MASTEROPPGAVE - MGLU

Mai 2022

Programmering i naturfag: den algoritmiske tenkeren og utfordringer på veien

Programming in Science: The Computational Thinker and Challenges Along the Way

Gjelder MGLU: Type vitenskapelig

30 stp oppgave

Antall ord: 27 573

Helena Dragland og Cathinka Mønsted Ebert

OsloMet – storbyuniversitetet

Fakultet for lærerutdanning og internasjonale studier

Institutt for grunnskole- og faglærerutdanning

Forord

Denne masteroppgaven er skrevet som avsluttende prosjekt på vår fem år lange lærerutdanning ved OsloMet Storbyuniversitet. Det har vært fem år fylt med spennende undervisning, gode diskusjoner og mye latter. Mange gode vennskap har blitt til, og vi håper de kan vare livet ut.

Inkluderingen av programmering i LK20 har gjort temaet for denne masteren svært aktuelt for oss. Tanken på at vi som lærere i matematikk og naturfag skal undervise i dette temaet har gjort oss en smule engstelige. Derfor har det vært ekstra interessant å fordype oss i dette temaet. Vi har tilegnet oss mye ny kunnskap og en ny, positiv holdning til programmering i naturfag.

Det har likevel vært en krevende prosess, men takket være godt samarbeid og en utrolig dyktig veileder har vi sakte, men sikkert kommet oss gjennom oppgaven. Vi vil derfor takke vår kjære veileder Per Øyvind Sollid som har bidratt med gode råd og støtte underveis. Du har hatt troen på oss, selv i tider der vi har hatt våre tvil selv.

Nå gleder vi oss til å ta fatt på lærerhverdagen, og ikke minst programmeringsundervisningen!

Oslo, mai 2022

Helena Dragland og Cathinka Mønsted Ebert

Sammendrag

Temaet for denne masteroppgaven er programmering i naturfag. Programmering ble implementert i læreplanen høsten 2020, og er dermed nytt for mange lærere og elever. På bakgrunn av et økende behov for teknologiske ferdigheter i arbeidslivet må skolen undervise i programmering slik at elevene er forberedt på dagens og fremtidens samfunn. Den algoritmiske tenkeren er en modell for algoritmisk tenkning som inneholder strategier som kan brukes når elever arbeider med programmering i skolen. Vi vet lite om hvordan algoritmisk tenking ser ut og hvilke utfordringer elever møter på i norske klasserom i naturfag der elevene jobber med programmering. Det er derfor nødvendig å undersøke dette slik at programmeringsundervisning i naturfag kan utvikles.

Hensikten med denne studien er å undersøke hvordan elever på mellomtrinnet arbeider med programmering i naturfagundervisning, og utfordringer som kan oppstå i planlegging og gjennomføring av slike undervisningsopplegg. Studien vil svare på forskningsspørsmålene:

- Hvilke kjennetegn på algoritmisk tenkning kommer til utrykk i elevers arbeid med programmering i naturfag på mellomtrinnet?
- Hvilke utfordringer kommer frem i et undervisningsopplegg med programmering i naturfag på mellomtrinnet?

Dette er en kvalitativ studie der vi har benyttet oss av semistrukturert intervju av to lærere og observasjon av fire klasser som datainnsamlingsmetode. Dataene er analysert gjennom meningskondensering og innholdsanalyse.

Studiens resultater viser at flere av elevene benyttet seg av alle strategiene i den algoritmiske tenkeren. Noen av strategiene ble trukket frem i en introduksjonsøkt og andre strategier ble vektlagt underveis i undervisningen. Ettersom flere av elevene benyttet seg av disse strategiene og så ut til å få utbytte av dem, kan det argumenteres for at det å lære elever strategier fra den algoritmiske tenkeren kan bidra til å veilede dem gjennom programmeringen.

Det ble brukt en ressursperson med kunnskap innen programmering og algoritmisk tenkning i planleggingen av undervisningsopplegget. For at programmeringsundervisningens faglige og didaktiske hensikt skulle komme frem understrekes viktigheten av kommunikasjon mellom planlegger og lærere.

Utfordringer med samarbeid kom til syne der elevene ikke brukte hverandre som ressurs i arbeidet. For at et samarbeid skal fungere er det essensielt at elever vet hvordan de skal samarbeidet. I programmering kan dette gjøres ved å introdusere elever for parprogrammering der begge elevene får utdelt en rolle med eget ansvar. Videre kom utfordringer knyttet til behov for veiledning frem i undervisningen. For at lærere skal veilede elevene i programmeringsundervisning i naturfag kan det være hensiktsmessig å differensiere oppgavene elevene får, samt veilede dem underveis. En av klassene opplevde også teknologiske utfordringer som kom av utladede Spheroballer. For å unngå slike utfordringer er det viktig at lærere har satt seg inn i hvilke teknologiske utfordringer som kan oppstå og hvordan de kan forhindres.

I videre forskning kan en rette fokus på elevers opplevelse av bruken av strategier fra den algoritmiske tenkeren og utfordringer de møter i programmering i naturfag. I tillegg ville det vært interessant å følge elever fra et undervisningsopplegg med programmering i naturfag til et annet for å undersøke om og på hvilke måter elever tar med seg strategiene fra den algoritmiske tenkeren videre, og eventuelt hvorfor de gjør det eller ikke.

Abstract

The theme of this paper is programming in science classrooms. Programming was incorporated to the Norwegian curriculum as of the autumn of 2020, and is therefore still new to teachers and students. Based on an increased need for technological skills in society, schools need to teach programming to prepare the students for both the society of today and the future. The Computational Thinker is a model for computational thinking that presents various strategies for the students to use while working with programming in school. There is little knowledge about how computational thinking appears and what kind of challenges students face in while programming in Norwegian science classrooms. That being the case, it is necessary to investigate this in efforts to improve teaching involving programming in science classrooms.

The purpose of this study is to investigate how students in primary school work with programming during science instruction, and challenges that might occur in the planning and implementation of such instruction. The study will answer the following research questions:

- Which characteristics of computational thinking are expressed in primary school students work in programming in science?
- Which challenges emerges while teaching programming in primary school science classrooms?

This is a qualitative study using transcripts of semi-structured interviews of two teachers and field notes from observations of four teaching sessions. The data was analysed using meaning condensation and content analysis.

The study uncovered that several students utilise all strategies presented in the Computational Thinker. Some of the strategies were employed during the introductory phase, whereas others came in use throughout the session. Considering that many students were applying the strategies, and by observing the positive impact of these, it can be argued that the

Computational Thinker might contribute to facilitate the students' learning process when programming.

A person with experience in computational thinking was used as an advisor during the planning of the instruction. Throughout this process, sound communication between the advisor and the teachers is of key importance to achieve the academic and didactic purpose of the instruction.

Challenges related to cooperation emerged as students stopped utilising each other as resources. For a team to fully function it is essential that students know how to collaborate. When programming, a solution might be to introduce the students to pair programming, wherein both students are assigned a role that each has its own set of responsibilities. Further complications arose from the students' need for guidance as the instruction progressed. For the teachers to properly guide the students in science instruction that involves programming it might be useful to differentiate the tasks that are given to the students whilst continuously providing them with guidance. One of the classes experienced technological issues when some of the Sphero BOLT were discharged. To avoid such issues it is important that teachers are prepared for eventual technological obstacles that might arise, and how to prevent them.

Further research could focus on how students experience the use of strategies from the Computational Thinker, including the challenges they encounter over the course of programming within science. Moreover, an interesting aspect to further examine is how and to what extent students utilise the acquired knowledge from the Computational Thinker in other subjects.

Innholdsfortegnelse

1 Innledning

I dagens samfunn møter vi på teknologi overalt, enten vi er hjemme, på skolen eller på jobb. I hverdagen bruker vi blant annet teknologi til å kommunisere, tilegne oss kunnskap og få med oss nyheter. I tillegg brukes teknologi i stor grad i arbeidslivet og det vil stadig utvikles nye yrker som krever teknologiske ferdigheter i fremtiden ettersom samfunnet utvikler seg. Med den økende bruken av teknologi følger også behovet for mennesker med teknologiske ferdigheter og kunnskaper på arbeidsmarkedet. Yrker som omhandler bærekraft, fornybar energi og grønn omstilling trekkes frem som sikre fremtidsyrker og kompetanse rundt kunstig intelligens og koding etterspørres (UiB, 2022).

Skolens samfunnsmandat pålegger skolen å blant annet utstyre elevene med ferdigheter, kunnskaper og kompetanser som skal forberede elevene til et liv i dagens og fremtidens samfunn. Dette kommer frem i skolens formålsparagraf der det står at opplæringen skal sørge for at elevene blant annet utvikler kunnskap og holdninger for å mestre eget liv, samt delta i samfunnets fellesskap og arbeidsliv (Utdanningsdirektoratet, 2017). Ettersom teknologi regnes som én av flere ferdigheter som er nødvendig for å leve i dagens og fremtidens samfunn er skolen pålagt å undervise elevene i dette. Slike ferdigheter betegnes som *21st Century Skills*, og ved å lære elevene disse ferdighetene bidrar skolen til å forberede elevene på kravene som blir stilt dem senere i livet (Abbot, 2016).

Jeanette Wing satte i 2006 søkelys på computational thinking som en ferdighet som kan åpne for nye problemløsningsstrategier som kan brukes på tvers av fagdisiplinene (Wing, 2006). Hun hevdet at computational thinking er en nødvendig ferdighet for alle mennesker som skal fungere i dagens samfunn, og det bør derfor tilføyes de andre grunnleggende ferdighetene barn lærer i skolen (Wing, 2006). Gjennom å utvikle ferdigheter innenfor computational thinking kan en gå fra å konsumere teknologien til å uttrykke ideer og skape gjennom teknologi (Brennan & Resnick, 2012).

1.1 Problemstilling

En problemstilling argumenterer for behovet for en studie og danner en logisk overgang til studiens hensikt og forskningsspørsmål (Gjevjon, 2019). Vi vil i det følgende gjøre rede for problemstillingen for denne studien.

I LK20 ble teknologi implementert som et kjerneelement i naturfag. Dette kjerneelementet innebærer blant annet at elevene skal forstå teknologiske prinsipper og skape teknologi gjennom programmering (Utdanningsdirektoratet, 2020a). Fagdidaktikk innen programmering i naturfag er imidlertid et lite utviklet fagområde, noe som kan gjøre det krevende for lærere å legge til rette for læring i naturfaglig programmering (Tellefsen, 2021b). I tillegg er programmeringen i naturfag nytt for både lærere og elever, og det kan dermed oppstå utfordringer når programmering anvendes i naturfag. Kunnskap om disse utfordringene er viktig for å kunne tilrettelegge undervisningen på en hensiktsmessig måte.

Fordi programmering kan inneholde problemløsning der løsningsalternativene er mange, er det nyttig for elevene å tilegne seg ferdigheter innen problemløsning (Hazzan et al., 2015). En problemløsningsmetode som Utdanningsdirektoratet (2019) trekker frem knyttet til programmering er algoritmisk tenkning. Algoritmisk tenkning deler elementer, som blant annet problemløsning, med computational thinking (Lee et al., 2011). I en artikkel skrevet av Barr og Stephenson (2011) i forkant av implementeringen av computational thinking i læreplanen, kommer det frem at en vellykket implementering krever en praktisk tilnærming forankret i en operasjonalisert definisjon av begrepet. Videre trekker Barr og Stephenson (2011) frem et sett med spørsmål som må belyses. Noen av disse spørsmålene handlet om hvordan computational thinking kunne se ut i klasserom, hvilke ferdigheter vil elever vise og hva vil det kreves av lærere for å sette computational thinking ut i praksis. Disse spørsmålene er verdt å ta med seg videre ettersom implementeringen av programmering i lærerplanen fremdeles er å regnes som fersk.

Vi vet lite om hvordan algoritmisk tenking ser ut og hvilke utfordringer elever møter på i norske klasserom i naturfag der elevene jobber med programmering. Det er derfor nødvendig å undersøke dette slik at lærere kan kjenne igjen algoritmisk tenkning i egen naturfagundervisning og bli bevisst på hvilke utfordringer som kan oppstå. Dette kan videre bidra til å utvikle undervisning i programmering i naturfag.

1.2 Oppgavens hensikt og forskningsspørsmål

Oppgavens hensikt er å undersøke hvordan elever på mellomtrinnet arbeider med programmering i naturfagundervisning, og utfordringer som kan oppstå i planlegging og gjennomføring av slike undervisningsopplegg. For å undersøke dette har vi formulert følgende forskningsspørsmål:

- Hvilke kjennetegn på algoritmisk tenkning kommer til utrykk i elevers arbeid med programmering i naturfag på mellomtrinnet?
- Hvilke utfordringer kommer frem i et undervisningsopplegg med programmering i naturfag på mellomtrinnet?

2 Teori

I dette kapittelet vil vi presentere teori om algoritmisk tenkning og en utdypende beskrivelse av den algoritmiske tenkeren. Videre skriver vi om programmering der vi vektlegger fysisk programmering og pedagogiske tilnærminger til programmering. Til slutt trekker vi frem tidligere forskning om programmering i naturfag.

2.1 Algoritmisk tenkning/Computational Thinking

Utdanningsdirektoratet (2019) beskriver algoritmisk tenkning som en problemløsningsmetode. Det innebærer å gjøre vurderinger av hvilke steg som må utføres for å kunne løse et problem, og videre kunne utføre stegene for å løse problemet. I tillegg skal elever kunne reflektere rundt hva som er hensiktsmessig at teknologien løser og hva mennesker selv bør løse.

