bidro både jeg, min medstudent og elevenes faglærer, noe som gjør undervisningssituasjonen noe urealistisk. Det er klart at alle elevenes tilbakemeldinger underveis og i etterkant av opplegget er preget av at vi var flere lærere som kunne gi hjelp fort. Det er mulig at elevenes tilbakemeldinger ville vært mindre positive om det kun var én lærer i timene. I tillegg kan det ha hatt påvirkning på elevene at det kommer fremmede lærere inn i klasserommet. Det kan føre til at elevene oppfører seg bedre, jobber hardere eller prøver mer, fordi de vil være snille mot en ny lærer. Lærerne uttrykte ikke at elevene oppførte seg annerledes enn vanlig, og denne effekten er sannsynligvis liten. Dette er beslektet med det som gjerne kalles *observatøreffekten* (Kleven, Hjardemaal & Tveit, 2011), at mennesker oppfører seg annerledes når de blir observert. Særlig er det relevant for elevene som deltar i skjerm- og lydopptak. Det er ikke sikkert at elevenes interaksjon med hverandre foran mikrofonen foregår slik den ville foregått uten mikrofon, men det er vanskelig å si om elevene blir mer reserverte eller mer snakkesalige. Etter å ha sett opptakene kan det virke som at elevene fort glemmer at mikrofonen er der. For eksempel snakker de fritt om planer for helga, fritidsaktiviteter også videre.

Det er mange ting som kan være med på å påvirke informasjonen man får i et fokusgruppeintervju. Det kan være at elevene i gruppa ikke er trygge på hverandre, noe som fort fører til at noen elever ikke tør å ytre sine perspektiver (Krueger, 2014). Særlig kan det forekomme dersom de synes noe har vært vanskelig, siden de ikke ønsker å fremstå som «dumme» foran sine medelever. Det kan også være at noen elever uttaler seg mer positivt om både egne prestasjoner og opplegget for øvrig enn det de ville gjort utenfor intervjusituasjonen (Krueger, 2014). Dette er vanskelig å kontrollere, men der det er mulig – for eksempel når elevene snakker om hva de synes var vanskelig – har jeg sammenlignet resultatene fra intervjuene med resultatene fra skjermopptakene, for å få et bredere bilde.

Å bruke flere metoder for å belyse det samme forskningsspørsmålet, kalles gjerne *metodetriangulering* (Johnson, 2013). Målet er at elevenes skriftlige besvarelser, intervjuer

og skjerm- og lydopptak skal veie opp for hverandres svakheter, slik at datamaterialet er så helhetlig og nyansert som mulig.

I analyseprosessen er det alltid fare for at forskerens *bias* har en påvirkning på datamaterialet, både bevisst og ubevisst (Creswell & Miller, 2000). Jeg oppdaget tidlig i analysen at jeg hadde troa på undervisningsopplegget jeg hadde laget, og at jeg ville at det skulle fungere. Derfor har jeg underveis tatt noen grep, slik at jeg ikke har funnet bare det jeg ville finne. Først og fremst er analysen gjort i tett dialog med mine veiledere, såkalt *forskertriangulering* (Johnson, 2013). De har kodet deler av intervjumaterialet for å teste kodesettet, og har fått innblikk i nærmest alt jeg har gjort av analyse. For det andre har jeg prøvd å lete aktivt etter det som *ikke* fungerte med undervisningsopplegget, og eventuelle grunner til at programmering *ikke* skulle være en god idé i skolen. Beskrivelsen av analysen og resultatene er ganske grundig, med det mål at leseren skal forstå hvordan og hvorfor jeg trekker de konklusjonene jeg gjør. All kvalitativ analyse er tolkning, og det kan hende min tolkning er annerledes enn noen andre sin, men jeg påstår – i lys av de grepene som er gjort – at analysen er gjennomført på troverdig og hederlig vis, og at det er belegg for de resultatene jeg presenterer.

6. Resultater

Her presenteres resultatene fra analysen av datamaterialet. Som beskrevet i analysekapittelet resulterte kodeprosessen i 20 koder sortert i fem overordnede kategorier: DeFT, Representasjoner, Utfordringer, Motivasjon og Læring. I tillegg er det en kategori kalt «Diverse», med 7 koder. Jeg gjentar også – for ordens skyld – at under kategorien om motivasjon, er det *eierskap* som er det fremtredende fokuset. En enkel oversikt over kodene og kategoriene er gitt i tabell 7, og en detaljert oversikt er gitt i vedlegg A.

Først av alt presenteres de fire Captura-opptakene, slik at leseren kan få et innblikk i hvordan programmeringsaktiviteten artet seg. Deretter følger resultater som er relevante for problemstillingen, delt opp i fire ulike kategorier: Læring, utfordringer, representasjoner og motivasjon. I referansene til datamaterialet, bruker jeg forkortelsene oppgitt i tabell 8.

Tabell 7: *Endelig kodesett etter tematisk analyse, fordelt i seks kategorier.*

Kategori	Representasjoner	DeFT	Motivasjon
Koder	Formler & ligninger Kode Animasjon Graf Forsøk	Utfyllende Begrensende Konstruerende	Eierskap Virkelighetsnært Nyttig Gøy & interessant Variasjon

Kategori	Utfordringer	Læring	Diverse
Koder	Fagspesifikke Modellering & simulering Programmering Kognitiv belastning	Fysikk Programmering Syn på fysikk	Luftmotstand Parprogrammering Servert kode Tekst Tenke fysikk Undervisningsopplegg Utforskning

Tabell 8: *Forkortelser for referering til ulike datakilder.*

Datakilde	**Forkortelse**	**Eksempler**
Skriftlige elevsvar	ES klassenummer:elevnummer	ES 1:03 gir svar nummer tre fra den første klassen. ES 2:05 gir svar nummer fem fra den andre klassen.
Intervjutranskripsjoner	I intervjunummer:sidetall	I 1:9 gir side nummer ni i intervju nr. én. I 4:2 gir side nummer to i intervju nr. fire.
Observasjonsnotater fra Captura-opptak	C gruppenummer:timenummer	C 1:1 gir opptak av den første gruppens første time. C 3:2 gir opptak av den tredje gruppens andre time. C 1 og C 2 er hentet fra samme klasserom, og C 3 og C 4 er hentet fra samme klasserom.

6.1 Skjerm- og lydopptak

Opptakene fra klasserommet er en viktig datakilde for å finne ut av hvordan elevene jobber med programmeringsoppgavene, hvilke funksjoner de ulike representasjonene har, og hva slags utfordringer elevene støter på underveis. Her følger en kort oppsummering av hvordan programmeringsaktiviteten artet seg for de fire gruppene som er tatt opp. Eksempler fra opptakene vil bli brukt i den videre presentasjonen av resultatene. Enkelte av elevene som er tatt opp er også med på intervjuer, de er da gitt med samme navn i begge tilfeller. Det ble i skriveprosessen vurdert om alle de fire parene skulle presenteres. Fordi det er vanskelig å presentere disse resultatene helhetlig på en annen måte, særlig når de ikke er analysert systematisk, har jeg valgt å gi beskrivelser av samtlige opptak. Opptakene viser fire ganske forskjellige gjennomføringer av opplegget, og anses derfor som relevante for å gi leseren et innblikk i ulike aspekter ved programmeringsoppgavene, samt mulighet til å sette presenterte sitater inn i en større sammenheng.

Øyvind og August (C 1:1, C 1:2)

I det første opptaket møter vi de to elevene Øyvind og August. Øyvind har tilsynelatende erfaring med programmering fra før, men i et annet språk og en annen setting. Dette kommer til uttrykk blant annet når han bruker notasjon som ikke har blitt undervist (f.eks. v += a*dt i stedet for v = v + a*dt). Han virker å ha svært høy selvtillit og mestringsforventning, og går raskt til verks med oppgavene. August er stort sett stille, og bidrar med få interessante innspill.

Allerede tidlig i aktiviteten støter Øyvind og August på problemer. De har glemt at fartsøkning er a*dt, og skriver bare v = v + a. Tilsvarende gjør de for posisjonen. De to elevene bruker store deler av de første 40 minuttene på å finne ut av dette, uten å spørre om hjelp. Denne prosessen er i stor grad preget av tilsynelatende tilfeldig prøving og feiling, og det er tydelige utfordringer knyttet til både modelleringsaspektet og det syntaktiske programmeringsaspektet. Elevene bruker masse tid på å endre initialbetingelser og parametere for å få animasjonen til å gi mening, men kun én gang går de systematisk gjennom while-løkken sin, hvor feilen faktisk ligger. Dessverre ser de ikke at de har gjort feil, og går tilbake til initialbetingelsene.

Øyvind og August bruker animasjonen som pekepinn på om de har gjort rett. Mot slutten av den første timen ser animasjonen ut til å gi mening. Men elevene har hatt flaks – startfarten er 0,001m/s, tyngdeakselerasjonen er -0,005m/s^2, og tidssteget er altfor stort, satt til dt=1,0s. Animasjonen virker å gi elevene et inntrykk av at de har rett, helt til Øyvind kommenterer at «verdiene er jo helt kaos». De går litt videre i oppgavesettet og får opp posisjonsgrafen, men sliter med å forstå hva aksene betyr. Til slutt ber de læreren om hjelp. Han bruker god tid på å forklare dem hva som skjer i løkken, og hvordan kodelinjene henger sammen med bevegelsesligninger for konstant akselerasjon og konstant fart. Kort tid etter er timen slutt. Da har elevene rett kode i løkken, men initialbetingelsene er fortsatt «kaos».

I den neste økten fortsetter Øyvind og August å jobbe. Det første interessante som skjer, er at de bytter ut uttrykket de har for posisjonen (som var rett) med et uttrykk de kjenner fra før – den fulle bevegelsesligningen for konstant akselerasjon. I utgangspunktet går dette fint, men i omskrivningen blir *dt* byttet ut med *t*, som gir langvarige problemer. Nok en gang går det mye tid til prøving og feiling og justering av initialbetingelser, før de til slutt får hjelp av læreren og en kode som kjører rett.

Når elevene skal finne et uttrykk for luftmotstanden, sier Øyvind: «irriterer meg at vi ikke har blyant.» I stedet for å skrive ned formlene på papir, skriver elevene de tilsvarende formlene inn i løkken – hver kraft regnes ut for seg, før de legges sammen, og så deles kraftsummen på kulens masse, for å få akselerasjonen. Dette er tilsynelatende den eneste gruppen som ikke skriver kreftene delt på massen direkte inn for akselerasjonen. Elevene bruker litt tid på å overbevise seg selv om at de har gjort rett, før de fortsetter med oppgavene. Herfra og ut går det ganske knirkefritt, og det siste som skjer før timen avsluttes er at elevene kjører for en høyere startfart, både med og uten luftmotstand. Når de ser hvor mye høyere kulen kommer når vi ser bort i fra luftmotstanden, uttrykker Øyvind: «Hjelpes!»

Johanne og Linda (C 2:1, C 2:2)

Johanne og Linda sin første økt kan nesten beskrives som et «best case scenario». Elevene jobber godt og selvstendig hele økten, med få henvendelser til læreren. Dette opptaket skiller seg ut fra de andre, av to grunner. For det første er Johanne og Linda sin arbeidsmåte mer målrettet og systematisk enn de andre elevene som er tatt opp. For det andre bruker de mye mer tid på å diskutere fysikken i det som skjer, og utviklingen av fysikkforståelse kommer til syne i større grad.

Elevene kommer raskt i gang, og har etter kort tid riktig kode, med litt hjelp fra læreren. Den første utfordringen de bruker tid på, er å finne ut hva akselerasjonen skal være når man ser bort fra luftmotstand. De to bruker en del tid på å diskutere akselerasjonens retning i ulike deler av bevegelsen, om den er konstant eller ikke, hvordan farten endrer seg også videre. De trenger litt hjelp fra læreren for å skjønne at akselerasjonen må være g. Den neste interessante diskusjonen dukker opp når elevene sammenligner resultatene for forskjellige tidssteg. Etter å ha kjørt med dt=0,1, antar de at «jo mindre tidssteg, jo høyere går kulen». Men når de kjører med dt = 0,001 og ser at de får det samme resultatet som den teoretiske verdien, diskuterer de påstanden sin i lys av lærerens tidligere undervisning, og fra forsøk de har gjort tidligere i Tracker[8]. Videoen i Tracker blir mer nøyaktig når man har flere bilder pr. sekund, og elevene bruker dette til å konkludere med at jo mindre tidssteget er, desto mer nøyaktig blir simuleringen.

Underveis i hele den første økten ser vi at elevene foreslår hypoteser, og så varierer initialbetingelser og parametere strategisk for å teste hypotesene sine. Vi ser også at de stadig

[8] Tracker er et verktøy for å analysere bevegelser på film. Programmet lar deg blant annet spore et objekt fra bilde til bilde, og gjøre en regresjon av posisjonspunktene. For mer informasjon, se https://physlets.org/tracker/

kobler den fysiske intuisjonen sin og ting de har lært før med resultatene fra programmet, og prøver å få de to til å henge sammen. Elevene skal akkurat til å begynne med luftmotstand når den første økten er slutt. Før opptaket skrus av, sier elevene: «det var gøy dette her. Jeg synes vi var flinke.»

De første tjue minuttene av den neste økten går med til å finne ut av hva uttrykket for akselerasjonen skal være når luftmotstanden er med. Elevene diskuterer både seg imellom og med lærer, og kommer til slutt frem til riktig svar. I prosessen kommer det fram at elevene finner det utfordrende å jobbe med Newtons andre lov, og virker noe usikre når de diskuterer hvilke krefter som virker på kulen. Johanne og Linda finner faktisk uttrykket for akselerasjonen på egen hånd, ved hjelp av Newtons andre lov, men de blir avskrekket av at «det er så mye», og tror det er feil. Derfor spør de læreren om hjelp, og kommer frem til det samme uttrykket.

Når elevene skal kjøre programmet, viser det seg at de har åpnet feil programkode. Læreren oppdager det når han går forbi, og hjelper elevene med å bytte til riktig kodevindu. Når de skal kopiere akselerasjonsuttrykket sitt fra den første koden til den andre, mister de konstanten g på veien, og får derfor noen utfordringer når de kjører programmet – kulen kommer ikke ned igjen. Nok en gang spør de om hjelp, og drøyt 30 minutter inn i økten er de klare til å fortsette.

Johanne og Linda fortsetter i samme spor, og gjør ferdig oppgavene. Dessverre kjører de ikke programmet med høy startfart både med og uten luftmotstand, så de mister aha-opplevelsen vi ser både hos Øyvind og Diana sine grupper. De siste tjue minuttene av timen bruker de på ekstraoppgave tre, hvor de skal modellere fjærkanonen som skyter kulen opp. De følger instruksjonene og klarer seg ganske bra, og får til å lage en kanon med konstant akselerasjon som skyter kulen opp til den riktige høyden før timen slutter. Elevene er tydelig fornøyde med egen innsats når økten avsluttes.

Diana og Hedwig (C 3:1, C 3:2)

Diana og Hedwig møter programmeringsaktiviteten uten å ha sett noe programmering før. I den første økten får elevene mye støtte fra læreren, men de jobber godt og kommer ganske langt. Den oppgaven som tar lengst tid, er når de skal skrive riktig kode i while-løkken, for å simulere bevegelse. Særlig bruker de tid på å identifisere hva akselerasjonen skal være, også før luftmotstanden introduseres. Læreren bruker tid på å forklare dem hvordan de kan bruke Newtons andre lov, og hvordan løkken fungerer. Det kan virke som at elevene har liten

forståelse for hva som skjer i hvert tidssteg. På et tidspunkt skriver læreren inn en kodelinje for elevene, før han forklarer hvorfor han skrev det han skrev. Da virker det som at elevene i større grad forstår.

De to jentene leser oppgaveteksten nøye og sakte, før de tar fatt på én oppgave av gangen. Mens andre elever viser tegn til å ville eksperimentere og prøve seg frem, er det slående at Diana og Hedwig ikke har kjørt koden en eneste gang før de har holdt på veldig lenge, og nærmest skrevet ferdig programmet sitt.

Underveis mens de jobber, virker det som at fokuset sakte flytter seg fra animasjonen til posisjonsgrafen, i takt med at elevene blir tryggere på programmet og kulens bevegelse. Når den første økten er slutt, har elevene gjort ferdig oppgavene uten luftmotstand, og har akkurat begynt å diskutere luftmotstandens retning.

I begynnelsen av den neste timen har Diana og Hedwig et program som kjører, og de er klare for å legge inn luftmotstanden. Uheldigvis er initialbetingelsene fortsatt justert fra de siste oppgavene i forrige økt – gravitasjonen er som på månen, og startfarten er veldig høy. Dette gjør at elevene til slutt ender opp med å ha funnet feil verdi for luftmotstandskoeffisienten. Det er ingen krise i seg selv, men det er interessant å se at elevene aldri reagerer på at kulen kommer veldig høyt opp, heller ikke når de skal sammenligne med forsøket de har gjort tidligere.

Begynnelsen av den andre økten går med til å finne et uttrykk for kulens akselerasjon, når luftmotstanden er tatt i betraktning. Her lener elevene seg i stor grad på støtte fra læreren, og har etter ganske kort tid (ca. 15 minutter) et program som virker. Resten av økten jobber Diana og Hedwig jevnt og selvstendig, og kommer i land med alle oppgavene, til tross for noe redusert konsentrasjon og tempo mot slutten. Det siste som skjer er at de to elevene kjører koden med høy startfart, både med og uten luftmotstand. De blir tydelig overrasket over at luftmotstanden reduserer kulens maksimale høyde fra 500m til 2m.

Arne og Berit (C 4:1, C 4:2)

Arne og Berit går løs på programmeringsaktiviteten med friskt mot. Det tar ikke mange minutter før de har skrevet ned korrekte initialbetingelser og riktige beregninger i løkken. I første omgang mangler de bare å skrive at akselerasjonen er gitt ved g, men dette tenker de seg også frem til etter hvert.

Problemet til Arne og Berit er at alle beregningene i løkken er skrevet uten innrykk, og dermed virker ikke koden. I tillegg har de også et par andre syntaksfeil som skaper problemer. Men de to elevene forstår ikke hva som er feil, og begynner å endre på koden sin for å finne noe som fungerer. De sitter lenge og strever, og bytter ut mye av det som var helt riktig, med andre ting som er helt feil. Etter en stund er de tydelig frustrerte, men fortsetter å prøve uten å spørre om hjelp. Når programmet gir feilmeldinger, virker det som at Arne og Berit ikke leser dem, og de går lite systematisk til verks når de skal prøve å finne ut av hva som er feil.

Til slutt kommer en lærer bort og hjelper dem med å endre koden tilbake, men heller ikke læreren ser umiddelbart at det mangler innrykk inne i løkken. Han ser det til slutt, og elevene får endelig til å simulere det vertikale kastet uten luftmotstand.

Elevene går videre til neste oppgave, hvor de skal sammenligne resultatet sitt med det teoretiske resultatet for bevart mekanisk energi. I motsetning til mange andre, går de løs på oppgaven ved å skrive i kodevinduet, og prøver å få datamaskinen til å regne ut det teoretiske resultatet. Først prøver de å skrive opp ligningen for bevart mekanisk energi, og det virker som de håper at programmet skal løse ligningen for dem – sånn som i Geogebra og CAS. Dette fungerer ikke, men etter litt om og men skriver de ned uttrykket for den teoretiske høyden, som er riktig. Nok en gang lider elevene av syntaksfeil, og de får et svar som er helt feil, og som de skjønner er helt feil. De ender opp med å måtte spørre læreren om hjelp, før de skriver regnestykket inn på kalkulator og får riktig svar.

Resten av den første økten bruker elevene på å gjøre seg ferdig med oppgavene uten luftmotstand. I den neste økten finner de fort ut at akselerasjonen er kraft delt på masse, og de skriver inn korrekt uttrykk for akselerasjonen som skyldes luftmotstand. Men de glemmer at gravitasjonen fortsatt virker, og de strever lenge med å forstå hvorfor ballen aldri stopper opp, men fortsetter å gå oppover og oppover. Til slutt kommer læreren bort og minner dem på at gravitasjonen virker. Her kommer det frem at Berit synes det er utfordrende å forholde seg til to krefter – hun uttaler blant annet at luftmotstandsuttrykket «blir g».

Etter at Arne og Berit har fått riktig uttrykk for akselerasjonen, fortsetter de med oppgavene. De bruker lang tid på oppgaven hvor de skal sammenligne med fjærkanonforsøket for å finne k, mest fordi de har glemt å endre initialbetingelsene tilbake, så de har for lav startfart. Dessverre sammenligner de ikke med resultatet fra forsøket, men bruker i stedet resultatet fra simuleringen de nettopp har gjort, uten luftmotstand. Derfor ender de opp med en veldig lav

verdi for *k*, som i praksis er lik null. Videre fører dette til at de ikke ser noen nevneverdig forskjell på simuleringen med og uten luftmotstand i den siste oppgaven.

Det siste kvarteret av timen bruker Arne og Berit på ekstraoppgave to. De kommer godt i gang, men blir ikke ferdige, og får heller ikke en simulering som kjører som den skal.

6.2 Hva lærer elevene?

Elevene sier de lærer noe både om fysikk og noe om programmering

Som et utgangspunkt for videre diskusjon av hvorvidt elevene har opplevd dybdelæring, er det aktuelt å se på hva de selv sier at de har lært. I datamaterialet utkrystalliserer det seg tre tydelige kategorier. Elevene lærer noe om fagspesifikke begreper og konsepter («fysikk», 34 tilfeller), programmeringstekniske begreper og konsepter («programmering», 38 tilfeller), og noen elever uttrykker at de har lært noe om hva fysikk er og hvordan det praktiseres («syn på fysikk», 17 tilfeller). I tillegg er 20 tilfeller av læring kodet som «annet».

Av de fagspesifikke tingene elevene har lært om, er det forståelsen av luftmotstand som markerer seg. Diverse utdrag fra datamaterialet er presentert her:

> Carina: Også luftmotstand. For jeg husket ikke formelen, men ...
>
> **I:** Sitter den?
>
> Carina/Frida: (*I kor*) Ja! (I 1:3)

> Vi har fått bedre forståelse av formlene, konstanter som K og hvordan endringen av den påvirker luftmotstanden. Når man ser animasjonen er det lettere å forstå hvordan alt henger sammen. (ES 2:13)

> Lina: Vi visste at det på en måte, at det gir motstand, men ikke kanskje så mye som …
>
> Gunilla: At det hadde så mye å si.
>
> Elever: *(om hverandre)* Nei, ja, mhm, ja.. ja. (I 2:4-5)

> Mikah: Mhm. Det var liksom ... jeg synes det var interessant at hvis du hadde på luftmotstand så kunne du øke farten med hundre, også økte bare høyden med tjue eller noe sånn da. (I 3:4)

Elevene uttrykker altså at de har lært noe om hvordan formelen for luftmotstand ser ut, og hvordan den fungerer. I tillegg har det gjort inntrykk på elevene at luftmotstanden har så stor effekt som den har for høye hastigheter. Det er verdt å nevne at mens mange elever fikk oppleve den store effekten luftmotstanden har på ballens bevegelse (basert på intervjuene og erfaringer fra klasserommet), ser vi av Captura-opptakene at enkelte grupper ikke kjørte koden med høy startfart. Det kan sies at de derfor gikk glipp av en positiv læringsopplevelse.