Algoritmisk tenkning blir ofte omtalt synonymt med det engelske begrepet computational thinking (Sanne et al., 2016; Sevik, 2016; Taraldsen & Myhra, 2019; Utdanningsdirektoratet, 2019). Det er ikke utarbeidet en enstemmig definisjon av begrepet computational thinking (Barr et al., 2011; Coulter et al., 2010; Council, 2010; Voogt et al., 2015). I denne oppgaven tar vi utgangspunkt i CSTA Standards Task Force (2016) sin definisjon computational thinking:

> "We believe that computational thinking is a problem-solving methodology that expands the realm of computer science into all disciplines, providing a distinct means of analysing and developing solutions to problems that can be solved computationally. With its focus on abstraction, automation, and analysis, CT is a core element of the broader discipline of computer science" (CSTA Standards Task Force, 2016).

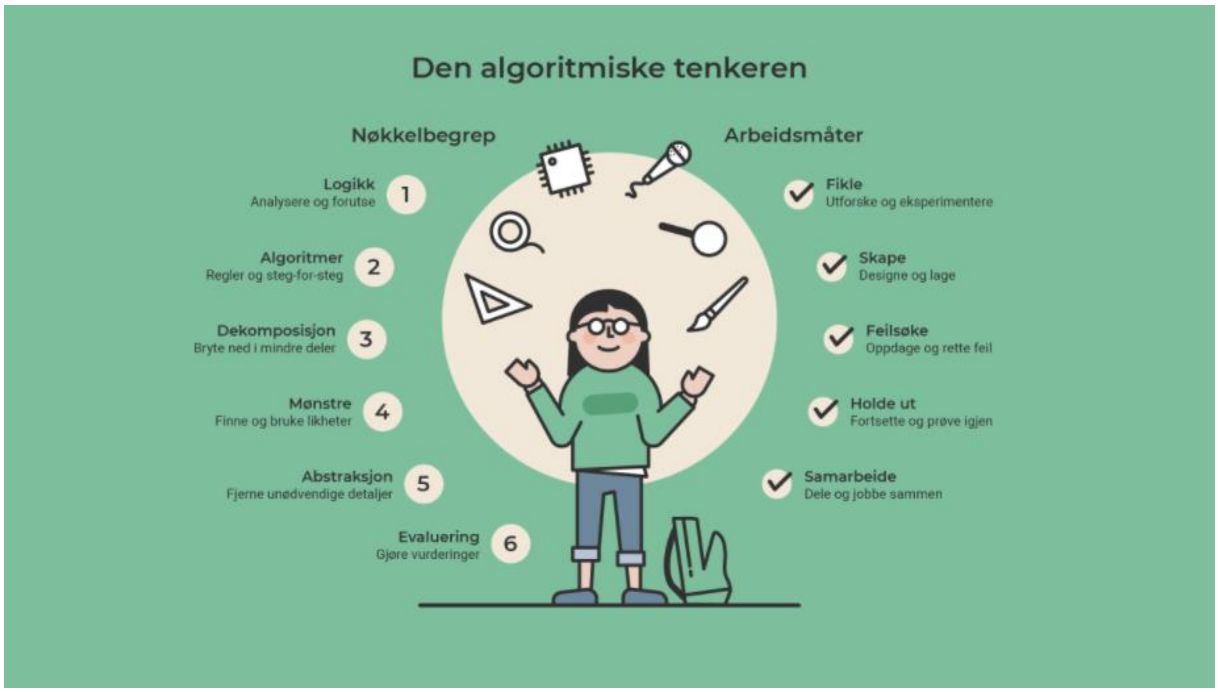
Ettersom vi ser at flere kilder omtaler algoritmisk tenkning og computational thinking synonymt velger vi også å gjøre det, og det har derfor blitt inkludert litteratur om computational thinking i oppgavens innledning, problemstilling og teori. Vi ser imidlertid forskjeller i definisjonene til Utdanningsdirektoratet (2019) og CSTA Standards Task Force

(2016), presentert over, ved at computational thinking innebærer bruken av teknologiske hjelpemidler, mens i algoritmisk tenkning skal det vurderes om det er hensiktsmessig å bruke teknologi eller ikke.

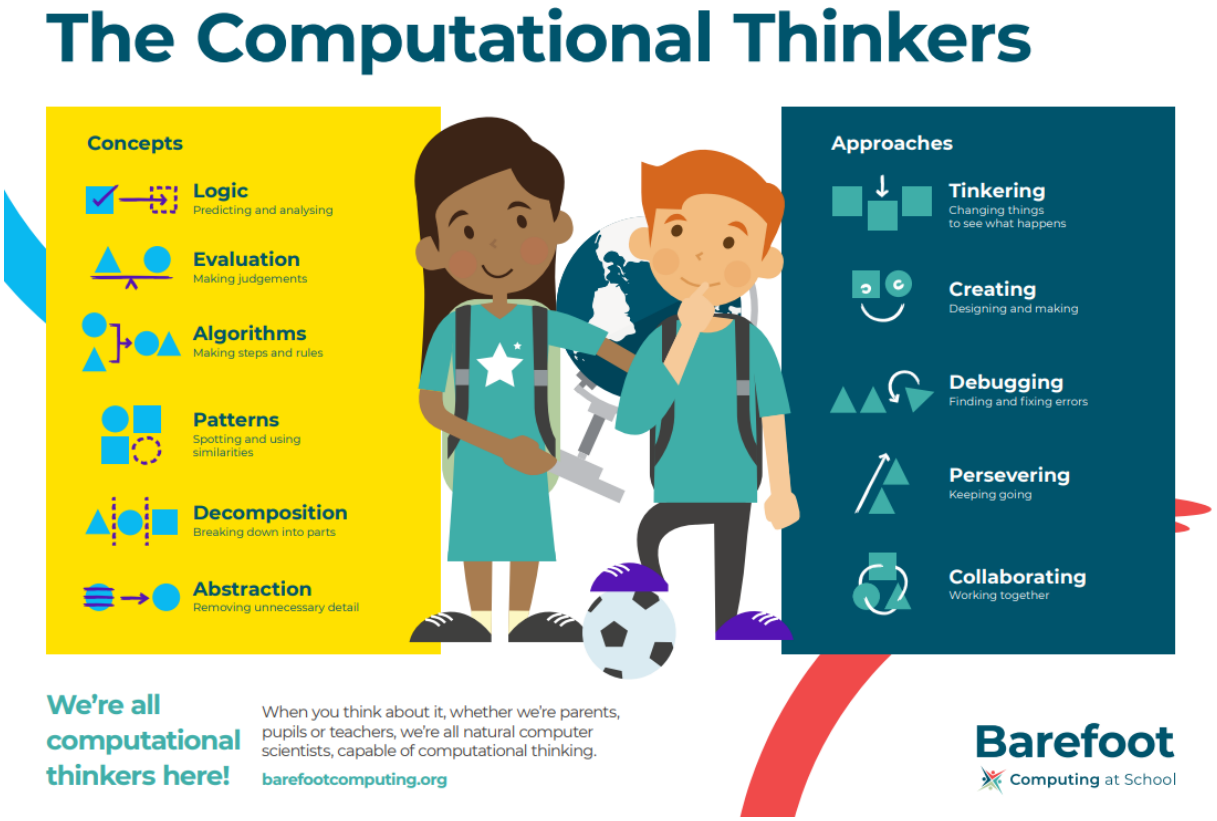
Utdanningsdirektoratet (2019) sin definisjon av algoritmisk tenkning og CSTA Standards Task Force (2016) sin definisjon av computational thinking trekker frem at det er en problemløsningsmetode. En studie gjennomført av Medeiros et al. (2018) viser at mange elever mangler strategier for å løse problemer i programmering. Elever opplever utfordringer med å analysere, planlegge og lage løsninger til et problem i programmering (Hazzan, 2011). Hazzan (2011) hevder at dette skyldes at det er for lite fokus på problemløsning. Videre påpeker Hazzan (2011) at dette fører til at elever utvikler egne og mindre effektive strategier, ofte prøv- og feile strategier. Sørby og Angell (2012) viser også til at prøv og feil-strategien blir mye brukt av elever som arbeider med programmering. For å forhindre dette mener Sørby og Angell (2012) at elevene må få oppfølging av læreren slik at de blir bevisst på hvilke strategier de kan bruke i programmeringen.

2.2 Den algoritmiske tenkeren

Utdanningsdirektoratet (2019) bruker "den algoritmiske tenkeren" som er en modell der hovedelementene ved algoritmisk tenkning trekkes frem (figur 1). Denne modellen er utarbeidet fra Barefoot computing (u.å-d) sin modell “The Computational Thinker” (figur 2). Hensikten med modellene er å danne et grunnlag for å løse problemer (Barefoot computing, u.å-d; Utdanningsdirektoratet, 2019). Den algoritmiske tenkeren inneholder seks nøkkelbegreper, som kalles for konsepter i The Computational Thinker. I tillegg inneholder den algoritmiske tenkeren fem arbeidsmåter som i The Computational Thinker er beskrevet som tilnærminger (Barefoot computing, u.å-d; Utdanningsdirektoratet, 2019). Disse nøkkelbegrepene og arbeidsmåtene opptrer nødvendigvis ikke alene, men er en del av en større helhet. Det vil derfor være vanlig at flere av nøkkelbegrepene henger sammen og at flere faller inn under de ulike arbeidsmåtene (Utdanningsdirektoratet, 2019).



Figur 1: Den algoritmiske tenkeren (Utdanningsdirektoratet, 2019)



Figur 2: The Computatinal Thinker (Barefoot computing, u.å-d)

2.2.1 Nøkkelbegreper

Logikk

Logisk resonnement handler om å kunne forklare hvorfor noe skjer eller kommer til å skje på bakgrunn av en forutsigbar respons (Barefoot computing, u.å-i). Dersom det blir brukt samme instrukser i samme program på ulike PC-er, gir det samme utfall. Barefoot computing (u.å-i) trekker frem at elever kan bruke logisk resonnement til å tenke gjennom hva hvert trinn gjør i en algoritme for å forutsi utfall (Barefoot computing, u.å-i). Dersom elevene opplever at algoritmen ikke fungerer etter hensikten vil det å tenke gjennom hva hvert trinn gjør bidra til at elevene oppdager hvor feilen er. Når årsaken til feilen er oppdaget kan logisk resonnement benyttes for å endre koden på en hensiktsmessig måte (Barefoot computing, u.å-i).

Barn kan forstå logikken i teknologiske programmer i tidlig alder og de utvikler raskt en mental modell for hvordan teknologien fungerer. Dette gjøres både gjennom å se på andre og gjennom å eksperimentere selv. De lærer for eksempel at for å få en person til å bevege seg på et spill må de trykke på en bestemt knapp, som gir en forutsigbar respons. Dermed vil logisk resonnement kunne bidra til å forutsi utfallet i et program (Barefoot computing, u.å-i).

Programvareingeniører bruker logisk resonnement når de utvikler nye og effektive koder. Alt datamaskinen gjør styres av logikk. Inne i datamaskinens prosessor blir hver handling som utføres brutt ned til logiske operasjoner basert på elektriske signaler. Programvareingeniører bruker de mentale modellene for hvordan programmeringsspråk, en maskinvare og et operativsystem fungerer. Når programvareingeniører tester nye programvarer eller søker etter feil (bugs) og fikser dem (debugging/feilsøking) stoler de på logiske resonnementer (Barefoot computing, u.å-i).

Algoritmer

Barefoot computing (u.å-b) beskriver algoritmer som en sekvens av instruksjoner eller et sett med regler for å få noe gjort. Disse sekvensene skal danne en fremgangsmåte for å løse et

problem (Sanne et al., 2016; Utdanningsdirektoratet, 2019). En algoritme kan utføres av både mennesker og datamaskiner, og i Utdanningsdirektoratet (2019) sin beskrivelse av algoritmisk tenkning blir det nevnt at elevene selv skal vurdere om det er hensiktsmessig å løse et problem ved hjelp av teknologi eller ikke. Eksempler på algoritmer som kan utføres av mennesker er artsbestemmelse i naturfag, matoppskrifter eller en reiserute. Algoritmer for reiseruter kan derimot være mer effektivt ved bruk av teknologi som GPS. For å gi elever trening i å lage effektive teknologiske algoritmer kan de arbeide med programmerbare leker eller roboter, som for eksempel Spheroball.

Abstraksjon

Det å abstrahere handler om å forenkle gjennom å hente ut den informasjonen som er relevant, og samtidig kvitte seg med irrelevant informasjon (Barefoot computing, u.å-a; Wing, 2017; Wing, 2008). Dermed kan abstraksjon innebære å representere noe på en forenklet måte med mindre grad av detaljer. I naturfag kan modeller være en abstraksjon av et større naturfaglig fenomen og grafer kan være en komprimert representasjon av et større datasett, der kun de nødvendige detaljene blir inkludert (Barefoot computing, u.å-a). På samme måte kan en kode som viser hvordan en robot skal bevege seg være en abstraksjon ved at alle unødvendige detaljer som hvordan roboten og omgivelsene ser ut er fjernet.

Mønstre

Det å finne mønstre handler om å se etter likheter og ulikheter (Barefoot computing, u.å-j). Et eksempel på mønstre i programmering er repetitive sekvenser. Dersom elevene finner slike sekvenser kan de lage generaliserende koder som kan settes inn i algoritmen de skaper. På denne måten bidrar mønstre til å finne en effektiv løsning på problemet (Anderson, 2016).

Dekomposisjon

Dekomposisjon handler om å dele opp problemet en står overfor i mindre deler slik at problemet blir mer håndterlig (Anderson, 2016; Barefoot computing, u.å-g). Gjennom å dele opp problemet i flere mindre delproblemer kan delproblemene løses hver for seg og dermed

være enklere å løse. I tillegg kan hver av partene i en gruppe løse hver sitt delproblem for så å sette det sammen til en helhetlig løsning på det overordnede problemet. Dette kan gjøre at hver av partene kan bidra med sin forståelse og sine ferdigheter til deler av problemet (Barefoot computing, u.å-g).

Evaluering

Underveis og til slutt skal elevene evaluere gjennomføringen av oppgaven ut ifra kriterier knyttet til et gitt problem. Dette innebærer å se på hva, hvordan og hvorfor de har programmert eller brukt andre metoder til å løse et problem (Sevik, 2016). Videre handler det om å vurdere produkter eller løsningers kvalitet og effektivitet opp mot formålet. For eksempel kan designmål, spesifikasjon og brukerbehov være kriterier som elevene kan vurdere ut ifra (Barefoot computing, u.å-h). Barefoot computing (u.å-h) trekker frem et eksempel hvor elever skal vurdere løsningens effektivitet på bakgrunn av hvilken rute som er kortest for en robot.

Det at elevene setter seg inn i programmeringen de har gjennomført kan bidra til at elevene reflekterer rundt egen tankegang (Brennan & Resnick, 2012). Videre trekker Brennan og Resnick (2012) frem at dette kan gjøres ved blant annet at elevene skal dokumentere egen prosess ved bruk av kommentarer i koden, lydinnspilte beskrivelser av koden eller ved å fortelle andre hva de har lært av programmeringen.

2.2.2 Arbeidsmåter

Fikle

Å fikle, her oversatt fra tinkering, handler om å løse problemer gjennom direkte utforskning og eksperimentering med en leken tilnærming (Dong et al., 2019). For å bli kjent med ny teknologi prøver en det ofte ut for å se hva det gjør og hvordan det fungerer. Dette gjøres gjennom å se på årsak-virkning (Barefoot computing, u.å-l). For eksempel ved å se på hva som skjer når den grønne knappen blir trykket på i et spill, eller hva som skjer når elever setter inn bestemte koder i et program. Selv om ideer kan virke feil kan de prøves slik at en ser hva som skjer og opparbeider seg erfaring med utfallet. Barefoot computing (u.å-l) mener

at dette er viktig for barns uavhengige læring. For eldre barn kan fikling være mer målrettet, med et større fokus på "utforsking og forbedring" (Barefoot computing, u.å-l).

Fikling vil kunne styrke elevenes evne til å tenke kreativt og tørre å prøve seg frem i andre situasjoner. Det er en grunnleggende ferdighet i den teknologiske verden der det skjer hyppige utviklinger og det vil kunne bidra til å se på disse endringene som muligheter fremfor utfordringer, noe som gjør det mulig å holde ferdighetene oppdatert og ta i bruk ny teknologi (Barefoot computing, u.å-l).

Skape

En sentral del av programmering er å forbedre allerede eksisterende produkter eller å skape noe nytt. Dette krever at elevene klarer å tenke kreativt for å finne gode løsninger. Ofte handler det også om å formulere et problem slik at elevene bevisstgjøres hva som skal lages eller løses (Barefoot computing, u.å-e). Det kan være å skape programvarer, fysiske gjenstander som for eksempel roboter eller å utarbeide algoritmer for robotens bevegelse. Videre handler det om å designe et produkt eller en løsning. Dersom løsningen ikke fungerer eller ikke er gunstig krever det evaluering og feilsøking for å gjøre hensiktsmessige endringer i koden (Barefoot computing, u.å-e).

Feilsøke

Feilsøking handler om å indentifisere feil og deretter rette dem opp når løsningen ikke fungerer som ønsket (Barefoot computing, u.å-f; Bers et al., 2014). Feilsøking kan ofte være tidkrevende, men prosessen kan være enklere hvis det gjøres underveis fremfor å starte feilsøkingen når hele koden er skrevet (Barefoot computing, u.å-f). På den måten kan en sørge for å vurdere om delproblemene er løst på best mulig måte individuelt, som igjen vil kunne optimalisere løsningen av det opprinnelige problemet (Barefoot computing, u.å-f; Shute et al., 2017). For å få til dette vil systematisk testing og modifisering være nødvendig (Shute et al., 2017). Barefoot computing (u.å-f) foreslår å feilsøke gjennom fire trinn, her oversatt av oss fra engelsk til norsk:

1. Forutsi hva som skal skje
2. Finne ut hva som skjer
3. Finne ut hvor noe har gått galt
4. Fikse det

Elever som har utviklet feilsøkingsstrategier vil kunne bruke strategiene i ulike programmeringsspråk. Dermed vil elevenes erfaring med programmering påvirke deres avgjørelser i feilsøkingen. For at elevene skal utvikle feilsøkingsstrategier må læreren ha fokus på at det å oppdage og undersøke feil bidrar til læring (Barefoot computing, u.å-f; Hazzan et al., 2015). Videre bør læreren gi elevene mulighet til å utforske problemet på egenhånd fremfor å gi dem svaret (Hazzan et al., 2015).

Holde ut

Det å holde ut innebærer å være tålmodig og ikke gi opp (Barefoot computing, u.å-k). Programmering krever ikke bare forståelse for algoritmer, koder og programmeringsspråk, men også viljen til å holde ut selv når en møter på utfordringer. I møte med ny teknologi kan en komme over flere utfordringer og det kan være behov for å teste ulike løsninger. Noen ganger vil det være behov for å endre den intuitive måten å tenke på til en mer tidkrevende og logisk måte å tenke på (Barefoot computing, u.å-k).

Dekomposisjon og feilsøking kan påvirke elevenes utholdenhet (Barefoot computing, u.å-k). Når elever står fast kan læreren oppfordre og/eller vise elevene hvordan de kan dele opp problemet i mindre deler slik at det blir enklere å løse. Videre bør elevene være kjent med de ulike trinnene i feilsøking slik at de vet hvordan de kan rette opp i feil som oppstår, i tillegg til at læreren oppmuntrer til å finne feil og rette dem (Barefoot computing, u.å-k).

Samarbeid

Samarbeid handler om å arbeide med andre (Barefoot computing, u.å-c). I følge Sevik (2016) er hensikten med samarbeid i programmering at elevene skal dra nytte av hverandres

kunnskaper og erfaringer gjennom samtaler og diskusjoner rundt utfordringer de møter på. Videre skal dette bidra til at elevene oppnår målet med oppgaven. Dette krever at elevene tenker kritisk og reflekterer rundt hvordan de kan løse oppgaven. Samarbeid kan motivere elever til å holde ut når de møter utfordringer (Barefoot computing, u.å-c). Det bør legges til rette for oppsummering og vurdering i fellesskap for å bidra til at elevene reflekterer rundt hva, hvordan og hvorfor de har programmert på den valgte måten (Sevik, 2016).

Ifølge Ludvigsen et al. (2015) innebærer samarbeid at elever kan planlegge, gjennomføre og vurdere oppgaver sammen for å nå et mål. Dette krever at elevene lærer seg strategier som å lytte til hverandre, inngå kompromisser og være villig til å teste andres løsninger fremfor å stå fast på sin egen. Videre mener Ludvigsen et al. (2015) at det kan være hensiktsmessig å knytte samarbeid mellom elever til arbeid med problemløsningsoppgaver og deltakelse i faglig diskusjon. Gjennom å samarbeide og øve på slike samarbeidsstrategier skal elevene oppleve det som verdifullt å bidra i fellesskapet, samtidig som at de skal respektere andre og vise omsorg for medelever.

Wassermann og Haukeland (2001) hevder at det å ta ansvar og kunne samarbeide gjør det lettere å bli akseptert i en gruppe enten det er på fritidsaktiviteter, i klasserommet eller på jobb. Derimot vil det å ikke kunne samarbeide eller ta ansvar føre til avvisning fra sosiale grupper (Wassermann & Haukeland, 2001). Videre påpeker Aakervik et al. (2006) at det å lære elever å samarbeide i ulike læringssituasjoner i klasserommet vil kunne føre til at elever blir flinkere til å forstå andres perspektiver og tanker. Det vil også gjøre elevene flinkere til å håndtere konflikter ifølge Aakervik et al. (2006). Ludvigsen et al. (2015) hevder imidlertid at sosiale ferdigheter er noe som må trenes opp og som læres over tid. Dersom det er forventet at elevene skal kunne samarbeide må de også ha lært hvordan de skal samarbeide (Ludvigsen et al., 2015).

Klassemiljøet kan påvirke samspillet mellom elevene. Trygt klassemiljø vil gi rom for ulike synspunkter og uenighet ifølge Bergkastet et al. (2019). Elevene skal føle seg trygge nok til å stille spørsmål ved det de lurer på (Säljö, 2001). Gjennom å stille spørsmål og svare på

spørsmål får elevene testet sin egen forståelse samtidig som de kan få en dypere forståelse av andres kunnskaper og ferdigheter. Kommunikasjon om forståelse er et sentralt aspekt i samarbeidet ettersom mennesker forstår ting ulikt. For å oppnå forståelse for hverandre må elevene kunne sette seg inn i det medelever formidler samtidig som de må gjøre seg selv forstått. Gjennom å sette seg inn i andres synspunkter kan elevene ende opp med å endre sitt opprinnelige synspunkt (Säljö, 2001).

Parprogrammering, kalt pair programming på engelsk, er en samarbeidsform som ofte blir benyttet ved programmering. Det innebærer at to elever deler en PC og arbeider sammen om å programmere og skape koder. Den ene er "sjåfør" og skriver koder og styrer programmet, mens den andre er "kartleser" som passer på at de beveger seg i riktig retning (Barefoot computing, u.å-c; Sevik, 2016). For at kartleseren skal sørge for at de er på riktig vei er det å stille spørsmål og se etter feil eller mulige endringer en del av jobben (Beck et al., 2005; Tsai et al., 2015). Det er viktig at alle parter bidrar og er ansvarlig for egen rolle for at samarbeidet skal fungere (Barefoot computing, u.å-c). Videre hevder Barefoot computing (u.å-c) at elevene skal bytte rolle regelmessig slik at begge elevene får erfaringer med de to rollene.

Lærerens rolle i samarbeid mellom elevene

Læreren skal fungere som organisator og veileder i samarbeidet mellom elevene (Aakervik et al., 2006), og bidra til at elevene samarbeider på en god måte. For å få til dette krever det først og fremst at elevene utvikler ferdigheter innen samarbeid (Blatchford et al., 2003). Dette kan innebære kommunikative ferdigheter, som å lytte, forklare og dele ideer, samt ferdigheter som å planlegge og organisere gruppens arbeid og lære å stole på hverandre og vise respekt for hverandre (Blatchford et al., 2003). Videre bør læreren være godt kjent med elevene som skal settes sammen til samarbeidsgrupper og undervisningen skal være godt organisert og planlagt (Blatchford et al., 2003). Hertz-Lazarowitz (2008) trekker frem at en viktig del av organiseringen handler om å forberede de fysiske rammene, altså rommet der undervisningen skal foregå.

I tillegg trekkes det frem at selve aktiviteten gruppene skal gjennomføre bør være engasjerende og passe utfordrende for elevene. Stenseth et al. (2019) understreker at det er i planleggingen av undervisningen lærerens kunnskaper om programmering er viktigst. Når undervisningen er i gang blir lærerens rolle i hovedsak å følge opp elevene i deres samarbeid gjennom observasjon, samt å svare på eventuelle spørsmål fra eleven (Stenseth et al., 2019).

2.3 Programmering

Programmering dreier seg om å utarbeide instruksjoner som bestemmer hvordan en datamaskin skal oppføre seg og reagere ut ifra forskjellige kommandoer (Sanne et al., 2016; Stenseth et al., 2019). Disse instruksjonene må fremtre som algoritmer i et programmeringsspråk som datamaskinen kan forstå og tolke (Sanne et al., 2016; Taraldsen & Myhra, 2019). Algoritmene deles opp i koder som igjen må struktureres korrekt ettersom det er koden som forteller hvilken handling som skal utføres (Sanne et al., 2016; Taraldsen & Myhra, 2019). Disse kodene kan skrives i ulike programmeringsspråk. Programmeringens komplekse sammensetning krever at de som skal programmere har kunnskap innen programmeringsspråket (Stenseth et al., 2019).

I skolen er det blant annet vanlig å bruke programmeringsspråket Scratch (https://scratch.mit.edu/), som er en form for blokkprogrammering. Blokkbasert programmering åpner for å la nybegynnere utforske og eksperimentere i programmering. Fordelen med dette programmeringsspråket er at elevene kan bruke ferdiglagde koder i form av blokker. Ettersom kommandoene allerede er formulert i eksisterende blokker, med ulike farger etter hva slags type kommando blokken gir, vil dette redusere skrivefeil gjort av den som programmerer (Begel, 1996; Maloney et al., 2010). På den måten blir fokuset rettet mot det å strukturere koder ved å sette sammen blokkene til en algoritme (Begel, 1996; Maloney et al., 2010; Taraldsen & Myhra, 2019). Videre har det blitt vist at bruken av blokkbasert programmering øker effektiviteten hos nybegynnere når de programmerer (Price & Barnes, 2015; Weintrop & Wilensky, 2017).

Blokkbaserte programmeringsprogrammer har blitt designet for å oppmuntre til fikling, gjennom utforsking og eksperimentering, der det er ment at brukerne skal utforske de ulike blokkene og hvordan utfordringene kan løses på ulike måter (Resnick et al., 2009; Taraldsen & Myhra, 2019). Fikling åpner for kreativitet og sørger for at elever får utforsket i direkte kontakt med programmet.

2.3.1 Hvordan programmering er implementert i læreplanen

Programmering ble en del av læreplanen i 2020 (Utanningsdiraktoratet, 2019), og er dermed blitt obligatorisk i skolen (Tellefsen, 2021a). Stortinget har vedtatt at programmering ikke skal være et eget fag, men at det skal implementeres i fagene matematikk, naturfag og kunst og håndverk (Flø, 2021). I naturfag finner vi programmering under kjerneelementet *teknologi*, der det presiseres at elever både skal forstå programmering, samt bruke det til å skape noe (Utdanningsdirektoratet, 2020a). Videre er en av de grunnleggende ferdighetene i naturfag digitale ferdigheter, som blant annet innebærer at elevene skal kunne bruke digitale verktøy til å programmere (Utdanningsdirektoratet, 2013). Programmering kommer eksplisitt frem i kompetansemål etter 7. trinn der det står at elevene skal: *"utforske, lage og programmere teknologiske systemer som består av deler som virker sammen"* (Utdanningsdirektoratet, 2020b) og etter 10. trinn hvor elevene skal: *"bruke programmering til å utforske naturfaglige fenomener"* (Utdanningsdirektoratet, 2020c).