Den andre tingen mange elever uttrykker at de har lært noe om, er programmering. Dette kommer blant annet frem når de intervjuede elevene blir spurt om å forklare hva som skjer i en programkode. Alle gruppene klarer å forutse den rette bevegelsen. Det er også programmeringsaspektet de fleste elevene trekker frem i den skriftlige besvarelsen:

> Jeg har lært å programmere grunnleggende regning og animasjoner i Python og å forstå hvordan datamaskinen tolker fysiske formler. (ES 1:01)

> Blant annet har vi lært hvordan man kan regne ved hjelp av programmering som er en veldig effektiv måte å løse oppgaver på. Vi har lært hvordan vi skriver inn ulike kommandoer for å gjøre det vi vil at skal skje og forandre på kommandoene hvis vi skulle trenge det. Vi lært (*sic*) hvordan vi kan lage figurer så vi kan få en bedre oversikt over hva vi prøver å finne ut av og hva som egentlig skjer. (ES 2:15)

Det er vanskelig å si nøyaktig hvor mye programmering elevene har lært, og om de ville være i stand til å skrive sin egen kode i etterkant av programmeringsaktiviteten. Men det er enkelte intervjusitater som indikerer at elevene har fått med seg noen av de viktige konseptene, selv om de ikke skulle være i stand til å gjenta aktiviteten uten støtte. For eksempel sier Nathaniel:

> Nei ... For min del må det være while-løkka. Interessant hvordan man bare kan sette opp én liten skrift, også gjør den det tusen ganger, uten at du trenger å skrive det om og om igjen. (I 3:3)

Løkker er essensielle for mange av problemstillingene vi programmerer i fysikken, og det virker som elevene har fått med seg hvordan de fungerer. Et annet konsept som er svært viktig, men også krevende å forstå, er oppdateringen av variabler inne i løkker. Også her

uttrykker elevene en viss grad av forståelse, og at de har plukket opp mye av det som ble sagt av læreren i timen:

> Ragnhild: Ja, bare, jeg husker bare at du, det du sa var at på venstre siden, altså, for eksempel v da, det er på en måte den gamle verdien, på en måte. Og på høyre, så har du på en måte, har du den oppdaterte verdien, på en måte. Så den, når på en måte while-løkka, liksom, eh, liksom kommer fremover, så er det sånn at den henter verdien, henter den nye verdien og på en måte setter den inn i det andre, hvis det gir mening.
>
> Sunniva: Ja, men, var det ikke motsatt, at ...
>
> Ragnhild: Kanskje det var det, ja.
>
> Sunniva: At den på høyre siden er den gamle, og den på venstre ... er …
>
> Nathaniel: Mhm
>
> Ragnhild: Åja, det var kanskje det det var ja
>
> Sunniva: Men det du sa etterpå var jo riktig da.
>
> Ragnhild: Ja, det er sant. (I 3:7)

Til sist kan det virke som at opplegget til en viss grad påvirker elevenes syn på fysikkfaget mer generelt:

> Frida: Man så jo litt hva man på en måte kan drive med innen fysikk da, at man kan programmere og sånne ting. (I 1:19)
>
> Arne: Det er nytt da. Nytt, i forhold til hva man tenker fysikk er da, at man tenker det er skriving og pugginga og formler, men så er det liksom, kanskje også litt praktisk da. (I 1:23)
>
> Åshild: Jeg synes det er veldig spennende å se at ... fysikk både er ... veldig mye av det er fra ganske lenge siden. Men at det fortsatt da kan være relevant med nye ting som kommer hele tida, jo lenger vi kommer inn i fremtiden da. Med koding, som er nytt, at man da hele tida kan knytte det opp til nye temaer. Og bruke det mer. (I 4:15)

Dersom vi sammenligner med læringsmålene for undervisningsopplegget presentert i kapittel 4.2, ser vi at elevene til en viss grad oppfyller målene som er satt (selv om det er vanskelig å avgjøre i hvilken grad uten å bruke gjennomarbeidede tester). At elevene helt klart opplever at de har lært noe, betyr også at det kan være hensiktsmessig å diskutere dybdelæring i lys av datamaterialet.

Elevene tar del i prosesser knyttet til algoritmisk tenkning

Erfaringer fra klasserommet, samt Captura-opptakene, forteller oss at elevene tar del i alle arbeidsmåtene som presenteres av Utdanningsdirektoratet (2019a) i forbindelse med algoritmisk tenkning. Elevene utforsker og eksperimenterer («fikler»), og de får til en viss grad lov til å skape sitt eget program. De bruker mye tid på feilsøking (om enn ineffektiv), og de viser generelt god utholdenhetsevne i møte med vanskeligheter. Både Øyvind og August (C 1:1) og Arne og Berit (C 4:1) prøver svært lenge å få programmet sitt til å fungere før de spør læreren om hjelp. Hos de andre gruppene ser vi færre tegn til slik utholdenhet, men disse elevene møter også færre vanskeligheter. Elevene er også flinke til å samarbeide, og viser at de setter pris på det:

> Mikah: Jeg synes det er effektivt, for da kunne man ha en som drev å jobba litt og prøvde å se det ... fordi det var to forskjellige sider av det på en måte da. Det var en sånn, hvor du måtte kunne regne ut litt for eksempel, med fysikk, også var det en som holdt styr på koden, og passa på at alt var bra der.
>
> Sunniva: Det var lett å gå litt sånn i surr, så det var på en måte greit å ha fire øyne på det.
>
> Oliver: Jeg tror det hadde vært vanskelig hvis vi hadde jobbet en og en. Du får liksom ... når du er to og to da, så får du et ... en som kanskje ser på det annerledes enn du gjør da. (I 3:9)

Disse aktivitetene er på mange måter innbakt i designet av undervisningsopplegget, og ikke nødvendigvis svært overraskende. Likevel er det en indikator på at programmeringsaktiviteten fungerer som tenkt, og bidrar til algoritmisk tenkning.

6.3 Hva opplever elevene som utfordrende, og hva slags støtte trenger de?

Datamaterialet viser i sum at elevene har mange av de samme utfordringene som observert av Basu et al. (2016) – fagspesifikke utfordringer, utfordringer knyttet til programmering, og utfordringer knyttet til modellering og simulering. Kategoriene er til dels overlappende, men jeg velger å presentere dem i tre ulike avsnitt, med forbehold om at hva som skal hvor kunne vært bestemt på en annen måte.

Fagspesifikke utfordringer

Når elevene i fokusgruppeintervjuene skal si hva de synes var vanskeligst i opplegget, er det mange som trekker frem det å finne rett uttrykk for akselerasjonen i tilfellet med luftmotstand. Dette passer godt med opplevelsen i klasserommet, der inntrykket var at nesten samtlige elever trengte hjelp med dette i løpet av de to øktene. Elevene sier blant annet:

> Peter: Jeg synes det var vanskelig å finne formelen for akselerasjonen, i en av de siste oppgavene. (I 3:7)
>
> Åshild: Når du skulle regne ut akselerasjonen. Vi måtte gjøre om luftmo ... eller, Newtons andre lov, også bruke luft ... luftmotstanden ... det synes jeg var litt, eller da trengte jeg litt hjelp i hvert fall. (I 4:5)

Det er ikke umiddelbart klart hvorfor elevene synes dette var vanskelig. De har tross alt brukt Newtons andre lov til å finne akselerasjon mange ganger før. Men i Captura-opptakene virker det tydelig som at elevene ikke har «fysikkunnskapen» sin så lett tilgjengelig når de programmerer. For eksempel er det mange som er usikre på hvilke krefter som virker på en kule i fritt fall (C 2:1), eller hvordan man bruker bevaringsloven for mekanisk energi til å regne ut kulens maksimale høyde (C 3:1).

I tillegg til disse utfordringene som mange av elevene har, gir Captura-opptakene innblikk i mange feil som varierer fra gruppe til gruppe. For eksempel virker flere av elevene ustødige på forhold mellom ulike desimaltall: «0,09. Fordi vi må ha noe som er bittelitt større enn 0,001.» (C 2:2), mens andre sliter med å lese av koordinatene på grafen (C 3:1).

Utfordringer knyttet til programmering

Flere av elevene trekker frem det programmeringstekniske som vanskelig. Dette gjør seg hovedsakelig gjeldende på to måter – som syntaktiske utfordringer, og som vanskeligheter med effektiv feilsøking. For eksempel sier Mikah:

Mye parentesbruk også, så er det litt sånn, vanskelig å få kontroll på. (I 3:7)

Dette med parenteser ser vi også tydelig i Captura-opptaket til Arne og Berit, hvor bruk av parenteser og regnerekkefølge hindrer elevene i å komme videre (C 4:1). Noen elever bruker feil syntaks for potens («^» i stedet for «**»), og står fast lenge på grunn av det (C 1:2), mens andre får feilmeldinger fordi de ikke er konsekvente med store og små bokstaver (C 4:1).

Det kommer frem av Captura-opptakene at elevene ikke vet hvor de skal lete etter feilene sine, og de har få verktøy for å evaluere hvilke deler av koden som er riktige, og hvilke som ikke er det. Selv om elevene enkelt kan se på animasjonen om de får den bevegelsen som de forventer, har de vanskelig for å forklare hvorfor animasjonen er som den er. For eksempel bruker Øyvind og August lang tid på å endre initialbetingelser, fordi de ikke klarer å se at det er linjene i while-løkken som er feil. Arne og Berit blir hindret av et innrykk som mangler, og ender opp med å forandre på den riktige koden sin. Utfordringene er både knyttet til at elevene ikke klarer å lese feilmeldingene som kommer fra Trinket (disse er ikke alltid lette å forstå), og at feilene sjelden fører til at programmet gir en feilmelding. Programmet vil for eksempel ikke protestere dersom tyngdeakselerasjonen har feil fortegn, og da vet ikke elevene nødvendigvis hvor de skal lete. Ulrik uttrykker det slik: «[Det] tok litt tid å finne feilene siden det var så mye koding.» (I 3:6)

Utfordringer knyttet til modellering og simulering

Captura-opptakene viser at elevene gjennomgående (men ikke uten unntak) strever med å koble sin fysiske intuisjon og det de har lært tidligere, med det som skjer i programmet. For eksempel sier Berit: «Vi glemte at ballen skulle ned også.» (I 1:12), mens Johanne og Linda spør hverandre «skal ballen komme ned igjen?» (C 2:1) Tilsynelatende har elevene vanskeligheter med å evaluere koden sin, fordi de blir usikre på hva som er rett. Man kunne tenke seg at elevene aktivt brukte laboratorieforsøket med fjærkanon som en «målestokk» for å evaluere sitt eget program, men det ser ut som at elevene tenker ganske lite på forsøket mens de jobber med programmeringen. Dette kommer frem av at elevene snakker lite om forsøket i Captura-opptakene. Vi ser også at når elever evaluerer hvor høyt kulen gikk, bruker de animasjonsvinduet som målestokk, heller enn verdiene fra forsøket. Når kula går ut av vinduet, tenker elevene at den går «alt for høyt», men de nevner hverken den faktiske verdien for kulen sin høyde, eller resultatet fra forsøket (C 1:2).

I forlengelse av dette, ble jeg svært overrasket over elevenes tolkning av oppgaveteksten: «*Klarer du å finne en verdi for* k *som gjør at den maksimale høyden til kula blir den samme som du målte i forsøket om mekanisk energi?»* Se for eksempel hva Mikah sier om saken:

> Mikah: Det var én oppgave. Det var den derre, eksperimentet vi hadde gjort med luftmotstand, så skulle du finne en k-verdi da, og da trodde jeg det vi hadde gjort, det var å bruke den derre formelen i selve simuleringen, og da, da hadde vi jo regna med at det ikke var noe luftmotstand, ikke sant. Så da tenkte jeg bare, "ja okey, k = 0 da, hvis det skal være.. hvis det skal stemme med den, der".

Det Mikah har gjort, er å finne en verdi for luftmotstandskoeffisienten som gir kulen lik maksimal høyde som i tilfellet uten luftmotstand, $k = 0$. Arne og Berit (C 4:2) (og mange flere) har gjort det samme. Da jeg gikk rundt i timen, var det mange elever som hadde $k = 0{,}00001$, uten å skjønne at dette i praksis er null. Her virker det altså som at det er en svikt i oppgaveteksten. Men det er også interessant at elevene ikke reagerer og synes det er rart å skulle reprodusere et resultat de alt har fått. Dette tyder på at elevene ikke har fått klar nok beskjed om hva poenget med opplegget er, i tillegg til at forsøket virker å være nærmest glemt når elevene kommer til dette punktet i oppgaveteksten.

Et av målene med opplegget er at elevene skal få en bedre forståelse for hvilke parametere og initialbetingelser som påvirker kulens bevegelse på hvilke måter. Det virker som at dette er utfordrende for elevene. Det mest åpenbare eksemplet er når Øyvind og August bruker mye av tiden på å endre initialbetingelsene sine for å få rett resultat (C 1:1). Det er for eksempel interessant at de tenker at g kan være noe annet enn $-9{,}81 \text{m/s}^2$, når de tilsynelatende vet godt at dette er tyngdeakselerasjonen. Problematikken blir ekstra tydelig når elevene klarer å få en animasjon som ser riktig ut, men hvor verdiene i kodevinduet er «helt kaos» (C 1:1). Det er også slående at mange elever glemmer å endre initialbetingelsene sine tilbake til verdiene fra forsøket når de skal legge til luftmotstanden. Da har de nettopp justert startfart og tyngdeakselerasjon i tilfellet uten luftmotstand, og flere kommenterer at det ikke ble påpekt i oppgaven at verdiene skulle endres tilbake:

> Gunilla: Noen ting sto ikke såå (*sic*) presist hva du skulle gjøre, sånn som, vi to regna ut nesten halve oppgaven da at gravitasjonen at vi var på månen. Fordi det sto ikke at vi skulle bytte tilbake liksom sånn da, ikke sant. (I 2:7)

Dette kan vise to ting. Enten har elevene vanskelig for å holde styr på hvilke betingelser som hører til hvor, så de ikke husker at tyngdeakselerasjonen er g på jorda. Eller, så viser det at

når de har skrevet koden og begynner å kjøre programmet, så er det begrenset hvor mye de forholder seg til det de allerede har skrevet. Diana og Hedwig (C 3:2) jobber nesten hele den andre økten med feil tyngdeakselerasjon og startfart, og selv når læreren påpeker at de må endre tyngdeakselerasjonen, så endrer de ikke startfarten (som står på linjen over). Det kan kanskje tyde på at elevene trenger flere påminnelser om når de er i hvilken situasjon.

I Captura-opptakene kommer det frem at alle elevene i større eller mindre grad har systematiske utfordringer. Det vil si at de «prøver og feiler» i stedet for å jobbe strategisk og analytisk for å løse oppgavene. Jeg ser utfordringen både når det er meningen at elevene skal bruke en «strategisk prøve og feile-metode» for å finne *k*, og når de løser andre oppgaver, som for eksempel å skrive inn riktig kode. Det første tilfellet ser vi for eksempel når Diana og Hedwig (C 3:2) skal finne en verdi av *k*. De går gjennom prosessen to ganger, først med g = -1,625 m/s^2, og så med g = -9,81 m/s^2, etter at læreren påpeker at de må ha samme tyngdeakselerasjon som på jorda. Det virker som at elevene kun i begrenset grad tar til seg den informasjonen de får av grafen, og i stedet for å justere én desimal av gangen og nærme seg rett svar, endrer de flere desimaler samtidig, noen ganger opp og noen ganger ned, uten noen klar plan. Flere ganger uttrykker de overraskelse når kulen kommer enten høyere eller lavere etter de har endret *k*-verdien, fordi de (tilsynelatende) forventer motsatt resultat. Kun én gang kommenterer de sammenhengen mellom *k* og kulens høyde, når Diana sier: «hvis jeg tar mindre tall, blir det høyere da?» Men Hedwig svarer ikke, og det blir ingen videre diskusjon om dette. Selv etter at elevene har gjennomgått prosessen én gang, ser vi den samme tendensen etter at de har justert *g* og skal gjøre det en gang til, når de øker *k* for å få kulen til å gå høyere (hvilket åpenbart er feil).

Et annet eksempel ser vi hos både Øyvind og August (C 1:1) og Arne og Berit (C 4:1) når de av ulike årsaker ikke får koden sin til å virke. I stedet for å lese gjennom koden linje for linje og verifisere hva som er rett og hva som ikke er det, endrer de tilsynelatende tilfeldige kodelinjer, verdier, tegn og uttrykk. Ofte endrer de flere ting på en gang før de kjører koden, og de klarer ikke å bruke tilbakemeldingen fra programmet til å komme videre.

Type utfordring påvirker læringsprosess

Captura-opptakene antyder at elever som bruker mindre tid på utfordringer knyttet til programmering og modellering, bruker mer tid på å diskutere resultatene sine i lys av fysikken de kan og skal lære. Datamaterialet lar oss ikke umiddelbart se om tidsbruken på ulike utfordringer gir innvirkninger på om elevene lærer fagspesifikke eller

programmeringstekniske konsepter. Det vi heller ser, er at indikatorene på dybdelæring som er beskrevet i kapittel 5.2 (Chin & Brown, 2000; Taub et al., 2015), er mye tydeligere hos Johanne og Linda (C 2:1, 2:2) enn hos noen av de andre gruppene. Dette kan bety at elevene i større grad tar del i dybdelæring dersom de får programmet til å kjøre fortere, uten å måtte bruke tid på syntaktiske og semantiske problemer i koden sin. Se avsnittet «indikatorer på dybdelæring er synlige når elevene jobber med flere representasjoner» (kapittel 6.4) for en mer detaljert beskrivelse av dybdelæringsindikatorene vi ser hos elever som jobber med opplegget.

6.4 Representasjonene i opplegget har mange ulike funksjoner

I designet av programmeringsaktiviteten ligger det en klar forventning om at de ulike representasjonene som elevene tvinges til å forholde seg til, skal fremme læring. Det er ingen selvfølge at dette er tilfelle, og særlig er det interessant å se på *hvordan* representasjonene bidrar til elevenes læring. Resultatene viser at representasjonene i opplegget har både utfyllende, begrensende og konstruerende funksjoner, og at de (blant annet) bidrar til læring på tre måter. For det første støtter de elevene i å evaluere koden og resultatene sine, fordi de får tilbakemelding fra programmet underveis. For det andre lar representasjonene elevene utforske fysikken, fordi de lett kan justere variabler og initialbetingelser. For det tredje ser vi at elevene får trening i å jobbe med sammenhenger mellom de ulike representasjonene, og at dette arbeidet sammenfaller med indikatorer på dybdelæring. I dette avsnittet tar jeg for meg disse tre punktene i større detalj, med eksempler fra datamaterialet og tilknytning til de tre funksjonene fra DeFT-rammeverket; utfyllende, begrensende og konstruerende.

Representasjonene støtter elevenes evalueringsprosesser

Koden som elevene skal skrive, gir først kun én type output, nemlig animasjonen. Det er derfor ikke overraskende at elevene bruker animasjonen til å bestemme om de har skrevet rett kode eller ikke. De har en forventning om hva animasjonen skal vise, og bruker det som kriterium når de evaluerer koden sin (Captura-opptak, f.eks. C 1:1). Mange av elevene uttrykker at animasjonen var svært viktig for å forstå hva de holdt på med. For eksempel i denne intervjusekvensen:

> Oliver: Uten animasjonen, så tror jeg vi ikke hadde skjønt så veldig mye.
>
> Ragnhild: Nei. Enig.

I: Hvis dere bare hadde fått grafen, for eksempel?

Oliver: Jo, hadde nok skjønt.. ja, hadde skjønt en del da.

Nathaniel: Ja.

Mikah: Men det hadde tatt lenger tid, føler jeg. Det hadde, på en måte, når du ser det, sånn, som en modell da, er det mye lettere å se hva det betyr, på en måte.

Ragnhild: Også, på en måte, når du, på en måte, endrer koden og sånn også, så er det mye, på en måte enklere å se, for eksempel, om ballen kommer enda høyere opp, så klarer du å se det, på, på en måte ...

Oliver: Animasjonen?

Ragnhild: ... animasjonen, ja. (I 3:12)

Også i andre intervjuer uttrykker elevene det samme – de mener at animasjonen er mer nyttig enn posisjonsgrafen (I 1:16, I 2:12). Når vi ser på Captura-opptakene, er det en gjennomgående trend at elevene snakker mer om posisjonsgrafen enn animasjonen etter hvert som tiden går. Fokuset skifter fra animasjon til graf (f.eks. C 3:1). Samspillet mellom animasjonen og grafen ser ut til å gå to veier. På én side begrenser animasjonen tolkningen av grafen. Det vil si at det blir helt tydelig for elevene hva posisjonsgrafen viser, fordi animasjonen gir en mer tilgjengelig representasjon. For eksempel ser vi at Diana og Hedwig (C 3:2) på et punkt sliter med å forstå grafen, fordi verdiene på aksene er gitt på standardform (f.eks. 6e-2). Da kjører de programmet sitt flere ganger for å se bevegelsen i sanntid. Selv om dette egentlig ikke tilfører noen ny informasjon, gjør det det lettere for elevene å forstå hva grafen egentlig viser.

På en annen side utfyller grafen animasjonen med mer informasjon om kulens nøyaktige bane, og verdier for tid og posisjon. For eksempel sier Øyvind: «Også hjelper det veldig på forståelsen av fysikken at man får en visuell framstilling av det som skjer. Og dobbel visuell også, siden vi fikk opp den grafen i tillegg. Som gjorde det veldig lett å ... sånn sett se etter feil og sånn da.» (I 4:7)

Ved ett tilfelle ser vi at koden begrenser elevenes tolkning av animasjonen. Øyvind og August har etter mye jobb fått en animasjon som ligner på den de skal få, og som de mener ser riktig ut. Etter å ha kjørt animasjonen noen ganger, sier Øyvind: «Nå har vi helt feil tyngdeakselerasjon og alt er bare kaos her.» (C 1:1) Det leder til at de ser gjennom koden på

nytt, før de spør læreren om hjelp. Dette viser to ting. For det første viser det at det er mulig å lage animasjoner som ser korrekte ut selv om fysikken i koden er feil. For det andre viser det at koden som elevene har skrevet tilsynelatende kan være en støtte når animasjonen skal tolkes i lys av fysiske lover.

Én av oppgavene elevene skulle gjøre var å regne ut kulens maksimale høyde ved bruk av bevaringsloven for mekanisk energi. Tanken var at elevene skulle få sammenligne koden sin med teorien, og se at den stemte. Dette er en svært naturlig – og viktig – evalueringsprosess som foregår i «ekte» arbeid med programmering i fysikk. Erfaringen fra klasserommet tilsier at mange av elevene ikke ser denne sammenhengen av seg selv. Mange ganger måtte jeg fortelle elevene at «nå har du sjekket koden din mot kjent teori og sett at den virker, så nå kan vi benytte den samme koden der hvor vi ikke har kjent teori, og stole på resultatene våre.» (f.eks. C 2:1, parafrasert)

Representasjonene lar elevene utforske fysikken

Elevene uttrykker det som positivt at programmeringen gjorde det enkelt å kjøre animasjonen med flere ulike verdier uten å bruke mye tid. Utsagn som støtter opp om dette er for eksempel:

> Johannes: Jeg synes det var liksom bra at vi skrev ned alle variablene, og så bytta vi på dem da, så da så man liksom hva slags funksjon de hadde for ballen da. (I 2:3)
>
> Sunniva: Jeg synes det var ganske greit når man hadde sånn, funksjoner at man kunne prøve ut forskjellige verdier selv, og se liksom hvordan det påvirket funksjonene og sånn. (I 3:3)
>
> William: Det jeg synes var gøy, var at man så på den grafen også kunne man se, hvis man for eksempel regner med da den verdien, så, hvordan grafen endra seg da, hvis man endra en annen verdi, så endra grafen seg igjen. (I 4:7)

Når elevene først har laget programkoden sin, så kan de bruke den omtrent som en simulering, og undersøke hvordan ulike parametere påvirker animasjonen og posisjonsgrafen. Det ser ut som at elevene opplever at dette styrker forståelsen av hva det er de driver med, og at de visuelle representasjonene (grafen og animasjon) utfyller de matematiske representasjonene (formler og ligninger på papir og i koden). De forskjellige representasjonene får en konstruerende funksjon, fordi det er lett for elevene å jobbe med sammenhengene mellom ulike størrelser i for eksempel Newtons andre lov, og se helt konkret

hvordan disse påvirker bevegelsen ligningen beskriver. Det indikerer at programmeringsmiljøet bidrar til at elevene får ta del i dybdelæringsprosesser.

Indikatorer på dybdelæring er synlige når elevene jobber med flere representasjoner

I dette avsnittet ønsker jeg å vise frem noen resultater som peker på at opplegget legger til rette for at elevene kan jobbe med sammenhengen mellom representasjoner, og at dette igjen bidrar til dybdelæring. Sekvensene er valgt på grunnlag av at de er merket eller kodet med koden «DeFT: Konstruerende» og at indikatorer på dybdelæringsprosesser er synlige. Det har alt vært nevnt at slike læringsprosesser er tydeligere i Captura-opptaket med Johanne og Linda (C 2:1, C2:2) enn hos de andre. Derfor går jeg først gjennom aktuelle sekvenser fra dette opptaket, før jeg tar for meg én annen interessant sekvens fra opptaket til Øyvind og August (C 1:2).

Den første halvtimen bruker Johanne og Linda til å skrive den første versjonen av koden. Mens de gjør dette, diskuterer de hyppig hvilken retning akselerasjonen skal ha, og hvordan farten til kulen endrer seg underveis i bevegelsen. I denne diskusjonen ser vi mange av indikatorene på dybdelæring beskrevet av Chin og Brown (2000). Elevene stiller mange spørsmål til hverandre om hvordan akselerasjonen påvirker farten, og produserer egne forklaringer og svar (både rette og gale) på spørsmålene sine. De knytter disse forklaringene til tidligere opplevelser, og prøver å bruke erfaringer fra når man kjører bil til å forklare akselerasjon og fart. Animasjonen bidrar til elevenes diskusjon ved å fortelle dem hva som skjer når akselerasjonen er henholdsvis positiv og negativ. Når de kjører koden med $a = 5$ m/s^2 og ser at kulen ikke kommer ned igjen, diskuterer de hvorfor dette er tilfellet, og hva som skjer med farten underveis i banen. Elevene jobber altså med sammenhenger mellom kodevinduet, animasjonen, formler og fysikkfaglige begreper, slik at disse får en konstruerende funksjon.