2.3.2 Spheroballer

Spheroballene er kuleformede roboter på størrelse med en tennisball. De består blant annet av en motor, lysdioder og en prosessor omgitt av et rundt, gjennomsiktig deksel i plast (figur 3A). Ballene kan programmeres gjennom en app (figur 3B) som kommuniserer med Spheroballene gjennom Bluetooth, med en rekkevidde på 30 meter. Spheroballen skal gi elever mulighet til å utrykke ideer, trene på programmering og lære mer om robotikk, gjennom lek og praktiske aktiviteter. I tillegg har ballen en LED-lys funksjon som gjør at elevene kan programmere ballen til å lyse i ulike farger (Sphero Inc, 2022).

Elevene kan kode gjennom tegning, blokkbasert programmering eller tekstprogrammering (Taraldsen & Myhra, 2019). Dette gir elevene mulighet til å gå fra tegning til blokkbasert programmering eller fra blokkbasert programmering til tekstprogrammering. I undervisningen som ble observert i denne studien brukte elevene blokkbasert programmering for å programmere Spheroballer.

A B

Figur 3: Spheroball (A) og programmering av Spheroball i app på iPad (B)

2.3.3 Fysisk programmering

Fysisk programmering handler om å programmere fysiske gjenstander som kan vekselvirke med omverdenen. De fysiske gjenstandene kan for eksempel være en micro:bit (https://microbit.org/) eller en Spheroball (https://sphero.com/). Det å arbeide med fysisk programmering støttes av Paperts konstruktivisme, som handler om at læringen skjer når eleven er engasjert i konstruksjonen av en virkelig gjenstand (Hodges et al., 2020). I fysisk programmering kan det bli lettere for elever å visualisere og videre realisere løsningene sine ettersom de kan lage koden inne i et program for så å teste den på den fysiske gjenstanden (Flowers & Gossett, 2002).

Fysisk programmering kan bidra til økt engasjement og motivasjon hos elever ettersom læringserfaringen og resultatene er synlige gjennom enheten som brukes i programmeringen (Hodges et al., 2020). I tillegg kan det gi en opplevelse av belønning når de klarer å bygge noe

(Hodges et al., 2013). Ettersom det finnes flere løsninger til en utfordring fremmer fysisk programmering feilsøking ved at elevene lærer gjennom å gjøre ulike feil og undersøke hvordan de kan rette opp i feilene (Hodges et al., 2020). Programmeringen åpner også for samarbeid og konkurranse, som ligger i oppgavenes og utfordringenes natur (Hodges et al., 2020). I tillegg kan elevene oppleve fysisk programmering som noe morsomt, noe som igjen fører til at de holder ut lenger i arbeidet (Przybylla & Romeike, 2015).

2.3.4 Pedagogiske tilnærminger til programmering

Schulz og Pinkwart (2016) har utviklet en modell som foreslår en fremgangsmåte i arbeid med fysisk programmering. Modellen består av de fire fasene forberedelse, implementering, utførelse og evaluering. *Forberedelse* innebærer å formulere et mål, legge en plan for å nå målet, forutse utfordringer og dele opp problemet. *Implementeringen* er selve kodingen og i *utførelsen* testes koden og resultatene observeres. Til slutt må resultatene *evalueres,* og eventuelle feil må identifiseres og rettes opp.

Schulz og Pinkwart (2016) gjennomførte en studie der de analyserte i hvor stor grad modellen var egnet til å beskrive elevers arbeid med fysisk programmering. Resultatene viste at elevene ofte slet med å overføre ideene sine til selve kodingen. Videre hadde elevene en tendens til å ignorere utfallet av koden dersom det ikke var ønsket utfall, noe som resulterte i mindre grad av feilsøking og økt grad av prøv-og-feil. Dermed var elevenes feilsøking mindre målrettet og større deler av koden ble endret enn det som egentlig var nødvendig. For å håndtere disse utfordringene foreslår Schulz og Pinkwart (2016) at læreren skal gjøre modellen tilgjengelig for elevene og oppmuntre dem til å reflektere rundt de ulike fasene samtidig som de programmerer. Dette kan bidra til å håndtere elevenes kognitive belastning i programmeringsoppgaver.

Use-Modify-Create er et pedagogisk rammeverk som oppfordrer elevene til å programmere gjennom å ta utgangspunkt i andres koder (Lee et al., 2011). Målet med denne arbeidsmåten er at elevene skal skape deres egen kode. Veien dit starter imidlertid med at elevene skal bruke et program som allerede er skapt av noen andre. Deretter skal de modifisere

programmet slik at det blir deres (Lee et al., 2011). Det å gjenbruke og videreutvikle andres koder har vært brukt i lang tid i programmering (Brennan & Resnick, 2012). Det kan innebære å gjøre små endringer som å bytte farge på en figur, eller mer omfattende endringer som hvordan en figur skal oppføre seg (Lee et al., 2011). Dette kan bidra til at elever kan lage mer komplekse koder enn de hadde klart å utvikle på egenhånd (Brennan & Resnick, 2012). Det er når elevene ønsker å gjøre en justering som innebærer at en kode må endres at de inne i modifiseringsfasen (Lee et al., 2011). På denne måten kan elevene oppnå en progresjon i programmering gjennom å kopiere og analysere andres arbeid, for så å overføre det til eget arbeid.

3 Tidligere forskning

I dette kapittelet vil vi presentere tidligere forskning innenfor programmering i naturfag. Vi har inkludert forskning om inkludering av fysisk programmering i naturfag og studier som belyser pedagogiske tilnærminger og utfordringer i programmering. Tillegg trekker vi frem studier som handler om ulike strategier og arbeidsmåter lærere og elever benytter i naturfagundervisning med programmering.

3.1 Inkludering av fysisk programmering i naturfagundervisning

Sentance et al. (2017) har gjennomført en studie av hvilke tilnærminger lærere velger å bruke når de har undervisning i fysisk programmering. I dette arbeidet kom det frem eksempler på lærere som brukte allerede eksisterende koder for å hjelpe elevene til å forstå hvordan kodingen kan se ut før elevene laget sin egen kode. Dette er et eksempel på hvordan programmeringen kan foregå gjennom å bruke modellen Use-Modify-Create. En av elevene i denne klassen understreket at det var nyttig å bruke en kode som eksempel for så å kunne undersøke hvordan noe lignende kunne gjøres i elevens kode (Sentance et al., 2017).

Basert på resultatene fra studien sin har Sentance et al. (2017) kommet med flere forslag til hvordan skoler kan håndtere inkluderingen av fysisk programmering i undervisningen. Disse forslagene kan bidra til å sikre en undervisning preget av motivasjon og effektivitet.

Forslagene er utarbeidet basert på Storbritannias satsning på introduksjonen av micro:bit i skolen, og kan dermed fungere som hjelp til land som vurderer en lignende satsning eller som allerede er i gang med et slikt prosjekt. Forslagene presenteres under i en versjon som vi har oversatt til norsk og forkortet noe:

- Lærere trenger tid til forberedelse i form av trening, planlegging og tilrettelegging av læreplanmål
- Det bør konsulteres med en lærer som har kunnskap innenfor programmering for å sikre en god pedagogisk tilnærming
- Det bør rettes oppmerksomhet rundt tidsperioden elevene jobber med programmeringen, der det er gunstig at perioden varer over flere uker med flere støttende aktiviteter og sammenheng med læreplanmålene
- Lærere bør få profesjonell opplæring

Chevalier et al. (2020) har i en studie undersøkt hvordan strukturering av elevers arbeidsprosess kan påvirke elevers valg av strategier når de programmerer. I studien kommer denne strukturingen frem ved at lærerne blokkerer elevenes tilgang på programmet de skal programmere i. Forskerne har brukt en modell kalt CCPS (creative computational problem solving) som beskriver hvilke stadier elever bør gå igjennom når de skal bruke roboter til å utvikle algoritmisk tenkning. Denne modellen består av de fem fasene: forstå problemet, danne ideer, formulere robotens oppførsel, programmere robotens oppførsel og evaluere løsningen. I tillegg la forskerne til en sjette fase kalt "oppførsel utenom oppgaven".

For å undersøke hvordan strukturering av elevenes arbeidsprosess påvirket elevene ble elevene delt inn i kontrollgrupper og testgrupper. I kontrollgruppene fikk elevene fri tilgang på programmet de skulle programmere i. De fikk ingen begrensninger og ingen instruksjoner om hvordan de skulle gå frem. I testgruppene ble elevene begrenset ved at deres tilgang på programmet ble blokkert de første ti minuttene. Deretter kunne de bruke appen og utforme koder. Senere ble de igjen begrenset ved at de ikke fikk lov til å utføre koden på roboten, før de igjen fikk tilgang på alle hjelpemidler.

Studien viser at elever som fikk mye frihet og ingen instruksjoner endte opp med å lage en kode og teste den. Denne prosessen fremstår som en prøve-feile-strategi der feilsøkingen ikke skjer systematisk. Elevene gjentok denne strategien gjennom hele undervisningen og endret dermed ikke strategi. Dette hevder Chevalier et al. (2020) kan begrense elevens mulighet til å utvikle andre viktige strategier innenfor algoritmisk tenkning.

Chevalier et al. (2020) trekker frem at elevene i testgruppene, som ble begrenset i ulik grad, utnyttet tiden bedre enn elevene i kontrollgruppene. Videre brukte elevene flere strategier, som for eksempel at de brukte de første minuttene på å forstå problemet. I tillegg brukte de roboten og banen til å uttrykke og forklare egne ideer og tanker. Det å gi elever begrensinger og instruksjoner når de arbeider med roboter i programmering kan bidra til å utvikle elevers kognitive prosesser som å forstå et problem, danne ideer og formulere en løsning. Dette kan bidra til at elevene evner å lage en plan for hvordan de skal tilnærme seg en utfordring i programmering som inkluderer roboter.

3.2 Pedagogiske tilnærminger og utfordringer

I en engelsk studie fra 2014 gjennomført av Sentance og Csizmadia (2017) ble det gjort en undersøkelse av 339 lærere som fikk spørsmål knyttet til hvilke pedagogiske strategier de mente fungerte godt i programmeringsundervisning og hvilke utfordringer de møtte på i programmeringsundervisning.

I studien har lærere rapportert hvilke utfordringer de opplever i møte med programmering. Det skilles mellom utfordringer som knyttes direkte til læreren, til elevene og til ressursene, der utfordringer med ressursene i stor grad handlet om tekniske problemer. Sentance og Csizmadia (2017) trekker blant annet frem læreres pedagogiske og faglige kunnskap i faget som en utfordring for læreren. Lærerne uttrykker bekymring rundt egen kunnskap i programmering selv om de har deltatt på kurs og brukt flere timer på å sette seg inn i det. Videre viser studien at lærere syntes det er vanskelig å differensiere oppgavene i

programmering ettersom elevene ofte ligger på ulikt nivå, der noen har raskere fremgang enn andre.

Under utfordringer som knyttes direkte til elevene trekker lærerne i studien blant annet frem elevenes mangel på grunnleggende forståelse. Elevene slet med problemløsning ved at de syntes det var utfordrende å bryte ned problemet i mindre deler. Samtidig trakk lærerne frem at elevene også syntes det var utfordrende å sette sammen alt de hadde lært til å bli en lengre algoritme (Sentance & Csizmadia, 2017).

Studien peker på viktigheten av å utvikle elevenes algoritmiske tenkning gjennom å lære dem strategier som dekomponering, abstraksjon, algoritme og finne mønstre. Videre legges det til at det å finne feil og lære av feilene er grunnleggende for at elevene skal utvikle programmeringsferdigheter. Dette krever tålmodighet og evnen til å stå i et problem. Dette mente flere av lærerne i studien at var en gjennomgående utfordring for elevene. Flere elever ville at læreren skulle rette opp feilen for dem. I studien mener de at utfordringen med feilsøking kan skyldes at elever blir lært opp til å søke etter riktig svar fremfor å søke etter feil. I tillegg viser studien at det bare var tre av de 339 lærerne som hadde strategier for å støtte elevene i å stå i et problem. De tre lærerne som trakk frem strategier for å støtte elevene i å stå i et problem mente at læreren måtte oppfordre elevene til å finne ut av feilene på egenhånd uten for mye innblanding av læreren (Sentance & Csizmadia, 2017).

Flere av lærerne i studien trekker frem at det er viktig at elevene er aktive og involvert i læringen, at elevene får åpne oppgaver som gir rom for utforsking og problemløsning og at elevene får mulighet til å diskutere i grupper (Sentance & Csizmadia, 2017). Videre mente lærerne at samarbeid hadde en motiverende effekt på klassen, gruppene og elevene selv. Lærerne trekker frem parprogrammering og jevnaldrende veileder, oversatt av oss fra engelsk til norsk, som to av samarbeidsstrategiene de hadde positiv erfaring med i programmering. Dermed ville de anbefale disse strategiene til andre lærere som skal gjennomføre programmering i skolen. En jevnaldrende veileder var en samarbeidsstrategi som gikk ut på at

en elev med mer kunnskap og erfaring kan veilede andre elever (Sentance & Csizmadia, 2017).

Tyrén et al. (2018) gjennomførte en studie med hensikt å undersøke hva som er viktig å tenke på i utviklingen av et undervisningsopplegg med BBC micro:bit for å trene elevenes computational thinking. Gjennomføringen foregikk i 21 workshops i forskjellige skolemiljøer med elever fra 4. til 6. trinn. En utfordring som kom frem var internettforbindelsen. De gangene internettet var tregt eller at de mistet internettforbindelsen ble elevene frustrerte og det gikk bort læringstid til feilsøking av internettforbindelse (Tyrén et al., 2018).