Den neste interessante sekvensen forekommer når Johanne og Linda skal sammenligne det analytiske resultatet fra bevaringsloven for mekanisk energi med resultatet fra simuleringen sin, for ulike verdier av tidssteget *dt*. Her ser vi at elevene ser på sammenhengen mellom ulike representasjoner. Både formler og ligninger, begreper, grafer og tidligere erfaringer med video og bilder pr. sekund brukes i diskusjonen til elevene, og får en konstruerende funksjon for læringsprosessen. I den samme diskusjonen ser vi flere av indikatorene på dybdelæring. Særlig ser vi at de fortløpende lager hypoteser som vurderes når programmet kjører, f.eks. når Johanne sier: «den regner jeg med blir ganske mye høyere da». For å vurdere resultatene sine

og eventuelle uoverensstemmelser med hypotesene, trekker elevene aktivt på erfaringer fra tidligere fysikktimer og hverdagssituasjoner.

Når elevene skal variere verdien av *k* for å finne den «riktige» verdien, ser vi en interessant diskusjon om hvilken funksjon *k* har i luftmotstandsuttrykket. Det kan virke som at elevene ikke umiddelbart har klart for seg hvordan denne parameteren påvirker luftmotstanden og kulens bevegelse. Før de kjører koden for ulike verdier av *k*, har vi følgende samtalesekvens:

> Johanne: Tror du at … den blir lavere når man har høyere k-verdi? Eller høyere?
>
> Linda: Om den kommer høyere opp, mener du?
>
> Johanne: Ja, hvis k-verdien er liksom, større. Jeg tror kanskje den blir lavere, at ballen kommer lavere.
>
> Linda: Ja … Hvis farten holder seg likt, holdt jeg på å si, og hvis vi tar et mindre tall. Så blir den vel mindr … nei, større kanskje? Hva var det du sa?
>
> Johanne: Jeg tror den blir mindre.
>
> Linda: Ja, mindre. Det gir mening.

Deretter prøver elevene noen forskjellige verdier for *k*, før de konkluderer:

> Linda: Så det betyr egentlig at jo større k-en blir, jo lavere kommer den.
>
> Johanne: Jo lavere blir høyden.

Det vi ser her, er at elevene først bruker formelen til å lage en hypotese om effekten av *k*, ved å se hva slags effekt justering av konstanten har for størrelsen av luftmotstandsuttrykket. Deretter kjører elevene koden med flere forskjellige verdier for å se hva som skjer, før de konkluderer med en (korrekt) påstand. Slikt hypotese- og teoriarbeid på liten skala er en indikasjon på at elevene her opplever dybdelæring, som beskrevet i kapittel 2.1 (Chin & Brown, 2000).

I Captura-klippet med Øyvind og August (C 1:2) ser vi også indikatorer på dybdelæring når elevene jobber med å justere størrelsen på luftmotstanden, men her ser vi enda tydeligere at Øyvind må forholde seg til en ny, uventet idé. Når elevene kjører programmet sitt og får en graf som ikke er symmetrisk (fordi grafen blir lineær når kulen når terminalfart), uttrykker Øyvind flere ganger at «det er ikke riktig» og «det gir ikke mening». Etter å ha kjørt programmet flere ganger, utbryter han: «Jo, selvfølgelig er det riktig! (…) Akselerasjonen er

jo null når ballen oppnår maksfart.» Denne sekvensen tyder på at Øyvind først presenteres med den nye ideen «grafen er ikke symmetrisk» som strider mot det han vet om vertikale kast, før han leter etter kriterier for å evaluere denne ideen. Først hevder han at koden er feil, men så får han den nye ideen til å henge sammen med det han vet fra før om luftmotstand og terminalfart, som beskrevet i KI-rammeverket. Det siste utsagnet om akselerasjon og terminalfart er en teoretisk beskrivelse av det han ser, som han produserer uoppfordret, og dette indikerer også dybdelæring (Chin & Brown, 2000). De samme indikatorene ser vi hos Johannes i et av intervjuene:

> Eller, jeg føler liksom at når vi la til luftmotstanden da, også så vi på grafen, liksom, så fikk jeg liksom bedre forståelse mellom ikke bevart og bevart mekanisk energi da. For da kunne man se liksom at når det ikke var bevart da, så gikk liksom mekanisk energi fra ballen til lufta. Ikke sant, så så man at det tok liksom lenger tid med å komme ned enn opp da. Fordi at luftmotstanden var der. Mens når det var bevart, så var det bare likt hele tiden, så kunne man se grafen at, det var liksom, symmetrisk da. Ja. (I 2:4)

Det virker som at når elevene endrer koden sin, slik at posisjonsgrafen endrer form, så er det et potensial for dybdelæring. Men det er ikke alle elevene som kobler grafens endrede form til teoretiske konsepter. For eksempel ser vi at Diana og Hedwig kommenterer at grafen blir «skeiv» når de legger på luftmotstanden, men de begir seg ikke ut på noen forklaring av fenomenet, annet enn å konstatere at det er sånn. Johanne og Linda kommenterer ikke grafens form i det hele tatt, de ser kun på høyden. I intervjuene er det kun Johannes som trekker frem posisjonsgrafens endrede form, noe som tyder på at det ikke var fremtredende hos flertallet av elevene.

Det vi ser i disse sekvensene, er at bruken av flere representasjoner sammenfaller med indikatorene for dybdelæring. Særlig er det fire deler av opplegget hvor læringsprosesser er mest tydelige: Når elevene skal finne uttrykket for kulens akselerasjon (med og uten luftmotstand), når de skal endre *dt* og sammenligne grafen med det analytiske resultatet for bevart mekanisk energi, når de legger til luftmotstanden og ser at posisjonsgrafen får en annen form, og når de skal endre *k* for å få den samme maksimale høyden som i forsøket med fjærkanonen.

6.5 Programmering kan bidra positivt til elevenes eierskapsfølelse

Når elevene blir spurt om hvordan de likte å jobbe med programmering, svarer nesten samtlige at det var spennende, gøy eller interessant (kodet som «gøy og interessant»). Det er ikke alle som uttrykker akkurat hva som var gøy eller interessant, men blant de som gjør det, virker det å være to aspekter som har innflytelse på elevenes motivasjon; variasjonsaspektet, og eierskapsaspektet.

Elevene er positive til programmering som variasjon

Programmeringsoppgavene oppfattes som *variasjon* til annen undervisning. Elevene setter pris på at timene har sett litt annerledes ut enn de pleier, og synes det er «spennende at vi gjorde fysikk på en helt annen måte enn vi har gjort tidligere» (I 3:8). Også de elevene som ikke nødvendigvis likte programmeringen, setter pris på variasjonen: «Samtidig er jeg glad for at man regner mest i boka i fysikk, liker det bedre. Men så tenker jeg at det er viktig med variasjon i timen, og gjerne noe jeg kan gjøre innimellom» (ES 2:1).

Elevene uttrykker at programmering påvirker eierskapsfølelsen både positivt og negativt

Kanskje det mest interessante aspektet knyttet til programmering og motivasjon, er hvorvidt elevene kjenner på *eierskap* til programmeringsoppgavene og den tilhørende læringsprosessen. Det er særlig i intervjumaterialet dette aspektet kommer fram, og vi ser at eierskapsfølelsen til elevene påvirkes både positivt og negativt i arbeidet med programmeringsoppgavene. For å belyse dette aspektet, bruker jeg sitater kodet med en eller flere av de tre kodene «eierskap», «nyttig» og «virkelighetsnært».

Positive indikatorer på eierskapsfølelse ser vi for eksempel når elevene blir spurt hvilken farge de valgte å ha på kulen i simuleringen (I 1:10). Ingen av de intervjuede elevene, og heller ikke elevene i Captura-opptakene valgte å la kulen være rød, som eksempelet i oppgaveteksten. Dette kan være et uttrykk for at elevene har behov for å gjøre programmet til sitt eget, og at kulens farge er en enkel måte å gjøre det på. I det samme intervjuet er det elever som har kalt kulen sin for «Dog» i stedet for «Ball» (I 1:9-10), som er et annet, lignende uttrykk for slikt eierskap. Vi ser også at kodeformatet er fleksibelt nok til at elever med litt ekstra kreativitet kan skrive programmet sitt annerledes enn foreslått for å løse oppgaven. I Captura-opptaket med Øyvind og August (C 1:2) kan vi se at de bruker mange kodelinjer på å definere krefter, summen av krefter og til slutt akselerasjonen, i stedet for å

gjøre denne utregningen for hånd. Programmet gir like gode resultater, men er i større grad elevenes eget verk. Ylva uttrykker dette når hun sier: «(…) det er veldig fint at "oi, jeg gjorde det." På en måte, man føler at man utvikler den ting, ja.» (I 4:6) Disse eksemplene viser fortrinnsvis at elevene kjenner på *proseduralt* eierskap over oppgavene, fordi de til en viss grad kan bestemme hvordan de vil presentere resultatene sine, og hvordan de vil går frem for å løse oppgavene de har fått.

Det kan være grunn til å tro at programmeringsaktiviteten også fostrer *kognitivt* eierskap over elevenes læringsprosess. Dette er basert på det elevene sier om at programmering i fysikk oppfattes som *virkelighetsnært* og *nyttig*. For eksempel er det en elev som skriver: «Når du bare jobber med oppgaver i boken er det ofte vanskelig å knytte det opp til virkeligheten og se sammenhenger. Men når vi programmerte kunne vi enkelt se hvordan det ville vært i virkeligheten, som gjorde det lettere å forstå temaet» (ES 2:4). Også i intervjuene trekker elevene frem dette: «(…) Og se hvordan det fungerte faktisk, og hvordan det påvirket en ball i virkeligheten, og ikke bare høre om det.» (I 4:3) Frida forteller at programmering er nyttig for å lære fysikk:

> Frida: Også brukte man liksom teknologi sammen med fysikk da, ikke bare var på penn og papir, men at liksom, kan bruke teknologi til å, eeh, liksom skjønne fysikkting. (I 1:19)

Når elever oppfatter opplegget som både virkelighetsnært og nyttig, er det også grunn til å tro at de opplever det som relevant. Som nevnt bruker flere av elevene ordet «interessant», som tyder på at spørsmålene de finner svar på i opplegget til en viss grad oppleves som elevenes egne spørsmål (se kapittel 7.1 for mer om dette). Dermed hevder jeg at elevene opplever eierskap ikke kun proseduralt, men også kognitivt, og at dette legger til rette for dybdelæring (Stefanou et al., 2004).

På en annen side uttrykker enkelte elever mer eksplisitt at de *ikke* følte eierskap til det de gjorde, for eksempel i følgende sekvens (I 1:6-7):

> **I**: Hvis dere skal sammenligne disse timene med vanlige timer. Hva synes dere om å jobbe på datamaskinen, fremfor å jobbe for hånd, med ligninger?
>
> Carina: Det er mindre regning.
>
> **I**: Er det bra, eller dumt eller?

Berit: Det er ganske greit, for PCen gjør det jo for deg på en måte.

Arne: Ja, men jeg synes egentlig det på en måte blir litt mindre fysikk, altså sånn, mindre matte da, mere data, du får ikke like mye eierskap til det å, faktisk regne ut noe. Jeg vet ikke, jeg bare følte ...

Berit: Du kommer ikke frem til svaret selv, på en måte. Så …

Arne: Du kan nærmest gjette, for du kan trykke og liksom se at det funker. Ja.

Fordi at datamaskinen gjør mye av regnearbeidet, opplever elevene at de ikke finner svaret selv. Dette kan ha sammenheng med hvor mye elevene har forstått av den koden de har skrevet. Dersom de ikke forstår hva de skriver, er det ikke rart at de ikke opplever at de har funnet svaret selv. Andre elever peker på at koden de får oppgitt har noe av den samme effekten. Eksempler er linjen for å plotte posisjonsgrafen, eller linjen for å tegne en luftmotstandspil i animasjonen. Her er det delte meninger fra elevene. Noen synes ikke at det gjorde noe (I 1:15), mens andre er mer kritiske:

I: (…) Hvordan opplevde dere å skulle ignorere deler av koden, som dere ikke skrev selv da?

Gunilla: Jeg synes sånne ting er litt irriterende. Fordi det er sånn "åja, du må bare akseptere at det er sånn", men du får på en måte ikke vite hvorfor det er sånn. Da ... Ja. Det synes jeg er litt irriterende.

Kjersti: Så du må, du må liksom skrive det, men du vet egentlig ikke hva det betyr.

Gunilla: Og da er det litt vanskeligere å huske det og forstå det òg. (I 2:10)

Hvis elevene ikke skjønner hva som foregår, så går det ut over både det prosedurale og det kognitive eierskapet til aktiviteten. Dersom elevene kun skriver det de får beskjed om, uten å bruke tid på å tolke og forstå det, så er koden «min» og ikke deres, og de får sånn sett lite eierskap til hvordan koden skal se ut og oppgaven skal løses. I forlengelsen av det er det heller ikke vanskelig å se for seg at elevene ikke nødvendigvis opplever (deler av) aktiviteten som relevant for sin egen læring. Hvis dette er tilfellet ofte og hos mange elever, så må konklusjonen være at programmeringsaktiviteten på denne måten hemmer dybdelæring.

Et annet aspekt man kunne tenke seg virket inn på elevenes prosedurale og kognitive eierskapsfølelse, er parprogrammeringen. Man kunne for eksempel se for seg at elevene opplevde mindre eierskap til koden når de ikke skriver den alene, eller at én av elevene i et

par tar mye av styringen og den andre blir sittende ved siden av. Det er tendenser til slike resultater i datamaterialet, for eksempel er det stor enighet i intervju fire om at det hadde vært bedre å jobbe to og to, men på hver sin datamaskin:

> **I:** (…) Så dere vil egentlig helst at begge jobber parallelt, på en måte, på hver deres PC?
>
> Ulrik: Ja.
>
> Ylva: Mhm.
>
> Åshild: Ja.
>
> (…)
>
> Øyvind: Det blir jo veldig fort til at bare én sitter på PCen mens den andre sitter ved siden av, når man bare har én PC på deling. (I 4:16-17)

Erfaringen fra klasserommene tilsier også at både lærer og elever glemte at man skulle bytte på å programmere underveis, og det er grunn til å tro at halvparten av elevene har skrevet en god del mer kode enn den andre halvparten. Likevel virker det som at de aller fleste elevene synes det fungerte veldig godt å jobbe to og to (I 1:7, I 2:7, I 3:9), og det er ingen som uttrykker eksplisitt at parprogrammeringen hadde en innvirkning på eierskapsfølelse.

Vi ser altså at på den ene siden styrkes elevenes prosedurale eierskap til oppgaven fordi at de får lov til å skape sitt eget program. Samtidig opplever de oppgavene som virkelighetsnære og relevante, og sånn sett styrkes det kognitive eierskapet. Disse to faktorene gir grunn til å tro at aktiviteten bidrar til dybdelæring. På den andre siden kan det hende at eierskapsfølelsen svekkes dersom elevene får presentert for mange ting som de ikke forstår hva betyr, eller hvordan de skal håndtere. Da kan de føle på at de ikke vet hva de gjør, eller hvorfor, og at programmet gjør utregningen for dem, heller enn at de finner svaret selv. Dette svekker både det prosedurale og det kognitive eierskapet, og dermed er det også en mulig begrensning for elevenes dybdelæring.

7. Diskusjon

Størsteparten av dette diskusjonskapittelet består av en diskusjon rundt programmering og dybdelæring, med utgangspunkt i forskningsspørsmålene og resultatene, i tillegg til en beskrivelse av et revidert undervisningsopplegg som tar høyde for denne diskusjonen. Videre vil jeg forsøke å plassere min oppgave i sammenheng med eksisterende forskning og teori på fysikkdidaktikkfeltet, og peke på interessante muligheter for videre forskning. Helt til slutt vil jeg kort beskrive noen begrensninger med denne oppgaven.

7.1 Programmering og dybdelæring

Resultatene fra datainnsamlingen og -analysen virker å sammenfalle med tidligere forskning som hevder at programmering kan bidra til dybdelæring. Det jeg ønsker å diskutere, med utgangspunkt i forskningsspørsmålene og resultatene mine, er noen aspekter ved programmeringsaktivteter som kan justeres for å bidra i større eller mindre grad til elevenes dybdelæringsprosesser. Det er klart at begrepet dybdelæring på mange måter er ganske vidt, og det er sannsynligvis mange ting som kan stimulere til dybdelæring. Siden dybdelæring per definisjon skjer over lengre tid, er det vanskelig å måle, fordi indikatorene jeg har brukt gjerne foregår i korte sekvenser. Det er liten tvil om at mange av aspektene ved programmeringsaktiviteten kan bidra til dybdelæring, og at dybdelæring kan skje også der hvor det ikke er tydelige observerbare indikatorer. For eksempel ser vi få indikatorer på dybdelæring hos Øyvind og August, men de uttrykker likevel at de har lært noe (f.eks. I 3:3).

Videre kan man problematisere hva det er elevene dybdelærer. For eksempel kunne man argumentere for at å få lov til å jobbe som en fysiker er en dybdelæringsprosess i seg selv, fordi det har muligheten til å endre på den grunnleggende forståelsen av hva fysikk er. Man kunne hevde at et opplegg med bruk av mange representasjoner gir dybdelæring, fordi elevene trenes i en svært viktig ferdighet, nemlig å bevege seg mellom representasjoner. Like fullt kan elever oppnå dybdelæring av fysikkfaglige konsepter, dersom de får riktig støtte. Det kan tenkes at forskjellige elever har lært forskjellige ting av å jobbe med programmeringsaktiviteten, og at alle har opplevd dybdelæring på ulike plan, både synlig og usynlig.

I det følgende argumenterer jeg for at programmeringsaktiviteten er svært fleksibel og kan brukes til mange ting, men at det virker mest nærliggende å bruke den til å lære fysikkfaglige

konsepter. Jeg fremhever de resultatene som viser at slik læring skjer, eller at slik læring ikke skjer. Kapittelet er i store trekk sortert i fire bolker knyttet til hvert sitt forskningsspørsmål – læring, utfordringer, representasjoner og eierskap. Det er en utfordring at mange av temaene henger tett sammen, og overskriftene er derfor kun et beste forsøk på å lage struktur i noe som er grunnleggende komplisert.

Hva er læringsmålene for programmering i fysikkfaget?

Som nevnt er det verdt å bruke litt tid på å diskutere hva det er vi ønsker å oppnå med programmering i fysikkfaget. Det er vel og bra å sikte mot dybdelæring, men vi må ha noen mål for hva det er vi vil at elevene skal oppleve dybdelæring i. Vil vi at elevene skal bruke programmeringsaktiviteten til å lære mest mulig av det tradisjonelle fysikkinnholdet, eller vil vi at de skal lære å programmere som fysikere, sånn at de på egenhånd kan modellere bevegelse i en rekke tilfeller? Man kan også se for seg at programmeringsaktiviteten er en aktivitet som først og fremst skal trene elevenes evne til å jobbe med flere representasjoner. Kanskje ønsker vi alle disse tingene? Før jeg videre diskuterer resultatene mine, vil jeg ta stilling til dette spørsmålet.

En aktivitet i fysikkfaget kan i grove trekk bidra til tre forskjellige mål. Ferdighetsmål handler om at elevene skal lære en ferdighet, for eksempel å bruke et redskap eller verktøy. Kognitive mål handler om at elevene skal tilegne seg kunnskap om fysikkfaglige konsepter og begreper. Affektive mål handler om at elevene skal utvikle «positive holdninger til naturvitenskap og en følelse av selvstendighet og oversikt» (Angell et al., 2019, s. 165). I utkastet til ny læreplan, ser vi at kompetansemålene domineres av kognitive mål («gjøre rede for …»), med enkelte ferdighetsmål innimellom («bruke og lage ...»). De affektive målene er fortrinnsvis beskrevet i overordnet del, hvor det blant annet står at elevene skal få «utfolde nysgjerrighet, skaperglede og engasjement» (Utdanningsdirektoratet, 2020). Når vi ser etter programmering i læreplanen, ser vi at det er tilknyttet både ferdighetsmål og kognitive mål. Under «digitale ferdigheter» står det om «bruk av programmering for å utforske fysiske problemstillinger», og kompetansemålet tilknyttet programmering har tilnærmet lik ordlyd. Elevene skal både *bruke* og *utforske*. Men programmering har også en klar rolle å spille i forbindelse med de affektive målene. Programmering er et nyttig og effektivt verktøy, og er en like integrert del av moderne fysikk som laboratorieforsøk. Sånn sett kan det vise elevene en ny side av hva fysikk er. Programmeringsaktiviteter gir også muligheter for skaperglede og engasjement hos elevene, og kan påvirke deres eierskap til faget.

Det har lenge vært uklart nøyaktig hva elevene skal lære av programmeringsferdigheter. Utfordringen er at «bruke og utforske» kan tolkes på svært ulikt vis av ulike personer. Dersom elevene åpner en ferdig kodesnutt og endrer verdien på en variabel for å se en simulering, så har de brukt programmering. Men elever som lærer å skrive en kodesnutt fra bunnen av, har også brukt programmering. Utkastet til ny læreplan åpner for en rekke måter for lærere å presentere programmering på, og hver enkelt lærer kan nærmest fritt velge språk, programmeringsmiljø, og vanskelighetsgrad. Som nevnt tidligere gjør dette at lærere kan «svippe innom» programmering én gang i løpet av året, og ikke bruke det hverken før eller senere. I et slikt scenario argumenterer jeg for at det er meningsløst å ha som mål at elevene skal lære å programmere. Programmeringsferdigheter er ferskvare, og hvis elevene ikke jobber med det jevnt over tid, rekker de færreste å bygge opp god programmeringskompetanse. Det vi heller kan se for oss, er at de får et møte med en autentisk, virkelighetsnær side av fysikken, et innblikk i hvordan fysikk kan foregå «på ordentlig», og en forståelse for noen av de helt grunnleggende programmeringskonseptene. Jeg mener det er realistisk å ha et slikt «overflatemål» for elevenes programmeringskompetanse samtidig som vi bruker programmering for å fostre dybdelæring av tradisjonelle fysikkonsepter. Dette oppfyller utkastet til kompetansemål, men legger vekten på de kognitive («utforske») og affektive («skaperglede og nysgjerrighet») målene heller enn ferdighetsmålet. Elevene skal bruke programmering, men de trenger ikke å mestre det.

Slik læringsmålene for mitt undervisningsopplegg er formulert, representerer de både kognitive og ferdighetsorienterte mål. Men de kognitive målene handler først og fremst om å forstå programmering- og modelleringsprosessen, og det er lite fokus på de fysikkfaglige elementene som tas opp i oppgavene. Dette reflekteres også i det elevene sier at de lærer – det er slående hvor mange som trekker frem at de har lært noe om programmering, heller enn fysikkfaglige konsepter. Hvis fokuset skal være på kognitive mål knyttet til fysikkfaglige konsepter heller enn programmering (som diskutert over), så kan det tenkes at deler av aktiviteten bør endres for å reflektere dette. Det inkluderer også formuleringen av læringsmålene for opplegget. I den videre diskusjonen fremhever jeg hvordan ulike aspekter ved undervisningsopplegget påvirker elevenes dybdelæring av fysikkfaglige konsepter. I kapittel 7.2 beskriver jeg en revidering av undervisningsopplegget (og læringsmålene) som tar høyde for denne diskusjonen, og fremhever fokuset på kognitive, fysikkrelaterte mål.

Som indikert tidligere i dette avsnittet, kan programmeringsaktiviteter også ha potensial til å bidra til oppnåelse av affektive mål som skaperglede og positive holdninger til faget. Det er kjent at mange elever har sterke stereotypiske oppfatninger av forskere og realister (f.eks. Angell et al., 2019; Meyer, Guenther & Joubert, 2019), og resultatdelen (kapittel 6.2) peker så vidt på at programmeringsaktiviteter kan ha påvirkning på disse. Et av læringsmålene for undervisningsopplegget er at elevene skal få større innsikt i hvordan «fysikere jobber med å modellere fysiske fenomener», eller hvordan fysikk «faktisk foregår». Dersom dette målet oppnås, kan det kanskje legge til rette for mer positive holdninger til fysikkfaget, som er basert på mer enn stereotypier. Utover dette er det i det følgende lite fokus på affektive læringsmål og elevenes syn på fysikkfaget mer generelt. Dette ville vært interessant å diskutere, men er nedprioritert av hensyn til tid og plass.

Programmerings- og modelleringstekniske utfordringer hemmer dybdelæring av fysikk

Resultatene indikerer at elever som bruker mye kapasitet på å håndtere utfordringer knyttet til programmering og/eller modellering og simulering, bruker mindre tid på å diskutere, stille spørsmål, lage hypoteser, knytte aktiviteter til gammel kunnskap, og rydde opp i uoverensstemmelser. Evidensgrunnlaget er selvfølgelig begrenset, men det kan være grunn til å tro at det er et mål å redusere særlig syntaktiske utfordringer så mye som mulig, slik at elevene får overskudd til å jobbe med fagspesifikke utfordringer knyttet til Newtons andre lov, krefter, luftmotstand og akselerasjon. Dette er i overensstemmelse med flere kilder i litteraturen (f.eks. Guzdial, 1994; Sengupta et al., 2013), og er ikke et revolusjonerende funn i seg selv. Det er ikke så rart at når elever må bruke energi på å lære seg et helt nytt verktøy, så er det mindre overskudd igjen til å reflektere over fysikkfaglige begreper og konsepter.

Et interessant spørsmål er hvor mye støtte man kan gi elevene i programmeringsprosessen uten at vi mister de positive effektene for fysikklæringen. Hvis man i praksis kunne gitt elevene hele koden, er programmeringen bare en glorifisert simulering. Men det skjer tilsynelatende noe når elevene skriver sin egen kode. Taub et al. (2015) hevder at programmeringsprosessen er verdifull for elevenes læringsprosess, og jeg mener at mine resultater viser det samme. Særlig er det tydelig i Captura-opptaket med Johanne og Linda, men flere elever uttrykker også i intervjuene at de har lært noe mer om luftmotstand, og at programmeringsaktiviteten hjalp dem med å forstå fysikken bedre. Én av elevene har skrevet: «Jeg har lært at man må tenke på en logisk måte for at programmet skal skjønne det. Det gjør det enklere for oss også å skjønne det bedre.» (ES 1:09) Dette har også støtte i annen litteratur, som hevder at å lære programmering og fysikk sammen er bedre enn å lære de to

emnene hver for seg (Guzdial, 1994; Landau, 2006; Malthe-Sørenssen et al., 2015). Det tilsier at noen av utfordringene knyttet til programmering og modellering er *ekstern* belastning som begrenser potensialet for dybdelæring, mens noen av utfordringene er *relevant* belastning, som i ytterste konsekvens kan bidra til at elevene opplever dybdelæring.

Av Captura-opptaket med Arne og Berit (C 4:1), ser vi at syntaktiske utfordringer som parentesbruk, innrykk, kolon og små/store bokstaver er *ekstern* belastning, som fører til at mye tid går uten at elevene nødvendigvis opplever læring. Dette eksempelet viser også, sammen med opptaket av Øyvind og August (C 1:1) at elevenes feilsøking ofte er ineffektiv og usystematisk, med stor grad av prøving og feiling. Basu et al. (2016) plasserer «feilsøking» og «systematiske utfordringer» i to forskjellige kategorier, men mine resultater peker på at de er sterkt forbundet. Videre virker det å være tydelig at disse feilsøkingsprosessene ikke fremmer dybdelæring, og sånn sett er ekstern belastning vi ønsker å redusere. Slike systematiske utfordringer er også beskrevet av Kohl og Finkelstein (2008) når elever jobber med flere representasjoner, og det er nærliggende å tro at noe av årsaken til elevenes utfordringer handler om bruken av representasjonsformer. Mer om dette i avsnittene under.

Det ser også ut som at noen av de programmeringstekniske utfordringene kan være relevant belastning som bidrar til dybdelæring. Semantiske utfordringer knyttet til hva som skal stå hvor i koden, kan være eksempel på dette. For eksempel sier Diana på et tidspunkt til Hedwig: «skal *k* stå innenfor eller utenfor løkken?» (C 2:1) En diskusjon om hvor parametere skal stå, kan man se for seg leder til bevissthet om hvilke størrelser som endrer seg over tid, og hvilke som er konstante. Andre elever peker også på forskjellen mellom det som skjer utenfor løkken («default verdier, utgangspunktet», I 4:9) og det som skjer inne i løkken («ballen beveger seg», I 3:6).

Utfordringene knyttet til modellering og simulering er ikke like enkle å kategorisere som enten ekstern eller relevant belastning, ettersom de arter seg litt forskjellig hos forskjellige elever. Som nevnt i resultatdelen, er det flere elever som strever med å velge riktige initialbetingelser og parametere. Umiddelbart skulle man tro at dette var relevant belastning, siden rette initialbetingelser henger tett sammen med en forståelse av det fysiske systemet man modellerer. Men hos Øyvind og August (C 1:1) ser vi klart at usikkerhet rundt initialbetingelsene hindrer elevene fra å lete etter feil andre steder i koden. Det er nærliggende å tro at dersom de var helt sikre på at initialbetingelsene var rett, hadde de

raskere vurdert å endre kodelinjene inne i while-løkken. Også andre elever operer med feil initialbetingelser selv uten å vite at de er feil (f.eks. C 3:2, I 2:7), og dette er åpenbart ikke fremmende for dybdelæring. Dersom utfordringer knyttet til initialbetingelser skal være relevant belastning, må man sørge for at elevene tar del i en målrettet prosess for å finne rette betingelser og parametere. En annen utfordring knyttet til modellering og simulering er hvorvidt elevene forholder seg til forsøket de skal modellere mens de jobber med programmering. I avsnittene under diskuteres noen aspekter ved det å forholde seg til flere representasjoner, som virker å være tett forbundet med dette.

Programmering som én og flere representasjoner

For å ta diskusjonen til Gravel og Wilkerson (2017) (se kapittel 3.2) et lite skritt videre, vil jeg argumentere for at programmeringsmiljøet som representasjonsform (med både kodevindu, animasjonsvindu, graf og eventuell numerisk output) skiller seg fra andre representasjoner ved at det på en veldig konkret måte lar elevene bruke, teste og utvikle sin egen fysikkforståelse. Andre datamaskinsimuleringer og animasjoner er ofte (kanskje ikke alltid) et medium som kommuniserer ny kunnskap til elevene, og på mange måter er de bare en annen form for lærer. Grafer og resultater som elevene produserer selv med penn og papir er først og fremst et uttrykk for elevenes egen fysikkunnskap.

Programmeringsmiljøet, derimot, slår på mange måter sammen disse to aspektene. På den ene siden må elevene bruke sin egen fysikkforståelse for å lage et operativt program, og de får tilbakemelding på den fysikkforståelsen når de arbeider med å tolke resultatene sine. På den andre siden kan programmet gi resultater som er «gjemt» i koden, og som elevene ikke forventer på forhånd. I likhet med autentisk fysikerarbeid, må elevene bruke den kunnskapen de allerede har til å lage et program som kan lære dem nye ting. I stedet for å være mengdetrening i regning og formler, åpner programmeringsmiljøet for en setting hvor elevene kan se konkrete sammenhenger mellom det de kunne fra før, og det de lærer av programmet.

Når man programmerer, arbeider man med mange ulike representasjoner samtidig, både grafiske, numeriske, algebraiske og begrepsmessige. Jo bedre man behersker sammenhengene mellom disse representasjonene, desto mer av datamaskinens muligheter kan man utnytte. Det er kjent at mange elever har utfordringer med å bevege seg mellom ulike representasjonsformer (Angell et al., 2019), og det virker derfor avgjørende å gi nok støtte i disse prosessene. Gode programmeringsaktiviteter bør kunne modellere for elever

hvordan man kan bruke ulike representasjonsformer om hverandre for å lære noe nytt om fysiske konsepter.

Det går an å se for seg at programmeringsmiljøet kun er en sammensetning av mange ulike representasjonsformer som vi allerede bruker. I kodevinduet behandles ligninger og formler, og i graf, animasjons- og outputvinduene behandles eksperimentelle og grafiske representasjonsformer. Men kanskje kan man si at kodevinduet er en ny type representasjonsform, som i større grad enn andre representasjonsformer vi er vant med krever system og struktur. Selv om programmering innebærer å skrive ned formler og ligninger, er ikke kodevinduet en egnet representasjonsform for å manipulere og løse ligninger, eller annen algebraisk aktivitet. Derimot kan kodevinduet gi elevene hjelp til å sortere fysikkunnskapen sin, og bidrar kanskje i større grad til å for eksempel skille mellom initialbetingelser, konstanter og tidsavhengige størrelser. Kanskje kunne man kalle kodevinduet en *algoritmisk* representasjonsform?

Det er krevende å forholde seg til flere representasjoner

Jeg har allerede nevnt at elevene bruker mye tid på prøving og feiling når de jobber med programmering, og at Kohl og Finkelstein (2008) finner de samme tendensene hos nybegynnere som arbeider med flere representasjonsformer. Det er ikke utenkelig at elevenes vanskeligheter når de programmerer skyldes at det er mange representasjoner å forholde seg til, og at kodevinduet er en ny og uvant representasjonsform for elevene. Resultatene viser at elevene strever med å koble sammen informasjon fra de forskjellige representasjonene, og bruke denne informasjonen når de skal evaluere koden sin og resultatene sine. For eksempel ser vi at de aktivt bruker animasjonen til å evaluere koden, men de har ikke nødvendigvis lett for å si om animasjonen er korrekt eller ikke. Vi antar at elevene vet hvordan det ser ut når man kaster en kule oppover, men de blir likevel usikre på om kulen skal komme ned igjen i animasjonen eller ikke.

Ainsworth (2006) poengterer at når en representasjon skal begrense tolkningen av en annen, er det avgjørende at elevene forstår den begrensende representasjonen. Dersom elevene skal bruke animasjonen som vurderingsgrunnlag, må de ha en helt klar oppfatning av hva det er de forventer å se. Det fordrer at de ser sammenhengen mellom forsøket de har gjort og animasjonen. Det virker som at elevene må støttes i å etablere denne sammenhengen før de begynner å jobbe, sånn at de raskt kan se på animasjonen og bestemme om resultatet gir mening eller ikke. Dette gjelder også hvis vi ønsker at elevene skal bruke forsøket med

fjærkanon som referanseramme for programmeringsaktiviteten. Resultatene viser at mange elever brukte resultatet fra simuleringen uten luftmotstand for å finne verdien av *k*, og dette tyder på at sammenhengen med forsøket ikke er godt nok understreket for elevene.

Bruken av flere representasjoner kan stimulere til dybdelæring

Som vist i kapittel 6.4, er det flere sekvenser hvor bruken av flere representasjoner sammenfaller med indikatorer på dybdelæring. Elevene bruker representasjonene på flere forskjellige måter, både som en hjelp til å tolke hver enkelt representasjon, til å få utfyllende informasjon, og til å se sammenhengen mellom de ulike representasjonene. Programmeringsmiljøet gir elevene umiddelbar tilbakemelding, men de må bruke sin fysiske intuisjon for å se om de har gjort rett eller ikke. Vi ser også at elevene setter pris på å kunne bruke programmet som en simulering etter det er skrevet ferdig, for å undersøke mange forskjellige scenarioer.

Resultatene viser at det særlig er fire prosesser i løpet av programmeringsaktiviteten hvor bruken av flere representasjoner stimulerer til dybdelæring (kapittel 6.4). Den første er å skrive ned koden, når elevene må trekke på begreper og formler de kan fra før, og sette disse i sammenheng med algoritmene i kodevinduet. Her er det særlig det å finne akselerasjonen både med og uten luftmotstand som er utfordringen for mange elever, og som virker å være der de lærer mest. Videre ser vi at oppgaven om nøyaktighet gir gode diskusjoner og tydelige læringsprosesser. Når elevene legger til luftmotstand, har de mulighet til å koble det de ser i posisjonsgrafen til teoretiske konsepter som mekanisk energi og terminalfart. Til slutt ser vi at oppgaven hvor de skal finne en verdi for *k* ved hjelp av prøving og feiling, har potensial til å gi gode prosesser rundt hypotesetesting og evaluering knyttet til flere representasjoner. Gjennomgående i opplegget ser vi også at elevene bruker koden og animasjonen til å se effekten av de ulike parameterne i modellen.

Hvis vi skal prøve å bruke dette til å si noe mer generelt om hvordan arbeid med representasjoner kan sees i sammenheng med programmering og dybdelæring, ser det ut til å være enkelte prosesser hvor representasjoner har konstruerende funksjoner og elevene opplever dybdelæring:

- Når elevene skal skrive kode, er de nødt til å benytte fysikkunnskap de har fra før, og strukturere denne. Det innebærer blant annet å se sammenhengen mellom formler og ligninger de kjenner fra før, og hvordan disse er representert i koden.

Det innebærer også å identifisere hvilke variabler som er konstante og hvilke som endrer seg, og sette disse på rett plass.

- Når elevene får sammenligne resultatene fra programmet med analytiske resultater, ser de at ulike måter å jobbe på kan gi det samme svaret. De får mulighet til å se hvordan valget av tidssteg er avgjørende for modellens nøyaktighet, og kan i forlengelsen av det (kanskje) få bedre innsikt i hva som skjer inne i løkken.
- Når elever kjører koden sin, må de tolke de grafiske resultatene sine i lys av det de vet fra før. Dette kan lede til bedre forståelse av sammenhengen mellom variablene i en formel, av effekten av ulike parametere i koden, og det kan lede til innsikter om fenomener som er vanskeligere å se i formlene. Det er når elevene kjører koden at de får mulighet til å oppdage nye ting. Denne prosessen innebærer både de hypotesene elevene stiller før de kjører programmet, og når de ser resultatene i lys av hypotesen etter de har kjørt koden.

Dersom vi sammenligner med resultatene til Taub et al. (2015) (kapittel 3.2), ser vi at dette ligner på tre av deres fire domener, nemlig det strukturelle, det prosedurale og det iverksettende. Det systematiske domenet mangler nok i mine resultater siden elevene kun modellerer én kule, og ikke et system av flere objekter.

Programmering åpner for kognitivt og proseduralt eierskap

Av resultatene kommer det frem at programmeringsaktiviteten har en positiv effekt på elevenes kognitive eierskap. Det virker som at problemstillingen elevene jobber med oppleves som virkelighetsnær og lett kan kobles til forsøket elevene har sett og gjennomført, og at programmering som konsept virker å ha status som nyttig hos elevene (i likhet med elevene til Bott et al. (2019)). Sånn sett ser det ut som at elevene opplever kognitivt eierskap til oppgavene, til tross for at de ikke selv har valgt problemstillingen. Savery og Duffy (1995) påpeker at det er mulig for elever å ta til seg en gitt problemstilling og gjøre den til sin egen, dersom problemet er presentert på rett måte. Dette er viktig, fordi «elevens mål i stor grad avgjør hva det er som læres» (Savery & Duffy, 1995, s. 32, min oversettelse). Det betyr at selv om programmeringsaktiviteten er ganske «satt», så er det mulig for elevene å kjenne på kognitivt eierskap til det de holder på med, og sånn sett være disponible for dybdelæring.

På den annen side viser resultatene også at flere elever opplever mindre eierskap til programmeringsaktiviteten enn andre mer tradisjonelle aktiviteter. Dette skyldes

tilsynelatende en manglende forståelse av koden som er gitt på forhånd, og koden elevene selv skriver. Elevene uttrykker at det er datamaskinen som regner ut svaret for dem, i stedet for at de finner svaret selv. Det tyder på at de ikke nødvendigvis ser på programmering som et verktøy de selv kan bruke og manipulere, men at de heller har en følelse av å «skrive av fra malen» (I 2:11). Selv om elevene føler eierskap til problemstillingen, så mangler de altså eierskap til de prosessene de utfører for å svare på den.

Dette betyr kanskje at i stedet for å strebe etter at elevene skal produsere mange egne spørsmål og problemstillinger, bør vi heller satse på å «selge inn» problemstillinger vi mener er gode, og som elevene kan adoptere som sine egne. Det er lite hensiktsmessig å la elevene velge egne problemstillinger dersom de ikke skjønner hvordan de skal bruke datamaskinen til å løse dem. I stedet for ser det ut til at elevene må få mer støtte til å faktisk forstå hva som skjer i programmet, og hvordan de kan bruke koden til å finne frem til ny informasjon. Én ting er at det blir riktig resultat, men de bør helst skjønne hva de har gjort rett, slik at det er de selv som løser oppgaven, og ikke datamaskinen eller læreren.

Det er verdt å nevne at de aller fleste elevene i prosjektet møtte programmering for aller første gang, og sånn sett er det ikke underlig at de trengte tett oppfølging for å forstå programmet sitt, heller enn å skulle finne egne spørsmål å besvare. Men man kan se for seg at programmering om noen år vil være implementert også på tidligere klassetrinn, og da kan man håpe at det er flere muligheter for å gi elevene frihet som de klarer å bruke. Fordi programmering er grunnleggende fleksibelt, og den samme koden kan brukes på mange tilfeller ved hjelp av små endringer, er det tilsynelatende et utappet potensial for å la elevene jobbe med sine egne problemstillinger. Særlig kan dette være relevant for undervisning i fysikk 2, hvor et av kompetansemålene i utkast til ny læreplan er at elevene skal «utforske en selvvalgt teoretisk eller praktisk problemstilling (…)» (Utdanningsdirektoratet, 2020). En slik oppgave *må* ikke inneholde programmering, men det er tydelig at den *kan*.

Sånn sett virker programmering å være et effektivt verktøy for å fostre elevenes prosedurale eierskap, deres kognitive eierskap, og dermed også dybdelæring. Fordi programmering ikke kun er et uttrykk for elevenes kunnskap, men også kan vise elevene noe de ikke visste fra før, er programmeringsmiljøet en plass for å både stille og finne svar på spørsmål knyttet til fysikk. Og fordi det til syvende og sist er få regler for hvordan et program skal skrives, åpner programmeringskode som representasjonsform opp for at elevene kan være kreative og løsningsorienterte når de jobber med fysikkproblemer. Men som jeg har vist er det avhengig

av at elevene har nok overskudd og forståelse av hvordan man modellerer ved hjelp av programmering. Dette fordrer kanskje mer kompetanse enn det læreplanen krever, men er likevel en spennende tanke. Kanskje kan man argumentere for en felles standard om hvilke(t) programmeringsspråk man bruker i hele skoleløpet, slik at elever stadig møter det samme språket. Algoritmisk tenkning, programmering og problemløsning er ikke begrenset til ett språk, men det virker fornuftig å redusere belastningen knyttet til å lære seg nye språk, samtidig som man åpner for større fleksibilitet, eierskap og nytteverdi høyere opp i skoleløpet.

Støttestrukturer som kan være nyttige ved bruk av programmering i fysikk

Som nevnt i metodedelen, var vi tre lærere i klasserommene hvor undervisningsopplegget ble gjennomført. Dette ga oss mye tid til å hjelpe hver enkelt elev, og mange av svakhetene i oppgavesettet lot seg utbedre av oppfølging fra en av oss tre. I et vanlig klasserom med bare én lærer, er behovet for støttestrukturer innbakt i oppgavesettet desto viktigere, slik at elevene i minst mulig grad er avhengig av lærerens hjelp for å komme videre. I lys av de temaene som er diskutert over, vil jeg nå peke på hvordan man kan organisere støttestrukturer som bidrar til at elevene får ta del i dybdelæring av fysikkfaglige konsepter når de arbeider med programmering, uten at læreren må løpe maraton i klasserommet. Slik jeg ser det har disse støttestrukturene tre hovedfunksjoner:

- De skal redusere mengden ekstern belastning knyttet til syntaktiske og feilsøkingsrelaterte utfordringer så mye som mulig.
- De skal støtte elevene i hvordan de forholder seg til og bruker de ulike representasjonene i opplegget.
- De skal bidra til at elevene forstår det de skriver i koden og kan ha kognitivt eierskap til kodeprosessen.

Noe av denne støtten kan gis av læreren når vedkommende underviser, men den kan også gis skriftlig som del av oppgavesettet.

Når det gjelder syntaktiske utfordringer, hevder Sengupta et al. (2013) at visuell, blokkbasert programmering er mer hensiktsmessig enn tekstbasert programmering, fordi man da unngår hele syntaksproblematikken. Dersom man ønsker at elevene skal få se programmering slik den ser ut for fysikere, trenger vi andre måter å motarbeide syntaktiske utfordringer på. Aho et al. (2014) foreslo å gi elevene et «oppslagsverk» med vanlige kommandoer som de kunne referere til. Kanskje kunne elevene også fått en sjekkliste med syntaktiske ting å sjekke

dersom koden ikke kjører. Det kunne redusere noen av utfordringene knyttet til feilsøking, og la elevene raskere finne ut hva de gjør galt. Dette er en støtte som er enkel å gi, uten at den reduserer mengden kode elevene skal skrive. I forlengelsen av dette vil jeg trekke frem poenget til Sørby og Angell (2012) om at oppgaveteksten til elevene bør modellere hvordan elevene skal jobbe, med mål og delmål som oppfordrer elevene til å tenke over hva de endrer og hvorfor. Kanskje bør man også forsøke å flytte fokuset mer over på semantiske utfordringer, for eksempel ved å gi elevene ferdige kodelinjer, og heller be dem plassere dem i riktig rekkefølge. Da slipper de unna de fleste syntaksfeilene, men de blir likevel tvunget til å bruke sin forståelse av både fysikk- og modelleringskonsepter.

Når det gjelder bruken av flere representasjoner, har jeg beskrevet over hvordan dybdelæring skjer når elevene jobber med sammenhengen mellom formler de kjenner og koden de skal skrive, når de sammenligner sine egne resultater med analytiske, algebraiske resultater, og når de tolker de grafiske resultatene sine i lys av den fysikkunnskapen de har fra før (representert ved mange ulike representasjoner). Det er tydelig at elevene må støttes i disse prosessene, og det virker hensiktsmessig å gi elevene mer eksplisitte beskjeder om hvordan de skal bruke ulike representasjoner. For eksempel innebærer dette å formidle hvordan de kan bruke Newtons andre lov til å finne akselerasjonsuttrykket som skal inn i koden, og det innebærer å minne dem på hvilke resultater de forventer å se i animasjoner og grafer før de kjører programmet. Chin og Brown (2000) foreslår at man eksplisitt skal be elevene om å ta del i prosesser som stimulerer til dybdelæring, som å stille gode spørsmål, lage hypoteser, og konstruere egne forklaringer. Det kan for eksempel være hensiktsmessig å «utbrodere» med flere korte oppgaver der hvor elevene skal jobbe med sammenhenger mellom representasjoner, og kanskje også gi noen hint til elevene om hva man ønsker at de skal diskutere før og etter de kjører koden sin.

At elevene ikke føler eierskap til kodeprosessen, skyldes tilsynelatende at de ikke skjønner hvorfor og hvordan programmet virker, og hvorfor de skriver det de skriver. Dette er et komplekst problem, og har sannsynligvis mange ulike årsaker og løsninger. For eksempel kan det være at det skyldes lærerens undervisning før elevene begynner med oppgavesettet. Det kan skyldes at introkurset elevene gjennomfører de første 60 minuttene ikke gir tilstrekkelig innsikt i modelleringsprosessene og Euler-Cromer-algoritmen, eller det kan skyldes at når elevene arbeider med oppgavesettet, glemmer de hva de har vært gjennom tidligere. Det kan også skyldes at elevene responderer negativt på kodelinjer de ikke blir forklart, som for eksempel linjen som tegner en vektor i animasjonsvinduet. For å gi elevene nok støtte i

kodeskrivingsprosessen, kan det tenkes at det er to utfordringer som må håndteres. For det første kan oppgavesettet – som nevnt – i større grad fokusere på semantiske utfordringer, og be elevene reflektere over hva som skal innenfor løkken, og hva som skal utenfor, og hvordan linjene i Euler-Cromer-algoritmen henger sammen og bidrar til å beskrive bevegelse. For det andre kan elevene hjelpes til å reflektere over hvilket system det er de beskriver, og hvilke parametere de må ha med. Dersom elevene får flyttet fokuset fra «hvordan skriver jeg denne linjen», til «hva må jeg ha med i programmet, og hvilken rekkefølge skal de komme i», kan det tenkes at de i større grad forstår koden de skriver. I mitt oppgavesett var kodelinjene for å definere initialbetingelser gitt, mens koden for å modellere bevegelse var overlatt til elevene. Kanskje burde det motsatte vært tilfellet – da kunne elevene fokusert mer på fysikken, og mindre på modelleringen.

7.2 Endringer i undervisningsopplegget som følge av analysen

I lys av diskusjonen virker det som at et effektivt grep for å fremme elevenes dybdelæring i programmeringsaktiviteter, særlig med tanke på kognitiv belastning og eierskap, ville være å gi elevene mer programmeringstrening. Hvis man for eksempel ga elevene flere, kortere oppgaver hvor de skulle simulere bevegelse med Euler-Cromer, kunne man se for seg at de ble vant til syntaksen som hører med en slik simulering. Da ville sannsynligvis den eksterne belastningen knyttet til programmeringstekniske utfordringer blitt redusert, og forståelsen og eierskapsfølelsen til programmet ville økt.