Studien til Tyrén et al. (2018) viser at elevene foretrakk muntlige svar eller instruksjonsvideoer da de hadde behov for hjelp. Introduksjonsvideoer kan være hensiktsmessig dersom størrelsen på elevgruppen gjør det utfordrende for læreren å hjelpe alle. Dermed kan læreren bruke tiden på å hjelpe elevene som har størst behov for hjelp. Videre trekker studien frem at elevene har behov for å forstå hvordan programmet fungerer og hvordan de kan bruke programmet (Tyrén et al., 2018).

3.3 Fikling i blokkbasert programmering

I en studie gjort av Dong et al. (2019) har forfatterne forsøkt å belyse begrepet fikling i forbindelse med åpne programmeringsoppgaver ved bruk av blokkbaserte programmer. I åpne oppgaver får elevene et mål de skal nå, men veien dit kan variere. Dong et al. (2019) hevder slike oppgaver gir elevene bedre mulighet til å utforske fritt og benytte seg av kreative løsninger til det gitte problemet. Dermed forventet de å se økt bruk av fikling i åpne oppgaver enn i lukkede "steg-for-steg"-oppgaver. I studien har de definert fikling som den oppførselen elevene viser når de tester, utforsker og står fast med kodene deres for å oppnå et mål i programmeringsoppgaven. Videre mener de at en viktig del av fiklingen er når elever uttrykker usikkerhet og tvil. Dersom elevene vet nøyaktig hvordan de skal gå frem for å løse problemet, og gjør dette i et høyt tempo uten tvil underveis, vil det ikke defineres som fikling.

I studien har forskerne forsøkt å identifisere og kategorisere hvilke former for fikling elevene benyttet seg av. Målet med dette var å lære mer om hvordan elever angriper et problem, samt identifisere former for produktivt slit. De har delt fikling inn i tre kategorier: test-basert fikling, prototype-basert fikling og konstruksjonsbasert fikling (vår oversettelse). Konstruksjonsbasert fikling handler om at elevene bruker mye tid på å gjøre store endringer i koden uten å teste underveis. De to førstnevnte formene for fikling vil bli beskrevet i mer detalj under.

Test-basert fikling handler om at elevene tar for seg en spesifikk del av koden de lager. Forskerne har videre delt det inn i to underkategorier, *feilsøking* og *kjøring* (Dong et al., 2019). *Feilsøking* handler om at eleven prøver og feiler for å rette opp i en feil. Feilen kan oppdages ved å gjøre små endringer og teste programmet jevnlig. Ved å jobbe på denne måten får elever erfaring med egenskapene til de ulike blokkene og lærer forskjellen på ønsket utfall og faktisk utfall. Forskerne understreker imidlertid at nybegynnere kan ha en tendens til å feilsøke uten å gå systematisk til verks. Dette så de i studien der en elev hadde kodet feil, og ikke kom seg videre fordi eleven ikke skjønte hvor i koden feilen lå. Dermed endte eleven opp med å forsøke å bytte ut ulike blokker i koden med andre tilfeldige blokker. Disse blokkene ga helt andre kommandoer, noe som førte til et annet utfall enn utfall da koden ble testet. Samtidig kan en argumentere for at eleven i et slikt tilfelle har funnet en blokk som uten tvil kan elimineres fra løsningen.

Kjøring handler om at elevene kjører gjennom den samme koden gjentatte ganger uten å gjøre endringer mellom hver gjennomkjøring. Dong et al. (2019) argumenterer for at en slik gjennomkjøring både kan være positiv og negativ. Dersom flere gjennomkjøringer av koden bidrar til at eleven forstår koden bedre vil dette gi gjennomkjøringen en positiv rolle. Uttrykker elevene derimot frustrasjon og tror at flere gjennomkjøringer vil kunne føre til at koden virker etter hvert, vil kjøringen av koden virke negativt.

Prototype-basert fikling handler om at elevene utvikler en kode for en spesifikk del av programmet. Dette kan gjøres både *i* programmet og *utenfor* programmet. Uansett lages prototypen ferdig og testes slik at koden fungerer som ønsket før den implementeres i programmet. I tillegg kan elevene prototype en bestemt delfunksjon og når den fungere kan delfunksjonen kopieres opp og brukes videre i programmet.

3.4 Parprogrammering

Det har blitt gjort mye forskning på parprogrammering opp igjennom årene. Dybå et al. (2007) gjennomførte en litteraturstudie for å undersøke empiriske bevis som støtter at parprogrammering er fordelaktig fremfor å programmere alene. Områdene de vektla i litteraturgjennomgangen var faktorer knyttet til effektiviteten av parprogrammering, der de trakk frem varighet, innsats og sluttproduktets kvalitet. I analysen kom de frem til at parprogrammering er mer effektivt med tanke på varighet og kvalitet sammenlignet med når elever programmerer alene. Når to personer programmerer sammen kan de bruke mindre tid på å løse problemet, og sluttproduktets kvalitet kan være bedre. Parprogrammering vil imidlertid kreve flere timer dersom en legger sammen timene hver person innad i paret bruker. Dermed kom Dybå et al. (2007) frem til at det sammenlagt krever mer innsats fra elevene å jobbe med parprogrammering. Studien trekker også frem sammenhengen mellom kvalitet og innsats, ved at høyere kvalitet krever høyere innsats. Til slutt understreker de at parprogrammeringens påvirkning på økt effektivitet avhenger av elevenes kunnskap, samt programmeringssystemets og oppgavens kompleksitet.

Muller og Padberg (2004) har forsket på hva gruppesammensetningen har å si for parprogrammeringen. Ettersom det er en iboende sosial interaksjon mellom to personer i parprogrammering er det naturlig å se på hvordan dynamikken mellom personene påvirker arbeidsmåten. Muller og Padberg (2004) rapporterte at det eksisterer en korrelasjon mellom gruppens prestasjon og i hvor stor grad gruppemedlemmene er komfortable i samarbeidet. Dette støttes av tidligere studier som viser at elever som opplevde å ha, det de selv mente var en inkompatibel partner førte til at de mislikte arbeidsmåten (McDowell et al., 2003). Likevel understreker Muller og Padberg (2004) at de ikke med sikkerhet kan si om elevparet presterer høyt *fordi* de føler seg komfortable i samarbeidet eller om de føler seg komfortable i samarbeidet *fordi* de presterer høyt.

Tsai et al. (2015) har gjennomført en studie der det ble undersøkt hvordan elevens kognitive belastning varierer avhengig av om de jobber individuelt eller i par. Elevene ble delt inn i tre ulike grupper. Noen elever arbeidet alene, noen samarbeidet ansikt til ansikt og noen samarbeidet ved hjelp av teknologi, altså de satt ikke i samme rom under gjennomføringen. I studiens resultater kom det frem at elevenes kognitive belastning var betydelig høyere da de jobbet alene og samarbeidet adskilt enn da de jobbet i par. Videre viser resultatene fra intervjuer gjort i studien at elevene selv foretrakk å arbeide i par, ansikt til ansikt.

4 Metode og analyse

Denne studien har som hensikt å undersøke hvordan elever på mellomtrinnet arbeider med programmering i naturfagundervisning, og utfordringer som kan oppstå i planlegging og gjennomføring av slike undervisningsopplegg. Dette har vi gjort gjennom å observere fire klasser som gjennomførte en programmeringsundervisning i naturfag, samt intervjue læreren som hadde hovedansvaret for planleggingen av undervisningsopplegget og en lærer som underviste i to av de fire klassene. I disse to klassene ble elevene plassert i grupper på to til tre elever.

I dette kapittelet skal vi først presentere hvilke metoder for datainnsamling vi har benyttet. Videre vil vi begrunne utvalg og rekruttering av informanter. Deretter presenteres utvikling av intervjuguide og observasjonsskjema, kontekst til studien og gjennomføring. Til slutt vil vi beskrive analysemetoden vi har brukt før vi diskuterer studiens reliabilitet, validitet og forskningsetikk.

4.1 Datainnsamlingsmetode

I denne studien er kvalitativt forskningsintervju benyttet. Hensikten med kvalitativt forskningsintervju er å få et innblikk i informantenes erfaringer knyttet til et fenomen sett fra informantenes perspektiv og kan dermed gi tilgang til deres personlige oppfatninger (Kvale et

al., 2015). I denne studien ønsket vi å se på hvordan lærere tilrettelegger for programmering i naturfag og læreres personlige oppfatning av elevers arbeid med algoritmisk tenkning og utfordringer. Dermed var det hensiktsmessig å intervjue lærere.

Videre har det blitt benyttet observasjon som metode fordi vi ønsket direkte tilgang til hvordan elevene arbeidet med programmering med vekt på strategier og utfordringer. Observasjon gir muligheten til å se det ikke-verbale og fysiske som skjer (Cohen et al., 2018). Ettersom det kan være vanskelig for elever å se tilbake og sette ord på hva som var utfordrende var det hensiktsmessig for oss å observere elevene i det øyeblikket de møtte på utfordringen.

Under observasjonen var vi observerende deltakere. Observerende deltakelse innebærer at forskeren er en tilskuer som i liten grad deltar i samhandlingen. Det vil likevel være noe samspill mellom forskeren og de som observeres (Fangen, 2019). Som deltakende observatør har forskeren mulighet til å engasjere seg gjennom spørsmål underveis (Christoffersen & Johannessen, 2012). Ettersom vi ønsket et autentisk blikk på den konkrete situasjonen, var det hensiktsmessig å være tilskuere, men samtidig ha mulighet til å stille spørsmål ved behov.

4.2 Utvalg og rekruttering av informanter

Det er blitt benyttet kriteriebasert utvalg. I kriteriebasert utvelgelse velges informanter som oppfyller spesielle krav (Christoffersen & Johannessen, 2012). Gjennom kriteriebasert utvalg endte vi opp med et utvalg bestående av to lærere og fire klasser på 7. trinn på en skole i Oslo. I denne studien var kriteriene at informantene skulle gjennom et undervisningsopplegg om programmering i naturfag. Videre var det et kriterium at læreren som ble intervjuet gjennomførte undervisning med elever på mellomtrinnet som vi kunne observere. Informantene som oppfylte kriteriene, ble skaffet gjennom oppsøkende kontakt rettet mot lærere. Dette resulterte i en informantgruppe bestående av en lærer som stod for planleggingen av undervisningsopplegget og en introduksjonsundervisning, en lærer som underviste elever i undervisningsopplegget i to av klassene og de fire klassene som skulle gjennomføre undervisningsopplegget.

De to lærerne i utvalget hadde ulike utdanninger og bakgrunn. Læreren som planla undervisningsopplegget, har gått grunnskolelærerutdanning 5-10, og har studert naturfag og programmering. I tillegg har læreren jobbet med programmering i skolen. Læreren som underviste i denne programmeringsundervisningen har verken studert naturfag eller programmering, men læreren har deltatt på et kodekurs som var preget av koronapandemien. Elevene hadde erfaring med blokkprogrammering, og i tillegg hadde læreren som planla undervisningen en introduksjonsøkt med elevene der de lærte å bruke Spheroball.

4.3 Utvikling av intervjuguide og observasjonsskjema

Det ble utviklet et observasjonsskjema (vedlegg 1) og to semistrukturerte intervjuguider (vedlegg 2 og 3) på bakgrunn av oppgavens teoretiske forankring, tidligere forskning og forskningsspørsmålene:

- Hvilke kjennetegn på algoritmisk tenkning kommer til utrykk i elevers arbeid med programmering i naturfag på mellomtrinnet?
- Hvilke utfordringer kommer frem i et undervisningsopplegg med programmering i naturfag på mellomtrinnet?

Semistrukturert intervju kan verken kategoriseres som en åpen eller en lukket samtale, men kan holdes i en hverdagslig tone med utgangspunkt i spørsmålene fra intervjuguiden (Kvale et al., 2015). Et semistrukturert intervju gir også mulighet til å stille oppfølgingsspørsmål (Kvale et al., 2015). Dette vil være hensiktsmessig for å få oppklaring i utydelige svar, for at informanten kan utdype svarene i større grad og for å bidra til at samtalen flyter.

Begge informantene ble spurt om utdanning og erfaring med programmering, deres definisjoner av programmering, algoritmisk tenkning og den algoritmiske tenkeren for å få en innsikt i deres forståelse av begrepene til videre spørsmål om dette. Videre gjorde vi noen endringer fra intervjuguiden av læreren som underviste til læreren som planla undervisningen

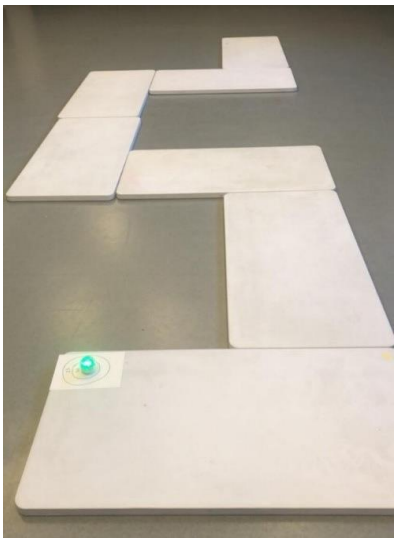
fordi de to lærerne hadde ulike roller. Læreren som underviste ble spurt om det ble gjort tilpasninger underveis og hvilke utfordringer som oppstod i undervisningen, mens planleggeren ble spurt om utfordringer og tilrettelegging knyttet til planleggingen.

For å besvare forskningsspørsmålene var observasjonens fokus strategier og utfordringer med vekt på den algoritmiske tenkeren. På bakgrunn av at alle observasjoner er ulike og at det ikke er mulig å forutse hvordan situasjoner utspiller seg vil det være hensiktsmessig å være åpen for andre observasjoner. Derfor hadde vi et åpent observasjonsskjema bestående av en kolonne til observasjon og en til tolkning. Dette for at det skal være tydelig hva som er vår egen tolkning og hva som ble observert. Videre er dette hensiktsmessig fordi det i ettertid kan vurderes om observasjonen er relevant for studien, ut ifra observasjonen og tolkningen som er gjort (Dalland & Keeping, 2020; Postholm, 2005).