For å få det fulle potensialet ut av programmeringsaktiviteter i fysikken, må man rett og slett programmere mye. Jo mer man programmerer, desto lettere vil det være å utnytte potensialet for dybdelæring. Men jeg har allerede påpekt at en slik «gjennomsyring» av programmering i fysikkfaget er usannsynlig, og på grunn av begrenset med tid i timeplanen trenger vi opplegg av en begrenset størrelsesorden. Derfor har jeg revidert undervisningsopplegget som har vært utgangspunkt for diskusjonen, med et mål om å ta høyde for de aspektene som er diskutert. Erfaringen etter å ha gjennomført undervisningsopplegget er at det i det store og hele fungerte ganske bra. Derfor er revideringene jevnt over ganske små, og består i første rekke av endringer i ordlyd underveis, heller enn store omstruktureringer i oppgaveteksten. Her presenteres kort de endringene som er gjort, med begrunnelse. Den reviderte oppgaveteksten til elevene ligger i vedlegg G.

Læringsmålene er justert for å flytte fokuset til kognitive, fysikkfaglige mål

Som nevnt oppfatter jeg det som hensiktsmessig å bruke programmeringsaktiviteter til å hovedsakelig fostre dybdelæring av kognitive og affektive læringsmål. De originale læringsmålene for aktiviteten var i stor grad fokusert på aspekter knyttet til programmering og simulering. Ordlyden er nå endret, slik at fokuset i større grad er på å *bruke* programmering til å *forstå* fysikk, heller enn å forstå alle programmering- og modelleringskonseptene. Endringen er ikke stor, men fremhever blant annet luftmotstandsuttrykket og Newtons andre lov. Som poengtert i kapittel 7.1, har jeg ikke diskutert affektive mål i nevneverdig grad, og det er derfor ikke belegg for å gjøre revideringer som tar høyde for dette.

Introduksjonskurset på Trinket.io er endret for å gjøre elevene tryggere på modellering

For å sørge for at elevene er bedre kjent med løkker og modellering av bevegelse før de begynner på aktiviteten om luftmotstand, er det lagt inn en ekstra oppgave i introduksjonskurset som veileder elevene i hvordan de skal sette opp en løkke for å modellere bevegelse. Tanken er at dette vil gjøre dem sikrere på syntaktiske og semantiske aspekter når de begynner på oppgavene om vertikalt kast. I tillegg er forklaringen av Euler-Cromer-metoden endret, slik at den i større grad er sentrert rundt vei-fart-tid-formlene elevene kjenner fra før. Det reviderte kurset er tilgjengelig her: https://trinket.io/jonathan_b_waters-5946/courses/intro-til-programmering.

Instruksene og oppgavene i oppgavedokumentet er endret

I diskusjonen har det vært nevnt flere deler av undervisningen hvor elevene trenger mer støtte. De har utfordringer med å se sammenhenger mellom representasjoner, og de har vanskeligheter med å trekke på tidligere fysikkunnskaper. Samtidig ser vi at de synes det er utfordrende å programmere, og at særlig programmeringstekniske utfordringer virker å hemme dybdelæring av fysikk.

Derfor er en del av instruksene og oppgavene i teksten endret, for å legge til rette for flere øyeblikk hvor dybdelæring kan skje. Det er flere små endringer, og jeg vil ikke ramse opp alle, men hovedtrekkene er som følger:

- Jeg har valgt å ta vekk noe av støtten elevene får i de første deloppgavene, slik at de selv må tenke mer på hvilke konstanter og variabler de trenger. Til gjengjeld har jeg prøvd å gi mer støtte når elevene skal skrive inn koden i while-løkken i programmet, blant annet ved å oppgi formelen for farten etter et tidssteg *dt*.

- Koden elevene skal skrive er nå delt i tre områder i stedet for to. I tillegg til «Initialbetingelser» og «While-løkke» er det nå et område som heter «Konstante størrelser», for å hjelpe elevene med å se sammenhengen mellom de ulike verdiene de bruker.
- For å hjelpe elevene med å luke ut syntaktiske feil, er det gitt en «smørbrødliste» med mulige feilkilder etter de har skrevet den første delen av koden.
- For å hjelpe elevene å bruke ulike representasjonsformer, er det nå eksplisitte oppfordringer til å bruke Newtons andre lov, å bruke penn og papir, og å sammenligne med forsøket de har gjort tidligere. Jeg har også prøvd å tydeliggjøre oppgaveteksten der jeg vil at de skal kommentere posisjonsgrafen, og der oppgavene går ut på å endre verdier i koden og se hva som skjer.
- For å stimulere elevenes fysikkfaglige diskusjoner, er det nå flere oppgaver som ber elevene tenke før de kjører koden – hva er det de forventer, og hvordan ser resultatet av simuleringen ut sammenlignet med hva de så for seg? Hvorfor eller hvorfor ikke ser de det de ser? Mye av dette lå «implisitt» i den originale oppgaveteksten, men er nå tydeliggjort.

Elevene får tildelt et «oppslagsverk» med vanlige kommandoer

Inspirert av Aho et al. (2014) har jeg laget et lite oppslagsverk, eller en samling med eksempler, som kan deles ut sammen med opplegget. Her står det kort om hvordan variabler fungerer, hvordan man lager objekter i Trinket.io, hvordan løkker virker og hvordan man kan modellere bevegelse. Jeg har også lagt inn en liste over feilsøkingstips. Oppslagsverket ligger tilgjengelig i vedlegg H.

7.3 Implikasjoner av denne oppgaven

I introduksjonen min, samt i kapittelet om tidligere forskning, har jeg gjort det klart at selv om det har blitt gjort noe forskning på implementering av programmering i fysikkfaget, er det fortsatt behov for å finne ut av hvordan dette skal se ut i den norske skolen. Programmering skal inn i læreplanen, men det er ingen tydelig, felles idé for akkurat hvor mye eller hva slags programmering elevene skal ta del i. I dette avsnittet vil jeg kort diskutere hvor mitt prosjekt passer inn i den forskningen som allerede er gjort, og hva den tilfører feltet. Deretter kommer jeg med noen innspill til videre arbeid med programmering i fysikk, som forhåpentligvis er til

nytte for fysikklærere og andre fysikkdidaktikere, før jeg peker på noen muligheter for videre forskning.

Sammenheng mellom denne oppgaven og tidligere forskning

Det er lite forskning som kan fastslå at programmering i fysikkfaget fremmer læring av fysikkens begreper og konsepter (Caballero et al., 2020; Nordby, 2019). I denne oppgaven har jeg fulgt opp arbeidet til Taub et al. (2015), ved å lete etter systematiske indikatorer på at læring skjer hos elevene når de arbeider med programmering, og jeg har trukket frem noen aspekter ved programmeringsaktiviteter som kan justeres for å fremme dybdelæring av fysikk.

Programmering og algoritmisk tenkning er ikke isolerte begreper, men påvirker og påvirkes av hvordan vi ser på andre aspekter ved læring og motivasjon i fysikkfaget. Jeg tilfører det bredere fagfeltet en sammenheng mellom programmering og de teoretiske konstruktene *eierskap* og *representasjonsformer*, som jeg ikke har funnet andre steder. Mitt prosjekt er på ingen måte uttømmende, men gir et utgangspunkt for å studere programmering i lys av andre aspekter ved fysikkfaget.

Særlig anser jeg denne oppgaven for å være relevant for den norske videregående skolens fysikklærere og fysikkdidaktikere. Med den nye læreplanen kommer også et behov for å bestemme hva programmering i fysikk skal være. I tillegg til å produsere et helt konkret eksempel på et undervisningsopplegg med programmering, har jeg gitt mine innspill på hvordan programmering i fysikk kan se ut, hvorfor det er hensiktsmessig, og hvilke faktorer som er viktige å tenke på i arbeidet med utviklingen av undervisningsopplegg. Håpet er at både det presenterte opplegget og refleksjonene rundt representasjoner, eierskap, utfordringer og støttestrukturer skal være et nyttig bidrag til det norske fysikkdidaktikkmiljøet når programmering skal implementeres i fysikkfaget i løpet av de neste årene.

Implikasjoner for det norske fysikkfaget

Som jeg har beskrevet i denne oppgaven, viser det seg at det er fullt mulig å gjennomføre et enkeltstående, fungerende undervisningsopplegg med programmering i fysikk 1, selv uten at elevene har noen tidligere programmeringserfaring. Jeg har også kommentert hvordan dette er et svært sannsynlig scenario, gitt den begrensede erfaringen med programmering hos både elever og lærere. Med det som bakgrunn, og i lys av diskusjonskapittelet i denne oppgaven, har jeg følgende konkrete innspill for utforming og gjennomføring av slike aktiviteter:

- Elevene kommer ikke til å bli dyktige programmerere ved å gjennomføre ett opplegg med programmering. Derfor bør fokuset være på kognitive og affektive mål, heller enn ferdighetsmål. Programmering har et potensial for å støtte elevenes læring i fysikk, og fortelle dem noe om hvordan fysikkfaget faktisk fungerer og kunnskap konstrueres. Jeg har også indikert i denne oppgaven at programmering bør kunne være et nyttig verktøy for å fostre skaperglede og kreativitet hos elevene.
- Elevenes eierskapsfølelse styrkes når opplegget er virkelighetsnært og relevant. Det virker hensiktsmessig å modellere situasjoner som elevene kan kjenne igjen fra sin egen hverdag, eller som er lette å demonstrere i klasserommet, slik som fjærkanonforsøket i mitt opplegg.
- For å nå kognitive og affektive mål, bør syntaktiske utfordringer reduseres så mye som mulig – elevene bør fokusere på hva som skal hvor i koden, og ikke hvordan det skrives. Det betyr også at undervisningen må gi god støtte til elevenes feilsøkingsprosesser.
- Programmeringsmiljøet kan utfordre elevenes evne til å jobbe med flere representasjoner. Dersom læreren er bevisst på dette, kan oppgavesettet utformes slik at elevene støttes i hvordan de skal bruke ulike representasjoner på hensiktsmessige måter.
- Det er mange elever som klarer å skrive det de skal, men som ikke nødvendigvis skjønner det de skriver. Det er viktig at undervisning legger til rette for å forstå modelleringsprosessen og Euler-Cromer-algoritmen, slik at elevene kan kjenne at de «eier» problemløsningsprosessen.

Selv om «ett opplegg»-modellen er et sannsynlig scenario de neste årene, vil jeg slå et slag for at elevene vil få mest ut av programmering i fysikkfaget dersom de får anledning til å bruke det ofte. Programmene trenger ikke være avanserte, og de trenger heller ikke være veldig forskjellige. Dersom elevene blir vant til å beskrive bevegelse med Euler-Cromer, kan man bruke energien de før brukte på programmering, til å åpne opp for mer elevkreativitet, egne problemstillinger og egne løsninger. Om elevene vet hvordan de modellerer bevegelse, eller kanskje allerede har en mal for å skrive slik kode, så kan man se for seg enkle programmeringsaktiviteter som lar seg gjennomføre på 45 minutter.

Valg av programmeringsmiljø: Fordeler og ulemper ved Trinket.io

Det er ikke umiddelbart gitt hvilket programmeringsmiljø som er best egnet for undervisning i fysikk. Selv om denne oppgaven prøver å si noe generelt om programmeringsaktiviteter, vil jeg i dette avsnittet kort presentere noen av fordelene og ulempene ved å bruke Trinket, slik det opplevdes under gjennomføringen.

En av de umiddelbare fordelene ved Trinket er at det er lett å lage animasjoner. Med litt forberedelse fra læreren er det også svært lite krevende for elevene å lage grafer for en rekke ulike fysiske størrelser, og disse er jevnt over fleksible og gir oversiktlige, ferdig skalerte plot i sanntid. En annen fordel er at miljøet er brukervennlig, og det er lett for elever å logge inn og arbeide med sine egne og andres kodesnutter. Under gjennomføringen opplevde vi svært få vanskeligheter i forbindelse med innlogging og navigering på Trinket sine nettsider. Det lille som dukket opp var knyttet til elever som hadde glemt passord og lignende. I det store og hele opplevdes miljøet som lettvint og effektivt, og det gjorde de tingene vi trengte. Det er også verdt å nevne kursfunksjonaliteten i Trinket. Selv om kursene tar litt tid å lage, har de potensial til å gi svært gode rammer å arbeide innenfor.

Men det betyr ikke at det ikke er ting man skulle ønske var annerledes, og som på sikt kan være avgjørende for om man velger å bruke Trinket fremfor andre, lignende programmeringsmiljø. For det første er mange feilmeldinger i Trinket vanskelige å lese, særlig dersom man har lite programmeringserfaring. For det andre hendte det – riktignok svært sjelden – at elevenes programmer krasjet, slik at nettleseren måtte lukkes og startes på nytt. Dette kunne for eksempel skje dersom elevene kjørte en evig while-løkke. Det er ingen mekanismer i miljøet som lar brukeren avbryte en kode som kjører.

Kanskje den tydeligste utfordringen ved bruk av Trinket kommer til syne om man ønsker å bruke den samme koden til mange forskjellige formål, og kanskje med mange forskjellige størrelsesordener. Animasjonsvinduet i Trinket har ikke noe innebygget koordinatsystem eller andre hjelpelinjer, så alle rammene man vil at elevene skal arbeide innenfor må lages på forhånd dersom man ikke vil at elevene skal bruke tid på det (noe som frarådes på det sterkeste). Dette gjør blant annet at hvis elevene vil modellere en bevegelse av en helt annen størrelsesorden enn den animasjonsvinduet er innstilt for, kommer de ikke til å kunne se denne bevegelsen på en meningsfull måte. Det viser seg vanskelig å finne løsninger som sikrer et fleksibelt animasjonsvindu som kan brukes i mange forskjellige settinger. Derfor er det muligens også begrenset hvor mye frihet man kan gi elevene i valg av problemstilling

dersom man ikke også vil bruke tid på å lære dem å sette opp sitt eget koordinatsystem. Disse koordinatsystemene er heller ikke enkle å lage, men må konstrueres ved å sette sammen sylindere og piler på møysommelig vis. Å sette inn tall i animasjonsvinduet er svært krevende for datamaskinen, og øker kjøretiden betraktelig. Alt dette viser at selv om Trinket har et kraftfullt animasjonsverktøy med mange muligheter[9], er det potensielt tidkrevende om man vil «gjøre det skikkelig», og for elever med lite programmeringserfaring er det avgjørende at læreren har gjort tilstrekkelig forarbeid.

Kanskje skulle man etterlyst et programmeringsmiljø som Trinket, men med to forbedringer. For det første hadde det vært svært nyttig om animasjonsvinduet hadde et innebygget koordinatsystem (se for eksempel Tychos.org). For det andre hadde det vært nyttig om man klarte å legge til rette for enkle, konkrete og lettfattelige feilmeldinger som elever kan lese og forstå. I en ideell verden kunne elevene også fått tilbakemelding på fysikken de programmerer, gitt et sett med betingelser fra læreren i forkant.

Videre forskning

Denne oppgaven er kun ett av mange nødvendig bidrag til forskningen på hvordan programmering skal implementeres i det norske (og internasjonale) fysikkfaget, og det er fortsatt mange ting som er interessante å se på i forbindelse med dette temaet. Det har vist seg fruktbart å bruke skjerm- og lydopptak av elever for å studere hva de finner utfordrende, hvordan de jobber med ulike representasjoner og hvordan programmeringen bidrar til dybdelæring. Det hadde vært interessant å gjøre flere slike studier, men med nok tid til å gjøre ordentlige transkriberinger av sentrale sekvenser, og helhetlig systematisk analyse av disse. Jeg har vist hvordan noen teoretiske rammeverk for elevers utfordringer (Basu et al., 2016), arbeid med representasjoner (Ainsworth, 2006), og eierskap (Stefanou et al., 2004) kan brukes til å studere elevers arbeid med programmering, og det er potensial for å fortsette dette arbeidet. Særlig er det behov for å se nærmere på hvilke roller representasjonene i programmeringsmiljøet har for elevene, og hvordan vi utformer opplegg som best mulig legger til rette for god representasjonsbruk. Det virker som at det etter hvert begynner å utkrystallisere seg noen klare utfordringer som elever møter når de programmerer i fysikken, og det er behov for forskning som kan si enda mer om hvordan vi skal utforme aktiviteter som lar elevene fokusere på det vi vil at de skal fokusere på. Som nevnt i kapittel 7.1, er det også høyst aktuelt å undersøke nærmere hvordan programmeringsaktiviteter kan være med på

[9] Se for eksempel https://bphilhour.trinket.io/physics-through-glowscript-an-introductory-course

å forme elevers syn på fysikk og fysikere, og bidra til å nå affektive læringsmål ved å legge til rette for skaperglede og nysgjerrighet.

Det er også et behov i den norske skolen for å finne ut av hvilke programmeringsmiljø som fungerer, og hvilke som ikke gjør det. Dersom andre studier kan vise hvordan ulike programmeringsmiljø virker å fungere, kan vi på sikt gjøre sammenligninger og finne ut om noe er mer hensiktsmessig enn noe annet. Kanskje er det ikke så viktig hvilket programmeringsmiljø vi velger å bruke – det er i så fall også interessant.

I likhet med mange andre har jeg presentert et opplegg som fokuserer på mekanikk. I tillegg til å få testet det reviderte undervisningsopplegget, er det også interessant å se på andre typer programmeringsopplegg som støtter læring i for eksempel termofysikk, lyd og lys, kjernefysikk eller lignende. Dersom noen kunne produsere – og teste – eksempler på slike opplegg, kunne det være med på å utvikle undervisningen i norske klasserom. Umiddelbart tenker jeg det ville være spennende å gjøre et opplegg med Euler-Cromer-algoritmen som i større grad har til hensikt å undervise sammenhengene mellom akselerasjon, fart og posisjon, heller enn krefter og Newtons lover.

7.4 Begrensninger med denne oppgaven

Helt til slutt før jeg konkluderer, vil jeg peke på noen begrensninger med denne oppgaven. Først av alt er det verdt å nevne at jeg har valgt noen teoretiske innganger, og belyst noen aspekter ved programmering og dybdelæring. Man kunne valgt noen andre innganger, og da ville det kanskje kommet fram andre resultater. Sånn sett er oppgaven på ingen måte uttømmende eller endelig, og det er helt sikkert nyanser i datamaterialet som ikke er reflektert i presentasjonen av resultatene. Særlig gjelder dette for Captura-opptakene som ikke er analysert systematisk, men det er sannsynligvis også tilfellet for intervjudataene. Jeg vil også påpeke at selv om oppgaven prøver å si noen ting om programmering i fysikk på et generelt plan, er utvalget for datainnsamling ganske lite, og det er ikke nødvendigvis én til én-generaliserbarhet til alle andre situasjoner. Oppgaven indikerer kun tendenser, og ikke «harde fakta».

Selv om oppgaven har som hensikt å bidra til debatten rundt det norske fysikkfaget, erkjenner jeg at programmering per i dag er svært nytt, og at situasjonen godt kan ha endret seg en god del om noen år. Det er ikke gitt at alle funnene i oppgaven er relevante eller nyttige dersom vi

på sikt møter elevgrupper i fysikkfaget som har klart mer programmeringskompetanse enn elevene i mitt prosjekt, som møtte programmering mer eller mindre for første gang.

8. Konklusjon

Inspirert av programmeringens fremmarsj i de nye læreplanene i norsk skole, har utgangspunktet for denne oppgaven vært problemstillingen: *Hvordan kan en programmeringsaktivitet i fysikk 1 bidra til dybdelæring for elevene?* I resultat- og diskusjonskapitlene har jeg belyst problemstillingen ved å se på elevenes læring, utfordringer, eierskap og bruk av representasjonsformer. Jeg har først argumentert for at vi først og fremst bør sikte på dybdelæring av kognitive og affektive læringsmål, det vil si fysikkfaglige konsepter (i mitt tilfelle Newtons andre lov, luftmotstand og mekanisk energi), et innblikk i hvordan fysikk fungerer på forskningsfronten og positive holdninger til faget. Deretter har jeg vist at en programmeringsaktivitet kan bidra til denne typen dybdelæring dersom:

- Elevene får lov å bruke overskuddet sitt på å arbeide med fagspesifikke utfordringer, utfordringer knyttet til initialbetingelser, parametere og underliggende fysiske prosesser, og semantiske utfordringer som har å gjøre med programmets struktur og innhold. Det innebærer at de må få mest mulig støtte i forbindelse med utfordringer knyttet til syntaks og feilsøking. Disse utfordringene tar vekk fokuset fra fysikken, og virker å være hemmende for dybdelæringsprosesser.
- Elevene får nok støtte til å benytte seg av de ulike representasjonsformene de møter i arbeid med oppgavene. Vi ser at representasjonene i programmet støtter elevenes læring, fordi de kan bruke koden til å enkelt variere parametere og se effekten av disse i de grafiske resultatene. Flere uttrykker at dette hjalp med å forstå sammenhengen mellom ulike deler av formlene de bruker. I tillegg virker det som at elevene har stort læringsutbytte av prosessene knyttet til å skrive programmeringskoden, særlig i arbeidet med Newtons andre lov.
- Elevene får tilstrekkelig støtte i programmeringsprosessen slik at de kan få kognitivt eierskap til det de holder på med. Oppgaven har vist at problemstillingen med luftmotstand oppleves relevant og virkelighetsnær for elevene, men flere sier de ikke får eierskap til aktiviteten likevel, fordi de ikke føler de har kommet frem til svaret selv. Det er avgjørende at elevene får støtte til å løse oppgavene på egen hånd, heller enn å kopiere kode fra oppgavearket.

Jeg håper å ha vist at programmeringsaktiviteter har potensial for å være et svært positivt bidrag til det norske fysikkfaget. I tillegg til å være en støtte for elevenes fysikkforståelse, gir

de et bredere bilde av hvordan fysikk kan foregå, og de kan la elevene bruke kunnskapen de har til å etablere ny kunnskap. Programmeringsmiljøet gir en ny og tilsynelatende god arena for å trene elevenes representasjonsbruk og la dem se sammenhenger mellom ulike representasjoner, og særlig kan kodevinduet bidra til at elevene må strukturere sin egen fysikkunnskap i større grad enn tidligere. Jeg har også pekt på potensialet som ligger i programmeringens fleksibilitet, og tatt til orde for at man på sikt kan bruke programmering til å la elevene utforske egne problemstillinger og hypoteser. Alt dette virker å bidra positivt til elevenes dybdelæringsprosesser.

Denne oppgaven gir to viktige bidrag til norsk skolefysikk og fysikkdidaktisk forskning. For det første gir den en enkeltstående programmeringsaktivitet som er klar til bruk i fysikk 1. For det andre bidrar den med å belyse noen generelle aspekter ved programmering og dybdelæring, og gir noen anbefalinger for planleggingen og gjennomføringen av programmeringsaktiviteter i norsk skolesammenheng, basert på den presenterte analysen. Jeg har også vist noen mulige tilnærminger for å arbeide med forskning på programmering i fysikkfaget, men erkjenner at problemstillingen åpner for mange ulike tilnærminger. Det er behov for mer forskning som kan si noe om hvilke dybdelæringseffekter som ligger i programmeringsaktiviteter, og hvordan vi best fostrer disse i praksis.

9. Referanser

Aho, K., Chandra, K. & Roberts, E. (2014). Introducing programming into the physics curriculum at Haverhill High School using the R language. *Proc. Am. Soc. Eng. Educ*.

Aiken, J. M. (2013). Transforming high school physics with modeling and computation. *arXiv preprint arXiv:1310.3725*.

Ainsworth, S. (2006). DeFT: A conceptual framework for considering learning with multiple representations. *Learning and instruction, 16*(3), 183-198.

Angell, C., Bungum, B., Henriksen, E. K., Kolstø, S. D., Persson, J. & Renstrøm, R. (2019). *Fysikkdidaktikk* Cappelen Damm.

Basu, S., Biswas, G., Sengupta, P., Dickes, A., Kinnebrew, J. S. & Clark, D. (2016). Identifying middle school students' challenges in computational thinking-based science learning. *Research and Practice in Technology Enhanced Learning, 11*(1), 13.

Befring, E. (2015). *Forskningsmetoder i utdanningsvitenskap* Cappelen Damm Akademisk.

Bott, T., Weller, D. P., Irving, P. W. & Caballero, M. D. (2019). Student-Identified Themes Around Computation in High School Physics. *Bulletin of the American Physical Society, 64*.

Braun, V. & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology, 3*(2), 77-101.

Caballero, M. D., Burk, J. B., Aiken, J. M., Thoms, B. D., Douglas, S. S., Scanlon, E. M. & Schatz, M. F. (2014). Integrating numerical computation into the modeling instruction curriculum. *The Physics Teacher, 52*(1), 38-42.

Caballero, M. D., Fisler, K., Hilborn, R. C., Romanowicz, C. M. & Vieyra, R. (2020). *Conference Report*. Innlegg presentert ved Advancing Interdisciplinary Integration of Computational Thinking in Science.

Caballero, M. D., Kohlmyer, M. A. & Schatz, M. F. (2012). Implementing and assessing computational modeling in introductory mechanics. *Physical review special topics-physics education research, 8*(2), 020106.

Chabay, R. W. & Sherwood, B. A. (2015). *Matter and interactions* John Wiley & Sons.

Chin, C. & Brown, D. E. (2000). Learning in science: A comparison of deep and surface approaches. *Journal of Research in Science Teaching: The Official Journal of the National Association for Research in Science Teaching, 37*(2), 109-138.

Creswell, J. W. & Miller, D. L. (2000). Determining validity in qualitative inquiry. *Theory into practice, 39*(3), 124-130.

Dalen, M. (2011). Intervju som forskningsmetode [Interview as a research method]. *Oslo: Universitetsforlaget*.

Deci, E. L., Vallerand, R. J., Pelletier, L. G. & Ryan, R. M. (1991). Motivation and education: The self-determination perspective. *Educational psychologist, 26*(3-4), 325-346.

Enghag, M. & Niedderer, H. (2008). Two dimensions of student ownership of learning during small-group work in physics. *International Journal of Science and Mathematics Education, 6*(4), 629-653.

Firebaugh, G. (2018). *Seven rules for social research* Princeton University Press.

Gravel, B. E. & Wilkerson, M. H. (2017). Integrating computational artifacts into the multi-representational toolkit of physics education. I *Multiple representations in physics education* (s. 47-70). Springer.

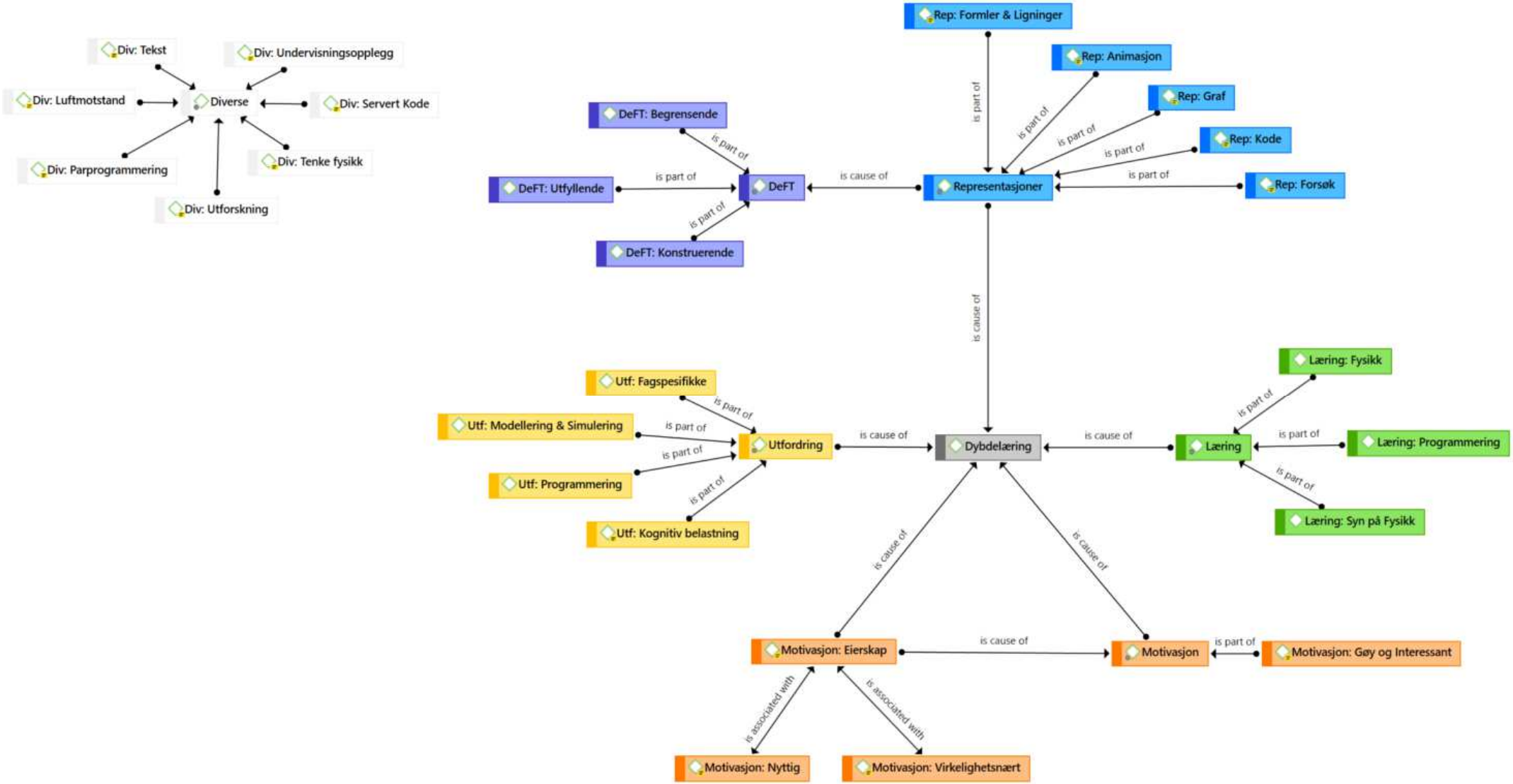
Grover, S. & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher, 42*(1), 38-43.

Guzdial, M. (1994). Software-realized scaffolding to facilitate programming for science learning. *Interactive learning environments, 4*(1), 001-044.

Haraldsrud, A. D. & Tellefsen, C. W. (2018). Programmering - for fysikkens skyld. *Fra Fysikkens Verden*, (3).
Johnson, B. R. (2013). Valditity of Research Results in Quantitative, Qualitative and Mixed Research.(kapittel 11) I BR Johnson & L. Christensen. *Educational Research: Quantitative, Qualitative, and Mixed Approaches*, 277-316.
Judd, C. M., Smith, E. R. & Kidder, L. H. (1991). Research methods in social relations. 1991. I: Fort Worth, TX: Holt, Rinehart and Winston, Inc.
Kleven, T. A., Hjardemaal, F. & Tveit, K. (2011). *Innføring i pedagogisk forskningsmetode: En hjelp til kritisk tolkning og vurdering* Unipub.
Kohl, P. B. & Finkelstein, N. (2017). Understanding and promoting effective use of representations in physics learning. I *Multiple Representations in Physics Education* (s. 231-254). Springer.
Kohl, P. B. & Finkelstein, N. D. (2008). Patterns of multiple representation use by experts and novices during physics problem solving. *Physical review special topics-physics education research, 4*(1), 010111.
Krueger, R. A. (2014). *Focus groups: A practical guide for applied research* Sage publications.
Kunnskapsdepartementet. (2016). *Fag - Fordypning - Forståelse. En fornyelse av Kunnskapsløftet*. (Meld. St. 28, 2015-2016). Hentet fra https://www.regjeringen.no/contentassets/e8e1f41732ca4a64b003fca213ae663b/no/pdfs/stm201520160028000dddpdfs.pdf
Landau, R. (2006). *Computational Physics: A Better Model for Physics Education?* College Park, Md. :.
Linn, M. C. & Eylon, B.-S. (2011). *Science learning and instruction: Taking advantage of technology to promote knowledge integration* Routledge.
Malthe-Sørenssen, A., Hjorth-Jensen, M., Langtangen, H. P. & Mørken, K. (2015). Integrasjon av beregninger i fysikkundervisningen. *Uniped, 38*(04), 303-310.
Mayer, R. & Mayer, R. E. (2005). *The Cambridge handbook of multimedia learning* Cambridge university press.
McDowell, C., Werner, L., Bullock, H. & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. *SIGCSE Bull., 34*(1), 38–42. https://doi.org/10.1145/563517.563353
Meyer, C., Guenther, L. & Joubert, M. (2019). The Draw-a-Scientist Test in an African context: comparing students'(stereotypical) images of scientists across university faculties. *Research in Science & Technological Education, 37*(1), 1-14.
Nordby, S. T. (2019). *Programmering og algoritmisk tenkning i fysikkundervisning* NTNU.
Opfermann, M., Schmeck, A. & Fischer, H. E. (2017). Multiple representations in physics and science education–why should we use them? I *Multiple representations in physics education* (s. 1-22). Springer.
Savery, J. R. & Duffy, T. M. (1995). Problem based learning: An instructional model and its constructivist framework. *Educational technology, 35*(5), 31-38.
Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G. & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies, 18*(2), 351-380.
Serbanescu, R. M., Kushner, P. J. & Stanley, S. (2011). Putting computation on a par with experiments and theory in the undergraduate physics curriculum. *American Journal of Physics, 79*(9), 919-924.
Sevik, K. (2016). *Programmering i skolen*. Senter for IKT i utdanningen.
Smith, T. W. & Colby, S. A. (2007). Teaching for deep learning. *The Clearing House: A Journal of Educational Strategies, Issues and Ideas, 80*(5), 205-210.

Stefanou, C. R., Perencevich, K. C., DiCintio, M. & Turner, J. C. (2004). Supporting autonomy in the classroom: Ways teachers encourage student decision making and ownership. *Educational psychologist, 39*(2), 97-110.
Stroupe, D. (2014). Examining classroom science practice communities: How teachers and students negotiate epistemic agency and learn science-as-practice. *Science Education, 98*(3), 487-516.
Sweller, J., Van Merrienboer, J. J. & Paas, F. G. (1998). Cognitive architecture and instructional design. *Educational psychology review, 10*(3), 251-296.
Säljö, R. (2016). *Læring-en introduksjon til perspektiver og metaforer* Cappelen Damm Akademisk.
Sørby, S. A. & Angell, C. (2012). Undergraduate students' challenges with computational modelling in physics. *Nordic Studies in Science Education, 8*(3), 283-296.
Taub, R., Armoni, M., Bagno, E. & Ben-Ari, M. M. (2015). The effect of computer science on physics learning in a computational science environment. *Computers & Education, 87*, 10-23.
Utdanningsdirektoratet. (2006). Læreplan i fysikk - programfag i utdanningsprogram for studiespesialisering. Hentet fra http://data.udir.no/kl06/FYS1-01.pdf
Utdanningsdirektoratet. (2017). Forsøkslæreplan i programmering og modellering X - programfag i utdanningsprogram for studiespesialisering. Hentet fra http://data.udir.no/kl06/PRM1-01.pdf
Utdanningsdirektoratet. (2019a). Algoritmisk Tenkning. Hentet 30.03 2020 fra www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/
Utdanningsdirektoratet. (2019b, 13.03.2019). Dybdelæring. Hentet 13.04 2020 fra https://www.udir.no/laring-og-trivsel/dybdelaring/
Utdanningsdirektoratet. (2019c). Læreplan i naturfag Hentet fra https://data.udir.no/kl06/v201906/laereplaner-lk20/NAT01-04.pdf
Utdanningsdirektoratet. (2020, 27. februar). Læreplan i fysikk (Utkast). I. Hentet fra https://hoering.udir.no/Hoering/v2/960
Voll, L. O., Øyehaug, A. B. & Holt, A. (2019). *Dybdelæring i naturfag*. Oslo: Universitetsforlaget.
Vygotsky, L. S., Cole, M., John-Steiner, V., Scribner, S. & Souberman, E. (1978). The development of higher psychological processes. *Mind in society*, 1-91.
Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127-147.
Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.
Ytterhaug, A. S. (2015). *Hvordan forstå at jorda går i en rettlinjet bevegelse rundt sola-Forskningsbasert utvikling og utprøving av undervisningsmodul i generell relativitetsteori*.

Vedlegg A: Oversikt over koder fra tematisk analyse av intervjuer



Kategori	Kode	Beskrivelse	Eksempel	Antall tilfeller
Representasjoner	Rep: Formler & Ligninger	Når elever omtaler regning for hånd, formler og ligner.	"Ja, det ga mening, fordi vi måtte jo, eller, vi måtte i hvert fall gjøre om formlene for hånd da (…)"	28
	Rep: Kode	Når elever omtaler det som skjer i kodevinduet, hvor kode leses og skrives.	"Jeg synes det var liksom bra at vi skrev ned alle variablene, og så bytta vi på dem da (…)"	86
	Rep: Animasjon	Når elever omtaler animasjonen i Trinket.	"Uten animasjonen, så tror jeg vi ikke hadde skjønt så veldig mye:"	56
	Rep: Graf	Når elever omtaler grafen i Trinket.	"Og jo mindre den er, jo mer nøyaktig blir det, på en måte. Den grafen vi for eksempel får opp da."	26
	Rep: Forsøk	Når elever omtaler forsøket med fjærkanonen.	"Også likte jeg at du på en måte gikk gjennom, typ, forsøket praktisk før (…)"	5
DeFT	DeFT: Begrensende	Når elevene implisitt uttrykker at en representasjon begrenset en annen.	"Jeg vil si at det er lettere å se hva vi holder på med med pgorammering, fordi da ser man faktisk ballen."	5
	DeFT: Utfyllende	Når elevene implisitt uttrykker at to representasjoner utfylte hverandre.	"Også kunne du jo se ballens posisjon selv om den ikke var inne i framea (…)"	26
	DeFT: Konstruerende	Når elevene implisitt uttrykker at de har jobbet med sammenhengen mellom ulike representasjoner.	"(…) man så på den grafen også kunne man se (…) hvordan grafen endra seg da, hvis man endra en annen verdi, så endra grafen seg igjen."	16
Motivasjon	Mot: Eierskap	Når elevene uttrykker at deler av undervisningsaktiviteten har hatt en effekt på eierskap, implisitt eller eksplisitt.	"Jeg synes det var gøy når vi skulle liksom lage vår egen ball (…)"	30
	Mot: Virkelighetsnært	Når elevene uttrykker at undervisningsaktiviteten opplevdes som virkelighetsnær.	"Også lage en simulering, som er liksom tilnærmet virkelig i hvert fall (...)"	22
	Mot: Nyttig	Når elevene uttrykker at programmering i fysikk er nyttig, implisitt eller eksplisitt.	"Nei, altså, man ser jo nytten ved digitale verktøy, i sammenheng med fysikk."	28

	Mot: Gøy & Interessant	Når elevene uttrykker at programmeringsaktiviteten var gøy, interessant, spennende e.l.	"Jeg synes det var interessant at hvis du hadde på luftmotstand så kunne du øke farten med hundre, også økte bare (…)"	58
	Mot: Variasjon	Når elevene uttrykker at programmeringsaktiviteten bidro med variasjon til fysikkfaget.	"Ja, det var deilig som et avbrekk fra vanlig tavleundervisning (…)"	20
Utfordring	Utf: Fagspesifikke	Når elevene omtaler fagspesifikke utfordringer de møtte i arbeidet med aktiviteten.	"Jeg synes det var vanskelig å finne formelen for akselerasjonen (…)"	12
	Utf: Modellering & Simulering	Når elevene omtaler utfordringer knyttet til modellering og simulering de møtte i arbeidet med aktiviteten.	"Vi glemte at ballen skulle ned også."	17
	Utf: Programmering	Når elevene omtaler utfordringer knyttet til programmering de møtte i arbeidet med aktiviteten.	"Mye parentesbruk også, så er det litt sånn, vanskelig å få kontroll på."	27
	Utf: Kognitiv Belastning	Når elevene omtaler deler av aktiviteten som "mye", "vanskelig", "viktig å være nøye" eller lignende, som tyder på stor kognitiv belastning.	"Jeg synes fysikk sånn generelt er ganske vanskelig, så (…) å putte inn en faktor til da, (…) ville gjort det enda vanskeligere."	34
Læring	Læring: Fysikk	Når elevene uttrykker at de har lært noe om fysikkspesifikke begreper og konsepter.	"Jeg forsto bedre konseptet med k i luftmotstanden."	34
	Læring: Programmering	Når elevene uttrykker at de har lært noe om programmering.	"Jeg har lært litt om hvordan programmering funker (…)"	39
	Læring: Syn på Fysikk	Når elevene uttrykker at de har lært noe om hva fysikkfaget er, eller at de har endret sitt syn på fysikkfaget og/eller fysikere.	"Man så jo litt hva man på en måte kan drive med innen fysikk da, at man kan programmer og sånne ting."	19

Diverse	Div: Luftmotstand	Når elevene snakker om luftmotstand, formelen (eller deler av formelen) for luftmotstand	"At den virker motsatt retning av fartsretningen."	44
	Div: Parprogrammering	Når elevene snakker om parprogrammering	"Jeg tror det hadde vært vanskelig hvis vi hadde jobbet én og én."	18
	Div: Servert Kode	Når elevene omtaler kode som var gitt i oppgavesettet.	"Jeg skjønte jo at det var det som skapte en pil, men jeg skjønte ikke helt hvorfor det, altså hele den lange koden."	23
	Div: Tekst	Når elevene snakker om oppgaveteksten de fikk utdelt, og omtaler utforming, ordlyd etc.	"Jeg leste mer sånn skumlesing, for jeg skjønte ikke helt hva det var. Egentlig."	16
	Div: Tenke Fysikk	Når elevene uttrykker at de tenkte på fysikk mens de jobbet med opplegget.	"Altså, jeg synes ikke jeg tenkte så mye på fysikk hele tiden, men (…) Og det var da fysikken begynte å komme mer frem."	23
	Div: Undervisningsopplegg	Når elever omtaler formatet på undervisningen, f.eks. hvordan introduksjonskurset fungerte, sekvensene hvor læreren underviste, eller omfang av oppgaver.	"Jeg synes det er bra når lærere gjør det på tavla på en måte, og du også kan gjøre det samtidig (…)"	49
	Div: Utforskning	Når elevene uttrykker at de brukte programmet til å justere ulike verdier og se hvordan de påvirket resultatene.	"Man kunne se de forskjellige endringene av funksjoner når man skrev inn forskjellige verdier."	19

Vedlegg B: Intervjuguide

Intervjuguide: Programmering med Glowscript i Fysikk 1

Intro
Tusen takk for at dere vil være med i dette intervjuet. Jeg har tenkt å bruke informasjonen jeg får her til å skrive masteroppgaven min, som skal handle om programmering i fysikkfaget. Dere kommer ikke til å kunne bli kjent igjen i oppgaven jeg skriver, der står det ingenting om hvem dere er, hvilken skole dere går på, eller hvor dere bor. Det dere sier her har ingenting å si for hvilken karakter dere får i faget. Læreren deres har ikke tilgang til opptaket av intervjuet.
For at jeg skal klare å huske hva dere sier, tar jeg lydopptak av intervjuet. Planen er å skrive ned hva dere sier så fort som mulig, på en slik måte at dere ikke kan kjennes igjen. Lydopptakene slettes ved prosjektets slutt, i desember 2020. Frem til det er det kun jeg, veilederne mine og et fåtall andre ansatte på universitetet som vil ha tilgang.

Dere er her på frivillig basis. Hvis dere finner ut nå – eller senere i intervjuet – at dere ikke vil være med likevel, kan dere si ifra om det, og gå ut av rommet.

- **Bakgrunn**
    - Hva synes dere om fysikkfaget så langt?
        - Er det lett? Vanskelig
        - Hva er gøy? Hva er mindre gøy?
    - Hvor mye erfaring har dere med programmering fra før?
    - Hvordan fungerte intro-oppgavene som utgangspunkt for å begynne på oppgavene om luftmotstand?
- **Elevenes opplevelse av programmering i fysikk**
    - Nå skal vi snakke litt om hva dere lærte eller ikke lærte i løpet av økta med programmering. Hva husker dere best fra opplegget?
        - Hva er én ting dere forstår bedre nå enn dere gjorde før vi hadde økten?
        - Fikk dere noen aha-opplevelser? I så fall, hvilke?
        - Hva visste dere om luftmotstand før økten med programmering?
            - Har dere lært noe om luftmotstand ila økten? Eventuelt hva?
    - Hvor krevende var det å forstå fysikken i opplegget? Hvor krevende var selve programmeringen?
        - Hvor mye tid brukte dere på det ene og det andre?
    - Hva synes dere om å lære og bruke programmering i fysikktimene?
        - Hva var lett?
        - Hva var vanskelig?
        - Hva var gøy? Eller kjedelig?
        - Hva synes dere om å programmere fremfor å regne med ligninger for hånd?
    - Hvordan fungerte det å jobbe i par?

  Et av målene med masteroppgaven min er å lage et oppgavesett som kan brukes flere ganger, og derfor trenger jeg tilbakemelding på hva som fungerte og ikke.
    - Hvordan opplevde dere å jobbe med oppgavearket?
        - Fikk dere nok informasjon? For mye? (ble det kjedelig og lite utfordrende?) For lite?
        - Var det for mye eller for lite tekst?

        - Fikk dere den hjelpen dere trengte hvis dere sto fast underveis? Fra hvor eller hvem fikk dere hjelp i så fall?
    - Eksempeloppgave: Hva skjer her? (While-løkke med oppdatering av en variabel)
    - Rakk dere å begynne på ekstraoppgavene? Hvordan var det å jobbe med disse?

- **VPython spesielt**

En av utfordringene med programmering i skolen, er at elevene ikke alltid kan nok programmering til å gjøre alt man ønsker. Én mulig løsning på det, er å gjøre oppgavene enklere. En annen er å gi kodelinjer som elevene skal kopiere inn i sin egen kode.

- Hvordan opplevde dere å skulle ignorere deler av koden dere fikk utdelt? Eller å bare kopiere kodelinjer (ref animert pil)?
    - Er det viktig å forstå alle linjene i et program før man bruker koden?
- Hvor godt forsto dere koden dere skrev? I hvor stor grad kopierte dere eksemplene?

Ikke all programmering foregår med animasjoner, så jeg ønsker å finne ut av hva animasjonen gjør med programmeringsaktiviteten.

- Var animasjonen en hjelp for å forstå programmet?
    - Hva slags informasjon fikk dere fra animasjonen i programmet?
- Var animasjonen en hjelp for å forstå fysikken?
- Hvis du har programmert før, hvordan synes du dette var til sammenligning?

- **Programmering i fysikk**
    - Hvordan har arbeid med programmering påvirket deres syn på fysikk?
        - Hvilket bilde hadde dere av en "gjennomsnittlig" fysiker/forsker før programmeringsøkten?
        - Har dere et annet bilde nå, eller er det det samme?
        - Tror dere at dere kommer til å bruke programmering i fysikk eller lignende fag i videre utdanning?
    - Hvordan påvirker programmering motivasjonen din for fysikkfaget?
    - Jeg ønsker å finne ut av hvordan elever og lærer kan arbeide med programmering i fysikkfaget for at dere skal lære mest mulig og bli motivert til å lære mer. Har dere noen tanker om dette? *(godt mulig dette blir besvart i løpet av intervjuet, faktisk)*

    - Er det noe mer dere har lyst å si om programmering i fysikk?

Vedlegg C: Læringsspørsmål til elevene

Helt til slutt

Som del av tilbakemelding på undervisningsopplegget, ønsker vi at dere svarer kort på disse to spørsmålene:

- Hvordan likte du å jobbe med programmering i fysikkfaget?

- Hva synes du at du har lært av å jobbe med programmering i fysikk?

Vedlegg D: Info- og samtykkeskriv

TIL ELEVER:
Invitasjon til å delta i forskningsprosjektet ***Programmering i fysikk***

Formål med prosjektet
I 2020-2022 kommer det nye læreplaner som beskriver hva elever i Norge skal kunne i de ulike fagene. I denne læreplan-reformen innføres programmering i realfagene – også i programfaget fysikk. Vi ønsker å undersøke hvordan programmering i Fysikk 1 best kan organiseres for at elever skal få både motivasjon og læringsutbytte. I skoleåret 2019-2020 utarbeider masterstudentene Andreas Fagerheim og Jonathan B. Waters programmeringsaktiviteter i fysikk som vi ønsker å prøve ut i din klasse. Dette er avtalt med læreren din og vil inngå i fysikkundervisningen for klassen.

Hva innebærer det å delta i undersøkelsen?
Vi ønsker å observere klassen mens dere arbeider med programmering i Fysikk 1, og vi ønsker tilgang til dine skriftlige svar på et oppgavesett knyttet til programmeringen. I tillegg ønsker vi å bruke programvaren «Captura» til å gjøre opptak av skjermbildet samt diskusjonen (lyd) når grupper av 2-3 elever jobber sammen om programmeringsoppgaver på en bærbar datamaskin. Vi vil også invitere noen elever til små gruppediskusjoner etter undervisningen om hvordan dere opplevde å arbeide med programmerings-aktivitetene. Disse diskusjonene ønsker vi å gjøre lydopptak av. De som deltar i gruppediskusjonene, får en liten oppmerksomhet til takk (antakelig i form av matservering under diskusjonen).

Hva skjer med informasjonen om deg?
Forskningsgruppa vil behandle alle personopplysninger konfidensielt og vil bare benytte de innsamlede opplysningene til forskningsformål. Forskergruppa vil ikke på noen måte bidra til vurdering av deg som elev, og din fysikkarakter vil ikke ha sammenheng med hva vi finner ut. Ingen andre enn forskergruppa vil ha tilgang til informasjonen vi lagrer, og du vil ikke kunne gjenkjennes i noen rapporter fra prosjektet. Innen desember 2020 vil alle lyd- og skjermopptak og alt skriftlig materiale der du kan identifiseres med navn eller på annen måte, slettes eller anonymiseres. Innsamlede opplysninger som er anonymisert, kan lagres også etter desember 2020 med tanke på oppfølgingsstudier. Som forskere forholder vi oss til etiske regler om lagring og bruk av personopplysninger. NSD har fått melding om prosjektet og har anbefalt at det kan gjennomføres som beskrevet her.