4.4 Kontekst til studien

I denne studien har vi observert fire klasser på 7. trinn som har gjennomført en undervisning i programmering i naturfag. Undervisningen var planlagt av en lærer og gjennomført av andre lærere. Læreren som planla undervisningen, hadde mer erfaring og kunnskap om programmering enn læreren som gjennomførte. Undervisningen hadde som mål å trene elevenes ferdigheter i programmering. Dette fordi teknologi er et kjerneelement i naturfag og videre er programmering et verktøy som ifølge den nye læreplanen skal benyttes i faget.

Ferdighetstreningen i programmering ble gjort ved at elevene skulle programmere Spheroballen til å gå gjennom en bane bestående av sammenlagte bord (figur 5). Elevene fikk et startpunkt og målet var å programmere ballen fra startpunktet til målskiven (figur 6). De kunne også velge å programmere fargeendringer på Spheroballen.



Figur 5: banen bestående av sammenlagte bord

Figur 6: målskive på banen

4.5 Gjennomføring

Datainnsamlingen foregikk i februar 2022 på en barneskole i Oslo. Lærerne, som skulle intervjues, fikk informasjon om studiens hensikt og at det ville foregå intervju av lærerne samt observasjon av klassen. Etter at de sa seg villig til å delta mottok de et mer detaljert informasjonsskriv og samtykkeskjema. Elever og foresatte ble informert om at det skulle gjennomføres en observasjon uten innhenting av personopplysninger.

Først observerte vi klasse 1, deretter klasse 2, 3 og 4 uken etter. Hver av klassene hadde en undervisningsøkt som varte i 60 minutter. I klasse 1 og 2 hadde læreren delt elevene inn i grupper på to til tre elever der hver gruppe hadde en iPad. I klasse 3 og 4 hadde læreren planlagt at elevene skulle ha hver sin iPad og elevene var ikke delt inn i grupper. Undervisningen ble gjennomført i en åpen sal slik at ikke stoler, bord og sekker skulle stå i veien for Spheroballene.

Vi startet med å presenterte oss for elevene som masterstudenter. Deretter plasserte vi oss ved en valgt gruppe og fulgte denne gruppen slik at vi kunne fokusere på samhandlingen mellom elevene i gruppen og hvilke strategier de benyttet seg av. Vi endte opp med å observere totalt ni grupper og fire enkeltelever.

Noen dager etter undervisningen intervjuet vi lærerne i et grupperom på skolen. Før lydopptaket startet informerte vi om intervjuets hensikt og at det skulle bli gjort et lydopptak, og samlet inn underskrevet samtykkeskjema. Lyden ble tatt opp på en diktafonapp der opptaket umiddelbart ble kryptert. Vi minnet om at læreren når som helst og uten begrunnelse hadde mulighet til å trekke seg. I tillegg informerte vi om at en av oss skulle intervjue og den andre skulle lytte og eventuelt legge til spørsmål ved behov.

Intervjuene startet med korte spørsmål om lærerne. Deretter ble det spurt om hvordan lærerne tilrettela for undervisningen. Videre beveget intervjuets fokus seg over på programmering, algoritmisk tenkning og utfordringer. Intervjuet av læreren som planla undervisningsøkten hadde et større fokus på planleggingen, mens læreren som underviste ble spurt mer om gjennomføringen av undervisningen.

Under intervjuet satt vi av god tid til refleksjon som ga mulighet for utfyllende svar. Dette viste seg å være en fordel ved flere av intervjuspørsmålene. Blant annet spørsmålet: "Stemmer det at du har brukt den algoritmiske tenkeren i planleggingen av undervisningen?" som ble stilt til planleggeren der læreren ble stille en stund og så ut til å tenke for deretter å komme på flere deler av den algoritmiske tenkeren som ble brukt. Det ble i tillegg stilt oppfølgingsspørsmål der noe var uklart og der vi ønsket utdypende svar. Da planleggeren ble spurt: "Opplever du noen utfordringer når du underviser i programmering i naturfag, eventuelt hvilke?" ble gruppesammensetning nevnt og da ønsket vi videre et utdypende svar så da ble dette spørsmålet lagt til: "Hva tenker du angående det du sier med gruppesammensetninger? Hva må du tenke på der?" På spørsmålet om hvilke strategier fra den algoritmiske tenkeren det var ment at elevene skulle bruke ble planleggeren usikker og så spørrende på oss. Da informerte vi om at vi hadde slått sammen arbeidsmåter og nøkkelbegreper og kalt det

strategier. Dette gjorde at informanten svarte på spørsmålet. På slutten av intervjuet fikk informantene mulighet til å legge til noe mer.

4.6 Analysemetode

4.6.1 Transkribering

Transkripsjon handler om å gjøre om det muntlige intervjuet til tekst. Hensikten er at datamaterialet, gjennom transkripsjonen, blir bedre strukturert noe som vil være fordelaktig i analysen (Kvale et al., 2015). Det ble tatt lydopptak under intervjuet slik at alle detaljer ble inkludert. Videre førte dette til at vi kunne være mer til stede i intervjuet, vise interesse og stille gode oppfølgingsspørsmål.

Transkripsjonen ble gjort manuelt ved bruk av programmet f4-Transkript. Lydfilen fra intervjuene ble lastet opp i programmet og en funksjon for å sette hastigheten ned gjorde det mulig å lytte samtidig som talen ble gjort om til tekst. Lydopptakene ble transkribert på bokmål og småord som "eh", "mhm" og "hmm" ble inkludert ettersom det er ord som kan gi uttrykk for at informanten må tenke seg om. I tillegg har lange pauser i informantens svar blitt markert med tre prikker og der vi i resultatene presenterer deler av et svar har vi satt tre prikker inni en parentes der svaret fortsetter, slik (…).

Observasjonsnotatene ble renskrevet rett etter hver observasjon. Dette er det stor enighet om blant forskere at er hensiktsmessig ettersom vi mennesker glemmer fort (Klette, 2003; Mulhall, 2003; Postholm, 2005). Under renskrivingen ble skrivefeil rettet opp og forkortelser skrevet ut. Videre fylte vi inn korrekt tegnsetting på sitatene og sørget for at alle sitatene hadde et tilhørende elevnummer. I tillegg skrev vi på klassenummer fra 1-4, gruppenummer fra 1-9 og elevene som arbeidet individuelt fikk betegnelsen A1-A4. Alt dette bidro til at notatene ble mer oversiktlige og dermed lettere å lese. Videre leste vi igjennom hverandres observasjonsnotater for å søke etter eventuelle uklarheter og tolkninger. Tolkninger ble markert med gul uthevningsfarge slik at de ikke skulle forveksles med de faktiske observasjonene. Vi lot imidlertid tolkningene stå i dokumentet ettersom det var ting vi ønsket å diskutere senere.

4.6.2 Analyse

Vi har benyttet oss av to ulike analysemetoder. Den første analysemetoden er en meningskondensering. Dette er en metode der informantenes uttalelser forkortes, samtidig som meningen blir bevart (Kvale et al., 2015). Denne analysemetoden ble brukt på intervjudataene for å korte ned lange svar og komprimere datamengden. Den andre analysemetoden som ble brukt både på de meningskondenserte intervjudataene og observasjonsdataene er en kvalitativ innholdsanalyse inspirert av Hsieh og Shannon (2005) *directed content analysis.* I denne analysemetoden tas det utgangspunkt i eksisterende teori for å lage kategorier (Hsieh & Shannon, 2005). I vår analyse lagde vi imidlertid i tillegg noen nye kategorier underveis. Derfor kan analysemetoden ses på som abduktiv, slik Kvale et al. (2015) beskriver det.

I vår studie har vi lagd forhåndsbestemte kategorier basert på den algoritmiske tenkeren. Disse deduktive kategoriene er nøkkelbegrepene og arbeidsmåtene vi finner i den algoritmiske tenkeren. Til sammen ble det 11 kategorier som vi har valgt å se på som 11 ulike strategier innenfor algoritmisk tenkning. Før analysen begynte skrev vi ned alle kategoriene og operasjonaliserte hver og én. For eksempel ble kategorien "dekomposisjon" beskrevet som å 1) dele opp problemet i mindre deler, 2) løse delproblemene hver for seg og 3) sette delene sammen for å gi en løsning på det overordnede problemet. Denne operasjonaliseringen betyr at dersom en elev gjennomførte en av de tre operasjonene ble dataen kategorisert under "dekomposisjon". Kategoriene med tilhørende operasjonalisering og eksempler på tilhørende data vises i tabell 1.

Tabell 1: Deduktive kategorier fra den algoritmiske tenkeren med operasjonalisering og eksempler på tilhørende data.

Kategori	**Operasjonalisering**	**Eksempel på tilhørende data**
Logikk	Forklare hvorfor noe skjer eller kommer til å skje på bakgrunn av en forutsigbar respons.	*Elev: "Nå har vi fått den til å gå rett frem, men banen går jo ikke rett frem så da må vi få til en sving, Siden svingene er så skarpe må vi ha en 90 grader."*
Algoritmer	En sekvens av instruksjoner eller et sett regler for å få noe gjort. Disse sekvensene skal danne en fremgangsmåte for å løse et problem.	Elevene skulle danne en fremgangsmåte for å få Spheroballen fra start til mål.
Dekomposisjon	Dele opp problemet i mindre deler, løse delproblemene hver for seg og sette delproblemene sammen for å gi en løsning på det overordnende problemet.	*Elev: "Alle bordene er jo like lange så vi kan bare bruke disse to bordene til å lage alle delene så får vi liksom ballen til å gå hele banen til slutt."*
Mønstre	Se etter likheter og ulikheter for å videre danne regler og løse generelle problemer.	Elev: "H*vis vi uansett finner riktig speed og tid til første rett fram og sving så kan vi jo bare bruke samme kode på de andre strekningene liksom.*"
Abstraksjon	Forenkle gjennom å hente ut den informasjonen som er relevant, og samtidig kvitte seg med irrelevant informasjon.	Algoritmen elevene lagde var en forenkling av den virkelige banen.
Evaluering	Vurdere hva, hvordan og hvorfor en har programmert. Vurdere kvaliteten og effektiviteten på løsningen opp mot formålet.	*Elev: "NEI, den var altfor kort! Da må jeg bare gjøre at den går litt lenger eller starte den litt lenger bort."*

Fikle	Løse problemer gjennom direkte utforsking og eksperimentering. Prøve seg frem og teste løsninger.	*Elev: "Jeg vet ikke helt hva vi skal, vi må kanskje forsinke den? Jeg tror vi bare må teste."*
Skape	Forbedre allerede eksisterende produkter eller skape noe nytt.	Elevene skulle lage en algoritme fra bunnen av.
Feilsøke	Identifisere og deretter rette opp feil gjennom systematisk testing og modifisering.	*"Den stopper for tidlig. Hvor mye har vi hastigheten på da? Vi må vel øke?"*
Holde ut	Fortsette og prøve å finne en løsning, selv når enn møter på utfordringer.	*Elevene prøvde en gang til med samme kode og så at de kom samme sted. De prøver på nytt og en elev sier: "Lengden er riktig, men den må litt lenger til venstre."*
Samarbeide	Vurderer ulike løsninger og diskutere seg frem til en hensiktsmessig løsning.	*Elev 2: "Den stopper for tidlig. Hvor mye har vi hastigheten på da? Vi må vel øke?"* *Elev 1: "Sant, vi prøver å øke med en fart til 4,2."*

Vi startet med å analysere intervjutranskripsjonene ved å ta for oss ett og ett av informantenes svar og satt inn den kondenserte meningen i en tabell (tabell 2). Videre la vi til en kolonne i analysetabellen som vi kalte *kategori*. Vi tok med oss de deduktive kategoriene fra den algoritmiske tenkeren og fylte inn kategoriene i raden til den meningen som samsvarte med kategoriens operasjonalisering. Videre fant vi data som ikke passet inn under de allerede eksisterende kategoriene. Derfor lagde vi nye induktive kategorier og fylte de inn i tabellen. Disse kategoriene ble lagd ut ifra vår data og er: *utfordringer rundt kommunikasjon om den algoritmiske tenkeren, utfordring knyttet til behov for veiledning, utfordring med samarbeid og teknologiske utfordringer*.

Tabell 2: analyse av intervjutranskripsjonene gjort gjennom meningskondensering

Informantens svar	Kondensert mening	Kategori
"Dette med å stå i et problem, utholdenheten til elevene. For vi ser ofte at elever blir fort utålmodige. Det har også vært en liten utfordring når vi har jobbet med programmering at de gir litt fort opp hvis de ikke klarer det. Da må de lære at det handler om feilsøking, hvor er det feilen skjer? Hvor kan feilen ligge? Og prøve å motivere de til å holde ut ved å si at de ikke kommer til å klare det på første forsøk så det er viktig dette med utholdenheten. Dette hadde jeg fokus på i instruksjonsøkten."	Elevene bør trenes i feilsøking for å styrke utholdenheten deres.	Feilsøking og holde ut

Analysen av observasjonsnotatene begynte med at vi undersøkte om noe av den innsamlede dataen fra observasjonen kunne knyttes til de fire induktive kategoriene. Videre fylte vi dataen inn i tabellen. Deretter markerte vi alt vi mente var aktuelt for kategoriene fra den algoritmiske tenkeren. Vi diskuterte under hvilke av de 11 kategoriene dataen tilhørte, og fylte dataen inn i en tabell som bestod av kategoriene fra den algoritmiske tenkeren. Deretter gikk vi igjennom dataene igjen for å forsikre oss om at vi ikke hadde oversett viktig data.

Underveis i analysen oppdaget vi at noen av kategoriene fra den algoritmiske tenkeren var overordnet for hele undervisningsøkten. Først og fremst lå kategorien *skape* iboende i undervisningen ettersom elevene fikk i oppdrag å lage en kode for å få Spheroballen fra start til mål. Videre så vi på *algoritme* som en annen overordnet kategori. Dette var fordi elevene satte sammen en sekvens av koder for å danne algoritmen som skulle få Spheroballen til å bevege seg på ønskelig måte. Til slutt kom *abstraksjon* frem gjennom at elevenes kode var en type forenklet representasjonsform av den virkelige banen. De tre kategoriene lå altså som en forutsetning for å løse oppgaven, og elevene benyttet seg av strategiene gjennom hele undervisningen. Vi har derfor valgt å ikke fylle inn data under hver av disse kategoriene, og heller fokusere på hvilke andre strategier elevene benyttet seg av for å klare å skape algoritmen i undervisningen.

Videre undersøkte vi hvilke kategorier og data som overlappet på intervjudataene og observasjonsdataene, og fylte dataene inn i en felles tabell. Dermed fikk vi oversikt over alle dataene og hvilke kategorier som belyste hvilke forskningsspørsmål. Til slutt gikk vi igjennom transkripsjonene og observasjonsdataene en gang til for å undersøke om vi hadde oversett noen aktuelle eller interessante data.

4.7 Reliabilitet

Reliabilitet handler om hvor sannsynlig det er å oppnå samme resultat dersom en følger samme fremgangsmåte (Cohen et al., 2018; Kvale et al., 2015; Olsson et al., 2003). Høy overensstemmelse mellom resultater tyder altså på høy reliabilitet. I kvalitative studier vil imidlertid svarene påvirkes av situasjonen (Cohen et al., 2018; Kvale et al., 2015), der blant annet elevenes tidligere erfaringer og klassens sammensetning kan spille inn. For å gjøre studien så etterprøvbar som mulig har vi redegjort for forskningsdesign, utvalg, gjennomføring og analysemetode.

Ett av tiltakene som styrker oppgavens reliabilitet er utforming av intervjuguide. Spørsmålene er utledet av forskningsspørsmålene, og kategorisert etter tema. En slik intervjuguide bidrar til at forskningen kan gjentas av andre på et senere tidspunkt (Kvale et al., 2015). Videre har vi sørget for å unngå ledende spørsmål. Ifølge Cohen et al. (2018) kan ledende spørsmål bidra til å frembringe ønsket svar, og dermed svekke reliabiliteten. For å unngå dette har vi stilt åpne spørsmål med vekt på informantenes egne meninger og erfaringer.

Under transkriberingen ble programmet f4-Transkript benyttet for å senke farten slik at vi fikk med oss alt som ble sagt. Etter at intervjuene var transkribert lyttet vi til opptaket flere ganger for å rette opp i feil. Dette vil styrke reliabiliteten.

En måte å styrke observasjons relabilitet på er å utarbeide et observasjonsskjema (Cohen et al., 2018). Vårt observasjonsskjema bestod av kolonner for observasjoner og tolkninger.

Kolonnene for observasjon og tolkning bidrar til at observasjonene ikke blir påvirket av forskerens egne tanker og fortolkninger underveis (Cohen et al., 2018; Skilbrei, 2019). Ved at vi tok notater underveis og renskrev notatene rett etterpå førte det til at observasjonen ga et mer autentisk bilde av situasjonen.

4.8 Validitet

Validitet handler om hvor gyldige tolkningene av den innsamlede dataen er (Cohen et al., 2018; Kvale et al., 2015). Videre deles validitet ofte inn i ekstern og intern validitet. Ekstern validitet referer til i hvilken grad studien kan generaliseres. Ved kvalitative studier er hensikten som regel ikke å generalisere. Likevel skal forskeren legge frem tilstrekkelig data slik at leseren selv kan avgjøre om forskningen er overførbar til en annen situasjon (Cohen et al., 2018; Lincoln & Guba, 1986). Dette har vi forsøkt å gjøre ved å blant annet beskrive utvalget og studiens kontekst.

Intern validitet går ut på om dataene som er samlet er relevante for forskningens hensikt (Cohen et al., 2018). Det at vi har benyttet oss av kriteriebasert utvalg vil kunne styrke validiteten ettersom informantene oppfyller krav knyttet til forskningens hensikt. I tillegg har vi benyttet oss av både intervju og observasjon for å belyse forskningsspørsmålene. Dette vil være en form for metodetriangulering som bidrar til å styrke den interne validiteten ettersom det er flere metoder som støtter opp under tolkningen av datamaterialet (Yin, 2018).

En utfordring med intern validitet i kvalitative studier er forskerbias (Johnson & Christensen, 2019). Forskerbias oppstår når datamaterialet påvirkes av forskerens førforståelse som innebærer egne synspunkter, perspektiver og eventuelle fordommer (Dalland, 2012; Johnson & Christensen, 2019; Kvale et al., 2015). For å unngå dette har vi i forkant reflektert rundt egne synspunkter og hvordan det eventuelt kan påvirke studien. Fordi vi ikke hadde så mye kunnskap om programmering og algoritmisk tenkning i forkant av studien, så vi for oss at det kunne oppstå mange utfordringer i undervisningen. Den manglende førforståelsen kunne potensielt gi et negativt bias i datainnsamlingen. Gjennom å reflektere rundt dette og bli bevisst egen førforståelse ble vi bedre i stand til å stille oss nøytrale i datainnsamlingen. I

tillegg har vi operasjonalisert begrepene som brukes som kategorier i analysen av dataene. Operasjonaliseringen av begrepene er basert på teori, noe som bidrar til at kategoriene i analysen er tilknyttet teorien som studien bygger på. Dette taler for en styrket begrepsvaliditet (Eriksen & Svanes, 2021).

4.9 Forskningsetikk

I et forskningsprosjekt skal forskeren ta hensyn til etiske retningslinjer som gjelder for forskningen. Vår studie er gjort i tråd med Norsk senter for forskningsdata (NSD) sine retningslinjer. Før vi dro ut og samlet data på skolen meldte vi studien inn til NSD, og fikk godkjenning. Videre ga vi læreren et samtykkeskjema, der det stod kort om hva som var formålet med studien, kontaktpersoner, hva det innebærer og delta, personvern og at deltakelse er frivillig. Dermed kunne informantene gi informert samtykke. Deretter informerte læreren elevene og de foresatte om at de skulle bli observert i en undervisningsøkt i naturfag om programmering. De fikk også informasjon om at det ikke skulle bli samlet inn persondata under datainnsamlingen.

5 Resultat

I denne studien undersøker vi hvordan elever på mellomtrinnet arbeider med programmering i naturfagundervisning og hvilke utfordringer elever og lærere møter på. Dette er videre konkretisert gjennom forskningsspørsmålene:

- Hvilke kjennetegn på algoritmisk tenkning kommer til utrykk i elevers arbeid med programmering i naturfag på mellomtrinnet?
- Hvilke utfordringer kommer frem i et undervisningsopplegg med programmering i naturfag på mellomtrinnet?

I dette kapittelet vil resultatene fra analysen bli presentert etter forskningsspørsmålene, og under hvert forskningsspørsmål har vi sortert resultatene etter de ulike kategoriene fra analysen. Under hver kategori vil vi starte med å presentere operasjonaliseringen av kategorien. Deretter vil vi presentere tilhørende funn fra observasjonene og intervjuene.

5.1 Hvilke kjennetegn på algoritmisk tenkning kommer til utrykk i elevers arbeid med programmering i naturfag på mellomtrinnet?

5.1.1 Fikle

Fikling handler om å løse problemer gjennom direkte utforsking og eksperimentering. Dette vises når elevene prøver seg frem og tester løsninger.

I begynnelsen av undervisningen var det flere grupper som var usikre på hvordan de skulle angripe utfordringen. Det kan virke som at elevene var usikre på hvilket resultat de ulike kommandoene i koden ga. Måten gruppe 7 startet med å teste ulike løsninger gir imidlertid inntrykk av at de ikke var redde for å gjøre feil, noe som gjorde at de kom godt i gang med kodingen:

> *Elev 1: "Jeg vet ikke helt hva vi skal, vi må kanskje forsinke den? Jeg tror vi bare må teste."*
>
> *Elevene tester.*
>
> *Elev 2: "Ja, det KUNNE gått..."*
>
> *Elev 1: "Ja, vi må bare ta ett sekund til, okei vi sjekker."*
>
> *Ballen går utenfor banen.*
>
> *Elev 2: "Hahha okei det funka ikke!"*
>
> *Elev 1: "Vi må ha forsinkelse imellom."*

5.1.2 Feilsøke

Feilsøking handler om å identifisere og deretter å rette opp feil. For å få til dette vil systematisk testing og modifisering være nødvendig. Planleggeren hadde lagt vekt på feilsøking i introduksjonsøkten til denne programmeringsundervisningen. Dette fordi planleggeren mente at elevene trengte å fokusere på hvor feilen skjer og hvordan de kan løse feilen for å holde ut. I tillegg måtte elevene bevisstgjøres at de måtte gi det flere forsøk for å

komme frem til en løsning slik at de ikke skulle gi opp, noe som hadde vært en utfordring da elevene hadde arbeidet med programmering tidligere.

Vi observerte at flere av elevgruppene benyttet seg av feilsøking da de oppdaget at koden ikke fungerte som ønsket og dette kan skyldes at det har blitt lagt vekt på feilsøking i introduksjonsøkten. Elevene feilsøkte da Spheroballene gikk for langt eller kort på banen eller da den gikk feil vei. Etter hvert som slike utfordringer oppstod var det flere av gruppene som startet med å konstatere hva som var galt. Deretter foreslo de løsninger for å rette opp i feilen:

> Gruppe 5
>
> *Elev 2: "Den stopper for tidlig. Hvor mye har vi hastigheten på da? Vi må vel øke?"*
>
> *Elev 1: "Sant, vi prøver å øke med en fart til 4,2."*
>
> *Elev 2: "Nå gikk den så vidt over så vi prøver med 4."*
>
> *Gruppen endrer gradvis ned til de ender på 3,4 sekunder.*

Å komme med et slikt forslag krever at elevene kjenner til koden og hvilke kommandoer som gir hvilket resultat. Videre brukte gruppe 5 kunnskapen til å endre tallet inne på kommandoen som styrte farten. Ettersom de modifiserte koden ved å endre farten litt og litt, viser dette til systematisk testing, som videre fører til en kontrollert og gjennomtenkt endring.

En av elevene som arbeidet individuelt (A2) oppdaget underveis at Spheroballen ikke trillet langt nok. For å undersøke hvor i koden feilen lå startet eleven med å telle seg frem til hvilken plate ballen hadde stoppet på. Deretter så elevene ned på iPaden og begynte å telle seg frem til samme sted i koden. Det kan virke som om eleven hadde lagd en seksjon med koder per bord. Dette førte til at eleven kunne gå direkte inn i riktig kode for å endre hastigheten der ballen hadde trillet for kort. Dette kan tolkes som en form for feilsøking selv om eleven ikke benyttet seg av systematisk testing ettersom det fortsatt gjøres en modifisering i koden. Det at eleven

hadde delt opp den store koden i mindre deler førte til at det kunne være lettere å både oppdage og rette feilen.

Elevene i gruppe 8 var blant en av de gruppene som benyttet seg av mindre effektive feilsøkingsstrategier da gruppen oppdaget at ballen svingte slik at den havnet utenfor banen. I stedet for å endre fart eller vinkel, slik som andre grupper gjorde, bestemte de seg for å prøve med andre blokker, kalt ulike farger av elevene:

> *Elev 1: "Ja, men skal vi prøve med en helt annen da? Hva med en rosa i stedet eller noe sånn? Vi kan jo bare prøve å se."*
>
> *Elev 2: "Men den rosa har jo ikke noe å si for svingen. Vi skal få ballen til å svinge, da må vi jo heller velge noe som får den til å svinge."*
>
> *Elev 1: "Nei, men hvis vi bare prøver og ser sa jeg jo."*

Deretter prøvde gruppen seg frem med ulike koder helt til gruppe 6 oppdaget at de ikke klarte å rette opp i feilen og dermed prøvde å hjelpe dem:

> *Elev 1: "Dere må endre gradene på den blå koden også må dere få den til å gå i en større eller mindre fart for at den skal gå kortere eller lengre."*

Det at gruppe 6 bryter inn kan skyldes at de ser at gruppe 8 bruker mye tid på å finne en løsning, uten at de kommer frem til en løsning.

5.1.3 Holde ut

Holde ut handler om å ikke gi opp og fortsette å prøve, selv når en møter utfordringer. Flere av gruppene viste utholdenhet ved at de ikke ga opp da de gjorde feil, men gjorde justeringer i koden for å rette opp feilen. Dette viste gruppe 5 ved at de gjorde flere målrettede justeringer rettet mot endring av tid for å få Spheroballen til å endre lengde:

De tester gjennom hele banen på nytt og ser at ballen kommer litt for langt på slutten slik at den går utenfor bordet på slutten. Da sier elev 2: "Vi må ha litt færre sekunder."

Da kommer de i mål og treffer blinken på 25. Da vil de teste på nytt for å prøve å treffe akkurat på 100.

De prøvde en gang til med samme kode og så at de kom samme sted.

De prøver på nytt og elev 1 sier: "Lengden er riktig, men den må litt lenger til venstre."

Et annet eksempel på en gruppe som viste utholdenhet var gruppe 6 som oppdaget at ballen gikk for kort flere steder og rettet opp i dette. Deretter prøvde de på hovedbanen og landet på 100. Selv om denne gruppen klarte 100 poeng, ville de teste på nytt. Da de testet andre gang oppdaget de at ballen ikke landet samme sted som sist, noe som mest sannsynlig skyldtes at bordene hadde blitt tråkket på av andre elever og dermed forskjøvet seg. I stedet for å gi opp eller si seg ferdig fordi de allerede hadde klart å nå målet om 100 poeng ønsket de å fortsette for å klare det på nytt, og de gjorde små justeringer for å få ballen til å lande på 100 igjen.