Frivillig deltakelse – dine rettigheter
Det er frivillig å delta i undersøkelsen. Klassen din vil uansett gjennomføre programmeringsaktiviteten og få besøk av forskere fra prosjektet, men data om deg vil bare samles inn til forskningsprosjektet dersom du gir samtykke. Hvis du *ikke* samtykker til å delta i undersøkelsen, vil bare læreren ha tilgang til arbeidene dine, som vanlig, og du vil ikke bli bedt om lyd- eller skjermopptak.

Dersom du deltar i undersøkelsen, og så lenge du kan identifiseres i datamaterialet vårt, har du rett til å:

- når som helst trekke ditt samtykke uten å oppgi noen grunn. Hvis du trekker deg, vil alle opplysninger om deg bli slettet eller anonymisert
- få tilgang til, endre, eller slette all informasjon registrert om deg
- begrense bruken vår av dine personopplysninger
- få utlevert en kopi av de personopplysninger vi har om deg
- klage til UiOs personvernombud eller til Datatilsynet om behandlingen av dine personopplysninger.

Kontaktinformasjon prosjektledere: Jonathan B. Waters, jonathbw@student.uv.uio.no, Ellen K. Henriksen, e.k.henriksen@fys.uio.no, Tor Ole Odden, t.o.odden@fys.uio.no, Personvernombudet UiO, personvernombud@uio.no

Samtykke til deltagelse i forskningsprosjektet ***Programmering i fysikk*** i regi av Fysisk institutt ved Universitetet i Oslo

Jeg har mottatt og forstått informasjon om forskningsprosjektet ***Programmering i fysikk***, og har fått anledning til å stille spørsmål. Jeg er villig til å delta i prosjektet.

Dato:

Signatur:

Vedlegg E: Originalt oppgavesett m/ ekstraoppgaver

Oppgaveark: Vertikalt kast med programmering i VPython

Læringsmål: Når eleven er ferdig med denne økten, skal han/hun

- Forstå prinsippene bak enkel programmering, og bruke disse til å utforske begreper i mekanikk ved hjelp av VPython
    - Variabler, løkker, tidssteg, print til terminal
- Forstå akselerasjon som en fartsendring fra et lite tidssteg til et annet
    - (Og tilsvarende fart som en posisjonsendring fra et lite tidssteg til et annet)
- Kunne bruke en løkke til å undersøke bevegelse over tid, med og uten konstant akselerasjon
- Kunne bruke programmering til å undersøke fenomener som friksjon og luftmotstand
- Ha noe innsikt i hvordan fysikere jobber med å modellere fysiske fenomener.

Forutsetninger: Før eleven begynner på denne oppgaven, bør han/hun

- Ha gjort seg noe erfaring med grunnleggende programmering
    - Definere variabler, regne med disse
    - Oppdatering av variabler, x = x + 1
- Ha sett og hørt om hvordan en while-løkke fungerer, og brukt en slik løkke til enkle formål, f.eks. printe partall
- Ha fått undervisning i hvordan man regner på bevegelse med v = v + a*dt, s = s + v*dt

Først og fremst

Dere skal jobbe to og to. Begge elevene på hver gruppe bør ha sin egen datamaskin. Mens dere jobber med oppgaven skal én av dere jobbe med koden på sin datamaskin, mens den andre skriver svar her i oppgavearket på sin datamaskin. Underveis bytter dere plass, sånn at begge to får programmert, og begge får skrevet.

Dere skal skrive inn svar direkte i dette oppgavearket. Dere skal svare på oppgavene som er nummerert med a), b), c) også videre, i de røde rammene. I tillegg skal dere skrive inn link til koden deres øverst i oppgavearket, sånn at læreren kan se den.

Hensikt

Tidligere har vi gjort et forsøk hvor vi skjøt en ball opp i luften, og undersøkte hvor høyt den kom. Vi undersøkte om den mekaniske energien var bevart, og så at den ikke var fullstendig bevart, fordi luftmotstanden bremset ballens bevegelse. I denne oppgaven skal vi bruke programmering til å modellere en ball som blir skutt opp i luften.

Hensikten med oppgaven er å finne en sannsynlig verdi for luftmotstandskoeffisienten *k*. Denne koeffisienten forteller oss hvor stor luftmotstanden for et objekt er, og avhenger av objektets form, størrelse og overflate.

Fremgangsmåte

- Begynn med å følge denne linken: https://trinket.io/library/trinkets/3fc543c2a0
- Du må lagre koden sånn at du ikke mister endringene dine.
    - Trykk på lagreknappen over koden (den heter "remix"). Fyll inn e-postadresse, finn på et passord, og trykk "Create Account".
    - Nå skal det dukke opp en grønn knapp som heter "Remix". Trykk på den
    - Nå har du kopiert koden til din bruker. Du kan endre som du vil, og så lagre endringene ved å trykke på lagreknappen.

Trykk på "dele"-knappen over vinduet, og vel "Link" i menyen som kommer opp. Kopier linken til koden din her:

Til venstre for Trinket-logoen over koden, er det en knapp med tre streker. Trykk på den, og deretter på "Fullscreen" – da blir både koden og resultatene lettere å lese

- Ignorer den øverste delen av koden, som er rammet inn med ####. Her defineres blant annet koordinatsystemet og andre ting som er kronglete og tar tid

Bevegelse uten luftmotstand

Vi skal bruke en while-løkke til å undersøke bevegelsen til en ball som kastes oppover i luften. Først gjør vi beregninger uten luftmotstand, for å sjekke at programmet virker. Hvis du vet hvordan du gjør dette og/eller vil prøve på egen hånd, kan du det. Hvis du er mer usikker på hva du skal gjøre, kan du følge instruksjonene under:

- **Initialbetingelser:** Dette er kode som beskriver startsituasjonen, og andre konstante størrelser. Se for eksempel bildet på neste side.
- Først av alt må vi lage en ball. Kall den f.eks. "Ball". Plasser ballen i posisjonen (0,0,0), og gi den en farge du liker, ved å skrive f.eks. `color=color.blue.` Husk at radiusen til ballen er 0.16m.
- Deretter må vi definere noen initialbetingelser og konstanter:
    - Tyngdeakselerasjonen g = -9.81 m/s^2
    - Ballens masse m = 0.018 kg
    - Ballens startfart v - velg startfarten dere målte i forsøket med fjærkanonen
    - Start-tiden t = 0
    - Størrelsen på tidssteget, dt = 0.01s

```
28  #Initial Conditions
29  Ball = sphere(pos=vec(0,0,0),radius=0.16,color=color.red)
30  v = 4.5 # m/s
31  g = -9.81 #m/(s*s)
32  m = 0.018 #kg
33
34  t = 0 #s
35  dt = 0.01 #s
```

- **While-løkke:** I denne løkken beregnes ballens bevegelse for hvert tidssteg, frem til kriteriet for løkken ikke lenger tilfredsstilles. Husk at alt som skrives inne i løkken må ha ett innrykk (tab-knappen på tastaturet)
- Start while-løkken ved å skrive: `while Ball.pos.y >= 0:` . Dette forteller oss at løkken repeteres så lenge ballen er over y = 0.
- På neste linje skriver du `rate(700)` - dette bestemmer hastigheten på animasjonen din.
- For hvert tidssteg må vi vite hva akselerasjonen skal være. Bruk en linje på å definere akselerasjonen til ballen. Pass på at akselerasjonen får riktig fortegn.

```
36  #While Loop
37
38  while Ball.pos.y >= 0:
39    rate(700) #animation speed
40
41    a =
42    v =
43    Ball.pos.y =
44    t =
```

- Så oppdaterer vi farten, og deretter posisjonen.
  - Vi husker at

$$a = \frac{v - v_0}{\Delta t}$$

Det vil si at formelen for *v* etter et tidssteg Δ*t* med akselerasjon *a* er:

$$v = v_0 + a \cdot \Delta t$$

Bruk denne formelen for å oppdatere farten inne i løkken. I koden skriver vi *dt* i stedet for Δ*t*. Husk at du kan skrive f.eks. *v = v + 1*, hvor *v* etter likhetstegnet er den gamle verdien, og *v* før likhetstegnet er den nye verdien.

  - På samme måte er

$$v = \frac{s - s_0}{\Delta t}$$

Hva er formelen for y-posisjonen etter et tidssteg *dt* med hastighet *v*? Skriv inn denne. Husk at Y-posisjonen til ballen er gitt ved `Ball.pos.y`

- Til slutt i løkken må vi huske å oppdatere tiden. Det gjør du ved å legge til et tidssteg *dt* til *t*:

$$t = t + dt$$

- Hvis du nå trykker play, bør du se at ballen flyr opp i været, for så å snu og dette ned igjen. Dette er et godt tidspunkt å sjekke at du ikke har gjort feil i koden. Det er også et bra tidspunkt for å justere hastigheten på animasjonen, ved å bruke `rate`-parameteren.

Analyse av resultatene

- Vi ønsker å få greie på nøyaktig hvor høyt opp ballen kommer. For å gjøre det kan du legge inn denne linjen øverst i løkken din (men under `rate`-linjen): `y.plot(t,Ball.pos.y)` Denne linjen plotter ballens y-posisjon som funksjon av tiden.

```
36 #While Loop
37
38 while Ball.pos.y >= 0:
39   rate(700) #animation speed
40
41   a =
42   v =
43   Ball.pos.y =
44   t =
```

a) Nå kan du lese av ballens høyde ved å plassere musepekeren over grafen. Hvor høyt over utgangspunktet kommer den?

b) Bruken ligningen for bevaring av mekanisk energi til å regne ut hvor høyt ballen kommer dersom den mekaniske energien er bevart. Hvor godt stemmer resultatet fra programmeringen med verdien fra ligningen?

c) Hva skjer med resultatet for høyden hvis du endrer lengden på tidssteget ditt? (Det kan hende du vil endre på Rate-parameteren for å tilpasse animasjonsfarten). Kjør koden med

- *dt = 0.1*
- *dt = 0.001*
- *Hvordan er disse resultatene sammenlignet med det teoretiske?*

d) Prøv å endre verdien for g. *På månen er f.eks.* g = -1.625 m/s^2. *Hva skjer med formen på posisjonsgrafen? Hva skjer hvis du bruker lavere eller høyere verdier for startfarten? Oppfører programmet seg slik du forventer?*

Bevegelse med luftmotstand

Dersom programmet ditt kjører, og du får resultater som ligner på de teoretiske, er vi nå klare for å legge til luftmotstanden. Luftmotstand er enkelt fortalt et resultat av at objekter som beveger seg gjennom luft kræsjer med molekylene i luften. Luftmotstanden gjør at ting som beveger seg, bremses opp. En vanlig modell for luftmotstanden på et objekt, er

$$L = -k \cdot v \cdot |v|$$

Hvor L er kraften som virker på objektet (altså luftmotstanden), og k er en konstant som avhenger av objektets form og lufttettheten. v er farten til objektet, og $|v|$ er *absoluttverdien* til farten. Absoluttverdien til et tall er det samme som tallet uten fortegn. Så $|-2| = 2$.

e) Se nøye på formelen for luftmotstand. Hvilken vei virker luftmotstanden på et objekt som beveger seg rett opp? Rett ned? Horisontalt?

Nå skal vi legge til luftmotstanden i koden vår. Det gjør vi ved å legge til et ekstra ledd i utregning av akselerasjonen i while-løkken. Målet vårt er å finne en sannsynlig verdi for konstanten k, luftmotstandens størrelse. Denne er ikke så lett å regne ut, så vi skal bruke en "prøv og feil"-metode. Når vi har funnet en verdi for k, kan vi også modellere ballens bevegelse i helt andre tilfeller.

```
28  #Initial Conditions
29  Ball = sphere(radius=0.16,pos=vec(0,0,0),color=color.red)
30  v = 4.2 # m/s
31  g = -9.81 #m/(s*s)
32  m = 0.018 #kg
33  k = 0.02 # kg/m
34
35  t = 0 #s
36  dt = 0.01 #s
37
38  #While Loop
39
40  while Ball.pos.y >= 0:
41    rate(700) #animation speed
42    y.plot(t,Ball.pos.y) #plot y-position of ball
43
44    a =
45    v =
46    Ball.pos.y =
47    t =
48
49    arrow.pos = Ball.pos ; Darrow.axis = 0.1*vec(0,(a-g),0) ; Darrow.visible=True
```

- Velg en verdi for *k*, for eksempel $k = 0.02$. Denne må skrives inn under "`#Initial Conditions`" siden den er en konstant som ikke skal endres.
- Hvordan skal formelen for luftmotstand skrives ned for at datamaskinen skal forstå hva som skjer? Legg til formelen i uttrykket for akselerasjonen. Husk at for å få akselerasjon må du dele på massen *m*. *Hint: I koden kan vi skrive abs(v) for å få absoluttverdi*

- For å sjekke at luftmotstanden peker i riktig retning, kan du skrive inn

  ```
  Darrow.pos = Ball.pos ; Darrow.axis = 0.1*vec(0,(a-g),0) ;
  Darrow.visible=True
  ```

  nederst i while-løkken. Da tegner programmet en pil for deg som viser retningen på luftmotstanden. Du kan variere tallet foran `vec` for å justere størrelsen på pila.

Analyse av resultatene

f) Sett dt = 0.001. Kjør programmet med forskjellige verdier for k*, både veldig små (ca. 0.0001), veldig store (ca. 1), og noen midt imellom. Hva skjer med animasjonen? Hva skjer med posisjonsgrafen til kula? Hvordan forklarer du det du ser?*

g) Klarer du å finne en verdi for k *som gjør at den maksimale høyden til kula blir den samme som du målte i forsøket om mekanisk energi?*

h) Nå som du vet hvilken verdi k *har, kan vi bruke programmet vårt til å se på bevegelsen i et annet tilfelle. Hvor høyt kommer f.eks. ballen dersom du skyter den opp med en større startfart? Hvordan er det sammenlignet med tilfellet hvor mekanisk energi er bevart?*

Se eventuelt ekstraoppgave I for et annet problem hvor vi bruker k.

Ekstraoppgaver (valgfri rekkefølge)

Nå som programmet ditt virker, er det mange ting vi kan gjøre for å utvide og utforske modellen. Under følger tre forslag. Velg selv hvilken rekkefølge du vil gjøre dem i. Kanskje kommer du på noe annet som er gøy? Her er det kun kreativiteten som setter grenser. Husk å skrive hva du gjør, og hva som skjer, i svararket.

I) Terminalfart for ballen

Nå skal vi bruke luftmotstandskoeffisienten *k* som vi fant, til å finne ballens terminalfart.

Åpne trinketen på denne adressen: https://trinket.io/glowscript/b9686603fe . Husk å lagre, dele, og legge ved en link i den røde ruten under. Du kan se at denne koden ligner på den du skrev tidligere, og at luftmotstandskoeffisienten er satt til $k = 0$. Det som er annerledes her, er at ballen begynner i en høyde *h_start* over bakken, med startfart null. Du kan selv skrive inn hvor høyt du skal slippe ballen, ved å endre verdien `h_start` helt øverst i koden.

I a) Kjør programmet med k *= 0 og forskjellige verdier for h_start. Hvordan ser den blå fartsgrafen ut? Hvor fort faller ballen rett før den treffer bakken?*

I b) Sett k *til å ha den verdien du fant tidligere, og kjør programmet på nytt. Hva skjer med fartsgrafen nå? Hva med posisjonsgrafen?*

I c) Hvor fort faller ballen rett før den treffer bakken? Hvor høyt må ballen begynne for at den skal rekke å komme til terminalfart?

II) Bedre modell for luftmotstand

Modellen vi har brukt for luftmotstand er god, men ikke helt nøyaktig. Egentlig er det sånn at når farten er liten, er luftmotstanden gitt ved: $L = -k \cdot v$. For å modellere dette, skal vi bruke flere while-løkker etter hverandre, som beskriver ulike deler av bevegelsen. Begynn med å åpne trinketen du brukte for å modellere ballen vi kastet oppover.

- Vi vet at startfarten er $v = v_0$, og at farten deretter avtar - derfor kan vi bruke farten som kriterium i while-løkkene våre. Så lenge farten er større enn $v = 1$ bruker vi at luftmotstanden er $L = -k \cdot v \cdot |v|$. Etter det, og så lenge farten er mindre enn $v = 1$, bruker vi $L = -k \cdot v$. Så bruker vi $L = -k \cdot v \cdot |v|$ frem til ballen treffer bakken.
- Siden farten er stor til å begynne med, kan den første while-løkken vår se ut akkurat som den vi har nå. Men vi må endre kriteriet i den første linjen til: `while v > 1:` . Det forteller oss at løkken kjører helt til farten er $v = 1$.
- Etter det er farten liten frem til den har snudd og blitt større. Derfor lager vi en ny while-løkke. Denne løkken skal vare til farten er -1. Hvordan kan du sørge for at det skjer? Resten av innholdet i løkken kan være nesten likt den løkken du allerede har. Husk at du nå må ha et annet uttrykk for luftmotstanden! Pass på at luftmotstanden fortsatt har riktig retning.
- Når farten har blitt -1, går luftmotstanden igjen som $L = -k \cdot v^2$. Lag en tredje while-løkke som tar hånd om dette, frem til ballen lander.

Analyse:

II a) Hva skjer med bevegelsen til ballen? Prøv ulike verdier av k, og forskjellige tall som kriterier i while-løkkene (i stedet for v > 1 og v > -1 kan du for eksempel prøve v > 3 og v > -3). Hva ser du? Hvorfor ser du det du ser, tror du?

III) Modell for kanonen

Sånn som modellen er nå, bestemmer vi bare startfarten til ballen i det den forlater kanonen. Men med en ekstra while-løkke kan vi enkelt modellere en kanon som skyter ballen ut for oss.

- Det første vi må gjøre er å endre startposisjonen til ballen. Det gjør du i linjen hvor du definerer ballen under `#Initial Conditions`. Endre `pos=vec(0,0,0)` til f.eks. `pos=vec(0,-1,0)`. Da starter ballen 1 meter under nullnivået vårt. Vi må også endre startfarten til ballen, slik at den starter i ro inne i kanonen.
- Deretter må vi lage en while-løkke som beskriver bevegelsen til ballen mens den er inne i kanonen. Denne løkken må vi sette inn over den andre, slik at programmet leser den først.
- While-løkken for kanonen skal vare så lenge y-posisjonen til ballen er mindre enn 0, så vi skriver `while Ball.pos.y <= 0:`
- Inne i løkken skriver du omtrent det samme som vi hadde i den andre. Merk at her er ikke akselerasjonen *g*, den er en annen, positiv verdi. Velg f.eks. $a = 4$.
- Når du nå trykker play, bør ballen skytes ut, før den beveger seg som før.

Analyse:

III a) Hvilke endringer ser du i posisjonsgrafen din?

III b) Prøv deg frem med forskjellige verdier for a. Hvor stor må akselerasjonen være for at ballen skal komme like høyt opp som før?

Ekstra utfordring: Modell av fjæra i kanonen

- I Fysikk 2 skal du lære at kraften som virker på ballen fra fjæra i kanonen, ikke er like stor hele tiden. En modell for kraften fra fjæra på ballen er gitt ved

  $$F = K \cdot x$$

  hvor *K* er en konstant som beskriver stivheten til fjæra, og *x* er hvor langt fjæra er strukket (eller sammentrykket) fra likevektsposisjonen (se figur). Vi antar at fjæra er plassert sånn at vi dytter den sammen når ballen er lavere enn nullnivået i animasjonen vår (det vil si at Ball.pos.y < 0). Da kan vi bruke `Ball.pos.y` i stedet for *x* når vi skriver i programmet vårt.

- Endre akselerasjonen i den første while-løkken (for kanonen), slik at akselerasjonen er gitt ved *K* og *x*. Husk å dele kraften på massen for å få akselerasjon. Pass også på fortegn - akselerasjonen skal ha positiv retning, men hvilket fortegn har `Ball.pos.y` inne i kanonen?

Analyse:

- *III c) Kjør programmet for ulike verdier av K, og se hva som skjer.*

- *III d) Sammenlign posisjonsgrafen med den du fikk for konstant akselerasjon - hvordan er den annerledes?*

IV) Bevegelse i flere retninger

Frem til nå har vi bare sett på bevegelse i én dimensjon - men når vi programmerer er det ikke så vanskelig å legge til en dimensjon til.

- For å bestemme ballens posisjon i y-retningen, har vi brukt `Ball.pos.y`. På samme måte kan vi bestemme ballens posisjon i x-retningen med `Ball.pos.x`.
- Prøv først å flytte ballen til siden, ved å legge til en linje under `#Initial Conditions`: `Ball.pos.x = 0.5` (eller en annen verdi du liker)
- Se nå om du kan få ballen til å bevege seg sidelengs, som om den ble kastet fremover. Se på hvordan vi fikk ballen til å bevege seg i *y*-retning, og gjør det samme for *x*-retningen. I utgangspunktet kan vi anta at det ikke er noen akselerasjon i *x*-retning, men at farten er konstant.

IV a) Kjør programmet for å se om du får den bevegelsen du forventer. Hva ser du?

IV b) Hvis du vil, er det lett å legge til en akselerasjon i x-retning også, for eksempel fra luftmotstand. Hva skjer med bevegelsen til kula nå?

Vedlegg F: Eksempelkode for originalt oppgavesett

Her presenteres koden slik den ser ut når elevene er ferdige med oppgavesettet. Det er lagt inn flere kommentarer som forklarer hva som skjer. Koden går over to sider.

```
GlowScript 2.9 VPython
################################################
#I denne bolken mellom linjene med ##, ligger kode som allerede er
#til stede når elevene åpner linken til Trinket. Koden sørger for at
#animasjonsvinduet har riktig størrelse, og at koordinatsystemet
#ser ut som det skal. Her gjøres det også klart et grafvindu og
#et pil-objekt, slik at elevene enkelt kan tegne piler i animasjonen
#og plotte posisjonsgrafen til kula med få kommandoer.

#Setting up scene
scene.background = color.white
scene.width = 650 ; scene.height = 500 ; scene.align = "left"
scene.autoscale=False
scene.range=2
scene.camera.follow(pos=vec(0,scene.range-1,0))

#Setting up coordinate system
cory = arrow(pos=vec(-0.5,0,0), axis=vec(0,2,0), shaftwidth = 0.05, color=color.black)
corx = arrow(pos=vec(-0.5,0,0), axis=vec(2,0,0), shaftwidth = 0.05, color=color.black)

for i in range(0,2,0.5):
    box(pos=vec(-0.50,i,0),size=vec(0.2,0.03,0.03), color=color.black)

floor = box(size=vec(10,0.01,10),color=vec(0.782, 0.644, 0.4))

#Setting up arrows
L_arrow = arrow(color=color.blue,shaftwidth = 0.05,headwidth = 0.07,visible=False)

#Setting up for plotting
G1 = graph(align="left",xtitle='t',ytitle='y')
y = series(graph=G1,label="Height",color=color.red)

################################################

#All koden under er skrevet av elevene, bortsett fra de to
#overskriftene "Initial Conditions" og "While Loop".

#Initial Conditions

#Denne linjen definerer en kule ved navn "Ball", med posisjon,
#størrelse og farge
Ball = sphere(pos=vec(0,0,0),radius=0.16,color=color.red)

```

```
#Linjene under definerer ballens startfart v, tyngdeakselerasjonen g,
#ballens masse m, starttiden t, og luftmotstandskoeffisienten k.
#Tidssteget dt sier hvor lang tid som går mellom hver beregning av
#ballens nye fart og posisjon.

v = 4.2 # m/s
g = -9.81 #m/(s*s)
m = 0.018 #kg
t = 0 #s
dt = 0.001 #s
k = 0.02 # kg/m

#While Loop

#Det er her ballens bevegelse regnes ut. Så lenge ballen er over
#bakken (y >= 0), kjøres linjene under igjen og igjen. For hver
#runde beveger ballen seg et lite stykke. Animasjonsvinduet
#oppdateres automatisk for hvert tidssteg.

while Ball.pos.y >= 0:
  rate(1000) #animasjonshastighet, antall tidssteg pr. sekund
  y.plot(t,Ball.pos.y) #plot ballens y-posisjon som funksjon av tiden

  a = g -k*(v*abs(v))/m #regn ut akselerasjonen med Newtons andre lov
  v = v + a*dt #regn ut hastighet for hvert tidssteg
  Ball.pos.y += v*dt #regn ut ny posisjon etter hvert tidssteg
  t += dt #regn ut tiden etter hvert tidssteg

  #Denne linjen er også en del av løkken, og tegner en pil for
  #luftmotstanden i animasjonsvinduet. Elevene får utdelt kodelinjen,
  #og kopierer den inn i koden sin.
  L_arrow.pos = Ball.pos ; L_arrow.axis = 0.1*vec(0,(a-g),0) ; L_arrow.visible=True
```

Vedlegg G: Revidert oppgavesett

Oppgaveark: Vertikalt kast med programmering i VPython

Læringsmål: Når eleven er ferdig med denne økten, skal han/hun

- Forstå hvordan Newtons andre lov kan brukes til å finne et uttrykk for akselerasjon.
- Forså hvordan luftmotstand påvirkes av *v* og *k*, og hvilken effekt dette har for bevegelsen til et objekt.
- Forstå de grunnleggende prinsippene bak enkel programmering
    - Variabler, løkker, oppdatering av variabler, tidssteg, print til terminal
- Kunne bruke programmering til å undersøke bevegelse med og uten konstant akselerasjon ved hjelp av VPython.
- Ha noe innsikt i hvordan fysikere jobber med å modellere fysiske fenomener.

Forutsetninger: Før eleven begynner på denne oppgaven, bør han/hun

- Ha jobbet med Newtons andre lov
- Ha gjort seg noe erfaring med grunnleggende programmering
    - Definere variabler, regne med disse
    - Hvordan while-løkker fungerer
    - Oppdatering av variabler, x = x + 1
- Ha sett og hørt om hvordan en while-løkke fungerer, og brukt en slik løkke til enkle formål, f.eks. printe partall
- Ha fått undervisning i hvordan man regner på bevegelse med v = v + a*dt, s = s + v*dt

Først og fremst

Dere skal jobbe to og to. Begge elevene på hver gruppe bør ha sin egen datamaskin. Mens dere jobber med oppgaven skal én av dere jobbe med koden på sin datamaskin, mens den andre skriver svar her i oppgavearket på sin datamaskin. Underveis bytter dere plass, sånn at begge to får programmert, og begge får skrevet.

Dere skal skrive inn svar direkte i dette oppgavearket. Dere skal svare på oppgavene som er nummerert med a), b), c) også videre, i de røde rammene. I tillegg skal dere skrive inn link til koden deres øverst i oppgavearket, sånn at læreren kan se den.

Hensikt

Tidligere har vi gjort et forsøk hvor vi skjøt en kule opp i luften, og undersøkte hvor høyt den kom. Vi undersøkte om den mekaniske energien var bevart, og så at den ikke var fullstendig bevart, fordi luftmotstanden bremset kulens bevegelse. I denne oppgaven skal vi bruke programmering til å modellere en kule som blir skutt opp i luften.

Hensikten med oppgaven er å finne en sannsynlig verdi for luftmotstandskoeffisienten *k*. Denne koeffisienten forteller oss hvor stor luftmotstanden for et objekt er, og avhenger av objektets form, størrelse og overflate.

Fremgangsmåte

- Begynn med å følge denne linken:
  https://trinket.io/glowscript/f73ff38312
- Du må lagre koden sånn at du ikke mister endringene dine.
  - Trykk på lagreknappen over koden (den heter "remix"). Fyll inn e-postadresse, finn på et passord, og trykk "Create Account".
  - Nå skal det dukke opp en grønn knapp som heter "Remix". Trykk på den
  - Nå har du kopiert koden til din bruker. Du kan endre som du vil, og så lagre endringene ved å trykke på lagreknappen.

Trykk på "dele"-knappen over vinduet, og vel "Link" i menyen som kommer opp. Kopier linken til koden din her:

Til venstre for Trinket-logoen over koden, er det en knapp med tre streker. Trykk på den, og deretter på "Fullscreen" – da blir både koden og resultatene lettere å lese

- Ignorer den øverste delen av koden, som er rammet inn med ####. Her defineres blant annet koordinatsystemet og andre ting som er kronglete og tar tid å sette opp.

Bevegelse uten luftmotstand

Vi skal bruke en while-løkke til å undersøke bevegelsen til en kule som kastes oppover i luften. Først gjør vi beregninger uten luftmotstand, for å sjekke at programmet virker. Hvis du vet hvordan du gjør dette og/eller vil prøve på egen hånd, kan du det. Hvis du er mer usikker på hva du skal gjøre, kan du følge instruksjonene under. Hvis du står fast, kan du se i dokumentet «nyttige kommandoer og vanlige feil», og se om noe der kan være til hjelp.

- **Forberedelser:** Når vi skal modellere et fysisk system, må vi ta utgangspunkt i den fysikken vi kan. Ta frem penn og papir. Tenk på forsøket du gjorde med fjærkanonen. Skriv ned (eller tegn) for deg selv hvilke krefter som virker på kulen som er i lufta, hvilke variabler du trenger, og hvilke størrelser som er kjent. Hvilke av størrelsene endrer seg over tid, og hvilke er alltid de samme? Hvordan forventer du at kulen skal bevege seg når den skytes opp i lufta, sånn ca.? Når du har en oversikt over systemet, kan du begynne å skrive koden din.

- **Konstante størrelser:** Her skriver du ned de størrelsene som ikke forandrer seg underveis. Både kulens masse, og tyngdeakselerasjonen er slike størrelser. Her kan du også lage en kule. Kall den hva du vil, og plasser den i posisjonen (0,0,0). Gi den en farge du liker, og radius 0.16 m.

- **Initialbetingelser:** Dette er kode som beskriver startsituasjonen. Hvilken startfart målte du i forsøket med fjærkanonen? Vi setter starttiden til null. Vi må også bestemme hvor stort tidssteg vi skal ha. 0.01s er et bra utgangspunkt.

```
27 #Konstante størrelser
28 g = -9.81 #m/s**2
29 m =
30 #ta vekk denne linjen og lag en ball
31
32 #Initialbetingelser (startsituasjon)
33 v =
34
35 t = 0 #s
36 dt = 0.01 #s
```

- **While-løkke:** I denne løkken beregnes kulens bevegelse for hvert tidssteg, frem til kriteriet for løkken ikke lenger tilfredsstilles. **NB! Husk at alt som skrives inne i løkken må ha ett innrykk (tab-knappen på tastaturet)**

- Start while-løkken ved å skrive: `while Ball.pos.y >= 0:` . Dette forteller oss at løkken repeteres så lenge kulen er over $y = 0$. Hvis du har kalt kulen din noe annet enn «Ball», skriver du det i stedet for, f.eks. `Alfred.pos.y`.

- På neste linje skriver du `rate(100)` - dette bestemmer hastigheten på animasjonen din.

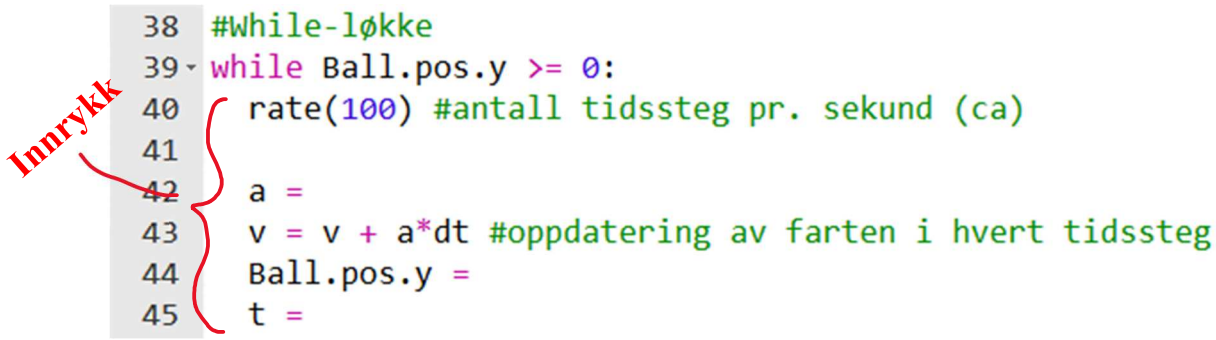
- For hvert tidssteg må vi vite hva akselerasjonen skal være. Bruk Newtons andre lov til å finne ut hva akselerasjonen til kulen skal være. Skriv gjerne på papir hvis du synes det er enklere. Husk at positiv retning er oppover.

- Så oppdaterer vi farten, og deretter posisjonen.
  - Vi antar at akselerasjonen er konstant på hele tidsintervallet. Det vil si at formelen for *v* etter et tidssteg *dt* med akselerasjon *a* er:

    $$v = v_0 + a \cdot \mathrm{dt}$$

    Bruk denne formelen for å oppdatere farten inne i løkken. Husk at du kan skrive f.eks. *v = v + 1*, hvor *v* etter likhetstegnet er den gamle verdien, og *v* før likhetstegnet er den nye verdien. Du trenger altså ikke å skrive v_0 i koden.

  - Vi antar at kulen har farten *v* på hele intervallet *dt*. Hva er formelen for y-posisjonen etter et tidssteg *dt* med konstant hastighet *v*? Skriv inn denne. Husk at Y-posisjonen til kulen er gitt ved `Ball.pos.y` (eller `Alfred.pos.y`).
    Hint: Formelen ligner veldig på formelen for v.



- Til slutt i løkken må vi huske å oppdatere tiden. Det gjør du ved å legge til et tidssteg *dt* til *t*:

  $$t = t + dt$$

  Dette er ikke avgjørende for at programmet skal skjønne hva som skjer, men blir viktig senere når vi skal plotte.

a) *Hva burde skje når du trykker play nå? Hvilken bevegelse er det vi prøver å modellere? Hva forventer du at skal skje?*

Kjør programmet. Oppfører animasjonen seg sånn som du forventer? Hvis ikke, kan du sjekke følgende ting i koden:

- *Har du rette verdier for ballens masse, og tyngdeakselerasjonen?*
- *Har du rette initialbetingelser? Disse skal være de samme som i forsøket du gjorde.*
- *Har du skrevet riktige kommandoer i while-løkken? Det kan være lurt å sammenligne med eksemplene i hjelpe-heftet.*
- *Pass opp for småfeil – sjekk at variabelnavn er skrevet likt overalt (også små/store bokstaver), at du bruker komma og punktum riktig, og at du har innrykk i while-løkken.*

Analyse av resultatene

Vi ønsker å få greie på nøyaktig hvor høyt opp kulen kommer. For å gjøre det kan du legge inn denne linjen øverst i løkken din (men under `rate`-linjen): `y.plot(t,Ball.pos.y)` Denne linjen plotter kulens y-posisjon som funksjon av tiden. Husk å skrive ditt eget navn for kulen, `y.plot(t,Alfred.pos.y)`.

```
38 #While-løkke
39 while Ball.pos.y >= 0:
40   rate(100) #antall tidssteg pr. sekund (ca)
41
42   a =
43   v =
44   Ball.pos.y =
45   t =
```

b) Nå kan du lese av ballens høyde ved å plassere musepekeren over grafen. Hvor høyt over utgangspunktet kommer den? Hvor lang tid bruker den på å komme opp og ned?

c) For å teste at programmet virker og faktisk beskriver virkeligheten, kan vi sammenligne med en teoretisk løsning. Bruk ligningen for bevaring av mekanisk energi til å regne ut hvor høyt kulen kommer. Husk å bruke den samme startfarten som du bruker i programmet. Hvordan stemmer dette resultatet med resultatet fra simuleringen din? Dersom de to resultatene er veldig forskjellige, bør du se om du har feil i koden.

d) Hva skjer med resultatet for høyden hvis du endrer lengden på tidssteget ditt? Kjør koden med

- *dt = 0.1*
- *dt = 0.001 (Det kan være lurt å kjøre med en høyere rate(), f.eks. 1000)*
- *dt = 0.0001 (Kjør med en mye høyere rate(), f.eks. 10 000)*
- *Hvordan er disse resultatene sammenlignet med det teoretiske du fant i oppgave c)? Hvordan tolker du effekten av* dt*?*

e) Hva tror du skjer dersom tyngdekraften er lavere? På månen er f.eks. g = -1.625 *m/s*^{2}. *Hva skjer med formen på posisjonsgrafen? Endrer den seg? Hvor høyt går kulen nå?*

Hva skjer hvis du bruker lavere eller høyere verdier for startfarten? Oppfører programmet seg slik du forventer?

Bevegelse med luftmotstand

Dersom programmet ditt kjører, og du får resultater som ligner på de teoretiske, er vi nå klare for å legge til luftmotstanden. Nå har vi ingen teoretiske resultater å sammenligne med, men vi kommer til å bruke forsøket vi gjorde med fjærkanon for å lære noe nytt om luftmotstanden på kulen.

Luftmotstand er enkelt forklart et resultat av at objekter som beveger seg gjennom luft krasjer med molekylene i luften. Luftmotstanden gjør at ting som beveger seg, bremses opp. En vanlig modell for luftmotstanden på et objekt, er

$$L = -k \cdot v \cdot |v| \qquad (1)$$

Hvor L er kraften som virker på objektet (altså luftmotstanden), og k er en konstant som avhenger av objektets form og lufttettheten. v er farten til objektet, og $|v|$ er *absoluttverdien* til farten. Absoluttverdien til et tall er det samme som tallet uten fortegn. Så $|-2| = 2$, og $|2| = 2$.

f) Se nøye på formelen for luftmotstand. Hvilken vei virker luftmotstanden på et objekt som beveger seg rett opp? Rett ned? Horisontalt? Klarer du å se hvorfor vi må bruke absoluttverditegnet?

Nå skal vi legge til luftmotstanden i koden vår. Det gjør vi ved å bruke Newtons andre lov til å finne et annet uttrykk for akselerasjonen enn det vi har. Vi trenger nesten ikke endre noen andre deler av koden.

Målet vårt er å finne en sannsynlig verdi for konstanten k, luftmotstandens størrelse på kulen vår. Denne er ikke så lett å regne ut, så vi skal bruke en "prøv og feil"-metode, ved hjelp av forsøket vi gjorde tidligere. Når vi har funnet en verdi for k, kan vi også modellere kulens bevegelse i helt andre tilfeller, som vi ikke har mulighet til å teste i klasserommet.

- Velg en verdi for *k*, for eksempel $k = 0.02$. Dette er en størrelse som ikke endrer seg med tiden. Sett den inn på et passende sted i koden.

- Hvilke krefter virker på kulen når den er i lufta? Bruk Newtons andre lov og uttrykket for luftmotstanden til å finne et uttrykk for akselerasjonen. Bruk gjerne penn og papir, hvis det er enklere. Skriv uttrykket for akselerasjonen inn i while-løkken.
  Hint: I koden kan vi skrive abs(v) for å få absoluttverdi

- For å sjekke at luftmotstanden peker i riktig retning, kan du skrive inn

  ```
  L_arrow.pos = Ball.pos ; L_arrow.axis = 0.1*vec(0,a-g,0) ; L_arrow.visible=True
  ```

  nederst i while-løkken. Da tegner programmet en pil for deg som viser retningen på luftmotstanden. Den første delen gir pila samme posisjon som kulen. Den andre delen bestemmer retningen og størrelsen på pila. Du kan variere tallet foran `vec` for å justere størrelsen. Den siste delen bestemmer bare at pila skal vises. Husk å endre Ball til ditt egent navn for kulen.

```
27 #Konstante størrelser
28 g = -9.81 #m/s**2
29 m =
30 #ta vekk denne linjen og lag en ball
31
32 #Initialbetingelser (startsituasjon)
33 v =
34
35 t = 0 #s
36 dt = 0.01 #s
37
38 #While-løkke
39 while Ball.pos.y >= 0:
40   rate(100) #antall tidssteg pr. sekund (ca)
41   y.plot(t,Ball.pos.y)
42
43   a =
44   v = v + a*dt
45   Ball.pos.y =
46   t =
47
48   L_arrow.pos = Ball.pos ; L_arrow.axis = 0.1*vec(0,a-g,0) ; L_arrow.visible = True
```

g) Hvordan tror du posisjonsgrafen til kulen endrer seg når du legger til luftmotstanden? Kommer kulen høyere eller lavere? Endres formen på grafen? Prøv å lage noen hypoteser for deg selv før du går videre. (Men du trenger ikke skrive noe)

Analyse av resultatene
(NB! Husk å endre verdiene i koden tilbake sånn at de reflekterer forsøket med fjærkanonen på jorda)

h) Sett dt = 0.001. Kjør programmet med forskjellige verdier for k*, både veldig små (ca. 0.0001), veldig store (ca. 1), og noen midt imellom. Hva skjer med animasjonen? Hva skjer med posisjonsgrafen til kula? Endrer den form? Hvor høyt kommer kula? Hvordan forklarer du det du ser?*
*(Hint: tallet «3e-2» er datamaskinen sin måte å skrive standardform på, og betyr «3 * 10 $^{-2}$», eller 0,03)*

i) Skriv kort (én setning) hvilken effekt k *har på luftmotstandsuttrykket.*

Skriv kort hvilken effekt k *har på kulens høyde.*

j) Klarer du å finne en verdi for k *som gjør at den maksimale høyden til kula blir den samme som du målte i forsøket med fjærkanonen som ble gjort i klasserommet? Husk at for å teste dette må du ha samme startfart og tyngdeakselerasjon som ble brukt i forsøket.*

k) Nå som du vet hvilken verdi k *har, kan vi bruke programmet vårt til å se på bevegelsen i et annet tilfelle. Hvor høyt kommer kulen dersom du skyter den opp med en høy startfart (*v *større enn 50)? Hvordan forklarer du at posisjonsgrafen blir tilnærmet lineær på vei ned? Hva sier det deg om summen av kreftene på kulen?*

Kjør programmet med den samme høye startfarten, men med k = 0 *(altså uten luftmotstand). Hva forteller dette deg om luftmotstandens effekt på kulen?*

Hva skjer om du bruker en tyngre kule? Prøv deg frem med ulike starthastigheter og masser, og oppsummer kort det du ser.

Se eventuelt ekstraoppgave I for et annet problem hvor vi bruker k.

Vedlegg H: Oppslagsverk for programmering i fysikk

Programmering i fysikk med Glowscript: Et oppslagsverk

Dette er en oversikt over nyttige kommandoer og funksjoner vi bruker når vi programmerer Glowscript. Kommandoene er særlig nyttige for å beskrive bevegelse med Newtons andre lov. Her finner du eksempler på:

Hvordan definere variabler
Hvordan lage objekter i animasjonsvinduet
Hvordan sette opp while-løkker
Hvordan oppdatere variabler
Hvordan vi kan beskrive bevegelse
Vanlige feil som er lurt å sjekke

Variabler

Du kan kalle en variabel hva du vil, og tilordne den en verdi. Dette gjør vi med likhetstegnet:

```
Min_Variabel = 5              # Min_Variabel får verdien 5
A = 4                         # A får verdien 4
Bil_3 = 63                    # Bil_3 får verdien 63
Min_sum = Min_Variabel + A    # Min_sum får verdien 9
```

MERK: Variabelnavn må begynne med en bokstav, og de kan ikke inneholde mellomrom. Hvis vi vil bruke to ord i variabelnavnet, skiller vi med understrek.

Objekter

Glowscript har mange forskjellige *objekter*. Et objekt er en ting som vises i animasjonsvinduet, og har mange innebygde funksjoner. Her er en beskrivelse av de vanligste objektene:

```
Min_kule = sphere(pos=vec(x,y,z), radius = r, color=color.red)
```

Sphere-kommandoen lager en kule. Du kan selv velge posisjonen til kula ved å sette inn tall for x, y og z. På samme måte kan du velge hvor stor kula skal være ved å sette inn et tall for r. Du kan også gi den en farge ved å bytte `color.red` med `color.blue` eller en annen farge (men ikke alle farger fungerer).

```
Min_boks = box(pos=vec(x,y,z), size=vec(bredde, høyde, lengde),
color = color.red)
```

Box-kommandoen lager en tredimensjonal boks. Du kan selv velge posisjonen til boksens sentrum ved å sette inn tall for x, y og z. På samme måte kan du velge hvor lange sidene av boksen skal være ved å sette inn tall for bredde, høyde og lengde. Du kan også gi den en annen farge ved å bytte `color.red` med `color.blue` eller en annen farge (men ikke alle farger fungerer).

While-løkker

En *while-løkke* kjører så lenge en gitt påstand er sant. Her er et eksempel:

```
# Det er viktig å definere variabelen vi skal bruke i
kriteriet for løkken, og variabler vi vil oppdatere inne i
løkken
A = 0
Min_Sum = 0

# Så starter vi løkken. Husk at alt inne i løkken må skrives
med innrykk
while A < 5:
     Min_Sum = Min_Sum + A
     print(Min_Sum)
     A = A+1
```

Denne løkken kjører fem ganger. Når A = 0, 1, 2, 3, og 4. For hver gang oppdateres variabelen `Min_Sum`, før programmet skriver den ut i resultatvinduet. Når vi kjører koden får vi opp følgende:

```
0
1
3
6
10
```

Oppdatering av variabler

Ganske ofte bruker vi løkker til å oppdatere variabler. Noen ganger ønsker vi å legge noe til en verdi vi allerede har, sånn som når vi oppdaterte `Min_Sum` i det forrige eksempelet. Når vi skriver

```
A = 5
A = A + 1
A = A + 1
```

Betyr det at A går fra å være 5 til å bli 5+1 = 6, og så til 6+1=7. Vi kan fargekode eksempelet for å gjøre det tydeligere:

```
A = 5
A = A + 1
A = A + 1
```

Når vi bruker den samme bokstaven på begge sider av likhetstegnet, betyr det altså at vi tar den *forrige* verdien for A, gjør noe med den, og lagrer resultatet som den *nye* verdien for å. Den *forrige* verdien er alltid på høyre side, mens den *nye* verdien er på venstre side.

Å beskrive bevegelse

Vi kan bruke løkker og oppdatering av variabler for å beskrive bevegelsen til et objekt. Her er et eksempel:

```
# Først setter vi opp startbetingelsene våre. Vi lager en boks, og setter startfarten til 10. Starttiden er null, og tidssteget er 0.01

Min_Boks = box(pos=vec(0,0,0), size = vec(1,1,1), color = color.blue)

v = 10 # m/s
t = 0 # s
dt = 0.01 # s

# Så setter vi opp bevegelsen i en while-løkke. Vi vil at boksen skal bevege seg i x-retning til den stopper, altså så lenge v > 0. Vi gir ballen negativ akselerasjon -1, og så regner vi ut farten og posisjonen for hvert tidssteg med bevegelseslignigene. Til slutt oppdaterer vi tiden

while v > 0:
     rate(100)
     a = -1 # m/s**2
     v = v + a*dt # m/s
     Min_Boks.pos.x = Min_Boks.pos.x + v*dt # m
     t = t + dt # s
```

Vanlige feil

Det er mange småfeil som er lette å gjøre når man programmerer, også for de profesjonelle. Her er en liste med ting som er lurt å sjekke når programmet ikke virker som du forventer:

Store og små bokstaver: Husk at programmet tar hensyn til dette. Hvis du har brukt stor bokstav i et variabelnavn, må du bruke stor bokstav alle gangene du skriver denne variabelen.

Komma og punktum: Husk at du må bruke punktum i stedet for komma i desimaltall. Kommategnet bruker du for eksempel mellom koordinatene når du lager en ball eller boks.

Variabler som ikke er definerte: Hvis du bruker en variabel i en utregning, må datamaskinen ha blitt fortalt hvilken verdi denne variabelen har. Pass på at du ikke bruker variabler i regnestykker som du ikke har definert lenger opp i koden.

Innrykk: Det er fort gjort å glemme innrykk i en while-løkke. Pass på at alt som skal skje inne i løkken er skrevet med innrykk!

Potenser: Husk at potenser i Glowscript skrives **, og ikke ^.

Løkker som kjører for alltid, eller ikke i det hele tatt: Sjekk at variabelen du bruker i kriteriet for while-løkker er definert lenger opp i koden, sånn at datamaskinen finner den. Sjekk at kriteriet er oppfylt, og at det skjer noe med variabelen inne i løkken, så den ikke kjører for alltid.

Sjekk enhetene på grafen din: Noen ganger kan det hende resultatene våre ser riktige ut ved første øyekast. Det er alltid lurt å sjekke enhetene på grafene våre – da kan vi for eksempel se om bevegelsen tar usannsynlig kort eller lang tid.

Koden ser riktig ut, men jeg får rare resultater: Dersom koden fungerer, men resultatene ikke gir mening, kan det hende at fysikken du har skrevet inn er feil. Sjekk at initialbetingelsene dine er riktige, og at de hører til det systemet du skal simulere. Sjekk hvilket uttrykk du har for akselerasjonen, og at du har skrevet bevegelsesligningene riktig. Du kan også prøve å kjøre med et mindre tidssteg, og se om det hjelper. Det kan være nyttig å printe enkelte verdier (f.eks. posisjon eller fart) for å se hva som skjer underveis i løkken.

Husk å tolke resultatene dine: Noen ganger kan vi få rare resultater som vi ikke forventet, men som likevel er riktig. Du bør alltid se om du klarer å gi mening til resultatene dine når du sammenligner med det du opplever til vanlig og de resultatene du forventet. Prøv å finne grunner til at resultatet ble som det ble. Noen ganger har vi bare gjort noe feil, men ikke alltid.