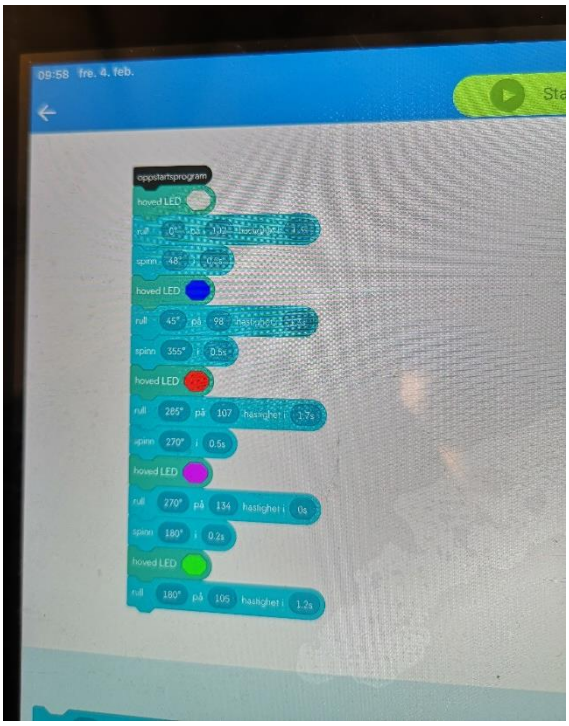
5.1.4 Evaluering

Evaluering går ut på å se på hva, hvordan og hvorfor elevene har programmert. I denne sammenheng skal elevene vurdere kvaliteten og effektiviteten på løsningen opp mot formålet. Dette skal elevene gjøre underveis og til slutt. Underveis så vi at elevene vurderte hvilke koder de måtte bruke for å få Spheroballen til å følge banen. De gangene Spheroballen ikke fulgte banen diskuterte de hvorfor den ikke gikk slik den skulle, og hvordan de kunne gjøre endringer. Dette ser vi blant annet i avsnittet om feilsøking der elevene identifiserte og rettet opp i feil. I avsnittet om å holde ut gir gruppe 5 inntrykk av at de vil øke kvaliteten på løsningen fordi de ikke gir seg på 25, men vil få ballen til å treffe 100 på målskiven.

Vi observerte at gruppe 3 gjorde vurderinger rundt effektiviteten av løsningen:

> *Elev 1: "Vi kan jo bare ta den på skrå så den slipper å gå rundt hele svingen for da går den jo raskere. Da tar vi bare litt mer enn 45 grader."*

Denne gruppen så at det var mulig å lage en kode slik at Spheroballen bevegde seg på skrå fremfor å sette inn en kode for at Spheroballen skulle gå rett frem og deretter sette inn en ny kode for at Spheroballen skulle ta en sving. Gruppe 3 viste at de vurderte hvilke grader som ga den riktige vinkelen for å få Spheroballen til å gå på skrått over to bord. De skrev 48 grader i koden (figur 7).



Figur 7: kode som får Spheroballen til å gå på skrå.

Elev A3 møtte på en utfordring der Spheroballen kjørte for kort og eleven kom frem til to mulige løsninger som kunne bidra til at ballen gikk hele veien til mål:

> *"NEI, den var altfor kort! Da må jeg bare gjøre at den går litt lenger eller starte den litt lenger bort."*

Eleven måtte dermed vurdere hvilke av de to løsningene eleven skulle benytte.

5.1.5 Samarbeid

Hensikten med å samarbeide er at elevene skal lære av andres kunnskaper og erfaringer gjennom samtaler og diskusjoner rundt utfordringene de møter på. I denne kategorien har vi plassert data fra situasjoner der grupper vurderer ulike løsninger og diskuterer seg frem til en hensiktsmessig løsning. Dette innebærer å lytte til hverandre, inngå kompromisser og være villig til å teste andres løsninger. I tillegg handler det om å støtte medelever.

I de to klassene som skulle arbeide individuelt observerte vi at elevene ikke kom like raskt frem til en løsning og dermed kan det tolkes som at effektiviteten var dårligere for de gruppene som arbeidet individuelt. I tillegg viste det seg at flere av elevene som arbeidet individuelt kom kortere i koden enn de elevene som samarbeidet. Dette så vi ved at det var flere av Spheroballene i samarbeidsgruppene som nådde målet om å fullføre banen, enn i de to klassene der elevene arbeidet individuelt.

I intervjuene kommer det frem at lærerne har tilrettelagt for samarbeid. Tilrettelegging for samarbeid handler om lærerens målrettede påvirkning av samarbeidet. Det kan både inkludere forhåndsbestemte påvirkninger, men også tilpasninger som blir gjort underveis.

Lærerens forhåndsbestemte påvirkninger på samarbeidet ble gjort gjennom å velge gruppestørrelse og sammensetninger av elever. For å tilrettelegge slik at alle elevene på gruppen skulle bidra hadde læreren satt opp grupper med to elever fordi:

> *"Jo flere de er jo kjipere er det jo egentlig for da er det så mye lettere at det er en person som ikke får noe med seg."*

Dette påpeker også planleggeren som mener at grupper på to til tre elever vil fungere best for at alle på gruppa skal bidra. Læreren la videre vekt på at gruppesammensetningen måtte endres fordi flere elever hadde vært borte i forberedelsesøktene de hadde hatt:

> *"(…) noen vet ikke hva Sphero er i det hele tatt, så hver gang vi har gjort det så har det vært første gangen for noen. Hvis det for eksempel er to elever som jeg vet sliter*

eller som jeg er litt redd at kanskje ikke lærer så raskt, at de på en måte har en som de kan lære av da, som har vært i undervisningen før."

Dette viser at læreren har tilpasset samarbeidet gjennom å legge vekt på at en av elevene skal være kjent med programmering av Spheroball fra før slik at denne eleven kan lære den andre på gruppen.

Det ble observert at læreren tilpasset samarbeidet underveis i undervisningen ved å gi beskjed til elevene om at de skulle bytte på hvem som holdt iPaden. Grunnen til dette kom frem under intervjuet:

"Da de var halvveis så sa jeg at de skulle bytte sånn at den andre kan legge inn koden også, så begge to husker det bedre og faktisk har gjort operasjonen ikke bare sett på det."

Under observasjonen viste det seg at flere grupper ikke byttet på hvem som holdt iPaden. I gruppe 7 var det eleven som holdt iPaden som kodet. Den andre eleven trykket aldri på iPaden eller brukte noen begreper innenfor programmering. Eleven fungerte som tilskuer og bidro imidlertid med positive rop og kroppsspråk som klapping og hopping da koden fungerte som ønsket. Det kan virke som om dette bidro til å motivere eleven som programmerte ettersom de jublet og smilte sammen underveis. Samtidig kan det diskuteres i hvor stor grad eleven som heiet lærte noe om hvordan de ulike kommandoene i koden fungerte.

Gruppe 5 bestod av elev 1 som hadde erfaring med programmering og elev 2 som ikke hadde vært med i introduksjonsøkten. Elevene i gruppe 5 byttet ikke iPad underveis, men hadde imidlertid en dialog om programmeringen:

Elev 1: "Vi må tenke på å stille gradene, vet du hvorfor vi må det?"

Elev 2: "Nei, jeg er litt usikker."

Elev 1: "Det er for at den skal svinge."

Elevene prøver på banen.

Elev 2: "Den stopper for tidlig. Hvor mye har vi hastigheten på da? Vi må vel øke?"

Elev 1: "Sant, vi prøver å øke med en fart til 4,2."

Elev 2: "Nå gikk den så vidt over så vi prøver med 4."

Elevene viser en form for samarbeid som åpner for å lytte til hverandres forslag og forklare uklarheter. Elev 2 sørget for at medeleven var deltakende ved å forklare hva som ble gjort og stille spørsmål underveis. Det kan virke som om elevene hadde funnet en måte å samarbeide på som begge virket å være tilfreds med. Dette understrekes av elevenes ønske om å ikke bytte på hvem som holdt iPaden underveis, da elev 2 sa til elev 1:

"Du er best til å styre iPaden og jeg liker å se meg rundt og si ting jeg tenker på."

5.1.6 Logikk

Logisk resonnement handler om å kunne forklare hvorfor noe skjer eller kommer til å skje på bakgrunn av en forutsigbar respons. Gruppe 2 testet koden de hadde lagd for så å gjøre et logisk resonnement for veien videre:

Gruppe 2: "Nå har vi fått den til å gå rett frem, men banen går jo ikke rett frem så da må vi få til en sving, Siden svingene er så skarpe må vi ha en 90 grader."

Litt senere skulle de få ballen til å svinge motsatt vei:

Elev 1: "Ballen svingte til høyre i stad, og da var den jo på 90 (grader)."

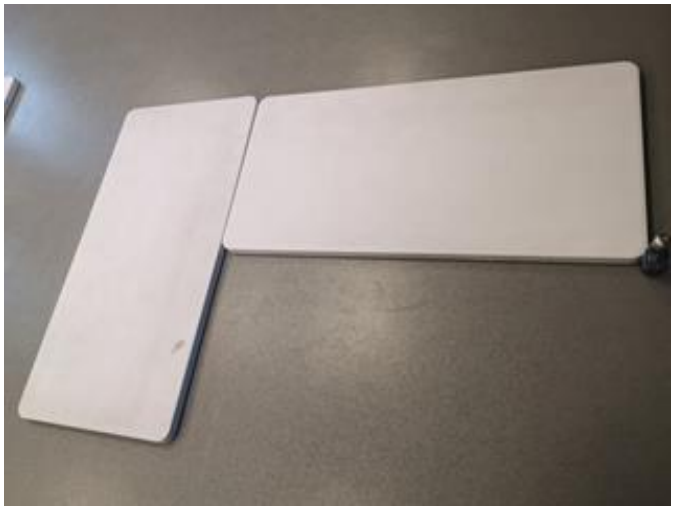
Elev 2: "Da må vi stille den sånn (ca. 270 grader) sånn at den svinger til venstre."

Elevene analyserer resultatet fra forrige sving ved å vurdere hvilken del av koden som fikk Spheroballen til å svinge ønsket vei. De viser forståelse for at det er antall grader som må endres for at Spheroballen skal svinge i en ny retning. Det kan virke som om elevene bruker kunnskaper fra matematikken om gradene i en sirkel for å sette inn rett tall. Dermed kunne de forutse hvilken vei Spheroballen ville trille da de satte inn 270 grader.

5.1.7 Dekomposisjon

Dekomposisjon går ut på å dele opp problemet i mindre deler slik at problemet blir mer håndterlig. Det kan innebære å kode en og en del for så å sette delene sammen til en lang kode, eller ta for seg en del, kode den ferdig for så å sette den inn i hovedkoden.

I begynnelsen av undervisningen sa læreren, i de to klassene som samarbeidet, at elevene kunne ta to bord hver for å teste koden de lagde før de skulle prøve på hovedbanen som de fikk to forsøk på. Dette kan fungere som en hjelp for elevene slik at de kan ha fokus på én del av banen før de tar for seg hele. Figur 8 viser bordene elevene brukte for å dele opp og teste deler av kodene sine.



Figur 8: Bordene elevene brukte til å dele opp og teste deler av kodene sine.

Flere av elevene brøt ned problemet ved å dele opp banen i mindre deler. Deretter la elevene til flere koder som endte i en fullstendig kode. Dette viser gruppe 4:

> *Elev 1: "Alle bordene er jo like lange så vi kan bare bruke disse to bordene til å lage alle delene så får vi liksom ballen til å gå hele banen til slutt."*

Noen andre grupper delte også opp banen ved å bruke to bord, men de flyttet bordene etter hvor langt de var kommet i koden slik at bordene samsvarte med hovedbanen svinger. I motsetning til gruppe 4 kan det virke som at elevene ikke klarte å visualisere banen i hodet og derfor måtte flytte bordene etter hvert som de utarbeidet koden.

Gruppe 6 laget en kode til de første to bordene. Deretter flyttet de bordene etter slik hovedbanen så ut og laget en ny kode. Slik fortsatte de til de hadde fire koder. Til slutt satt de sammen de fire kodene til en fullstendig kode. Da de gjennomførte koden på den hovedbanen oppdaget de at ballen gikk for kort flere steder. Dette kan skyldes at kodene ikke hang sammen ettersom kodene var programmert hver for seg, og at elevene ikke hadde tatt høyde for at de måtte legge til ekstra lengde fordi de ikke startet på samme sted som de avsluttet.

Læreren, i de to klassene som arbeidet individuelt, informerte ikke om at de kunne bruke andre bord for å dele opp banen. I disse klassene så det ut til at elevene var usikre på hvordan de skulle starte. Noen av elevene startet med å endre farge på Spheroballen og gikk sammen om å diskutere fargevalg. Disse elevene klarte ikke å lage en ferdig utviklet kode. Dette viser at elevene ikke gikk rett på å programmere Spheroballen etter banens form, men det ser ut til at de utforsket blokker. Elev A4 gikk banen del for del i tillegg til å bruke armene til å vise retning. Underveis la eleven gradvis til flere koder. Dette kan tolkes som at elev A4 har funnet en måte å dekomponere løsningen for å se sammenhengen mellom koden og banen, og dermed gjøre programmeringen mer håndterlig.

5.1.8 Mønstre

Det å finne mønstre i programmering handler om å se etter både likheter og ulikheter for å videre danne regler og løse generelle problemer. Dette viser flere grupper gjennom at de har forstått at dersom de vet hvordan ballen svinger og går rett frem kan de bruke dette til å fullføre banen ved å repetere koder. Dette viser gruppe 9 hvor elev 1 forklarer at de tenkte at hvis de finner riktig hastighet og tid til første bord også ut første sving kan de bruke samme kode på de andre strekningene også. Elev 2 legger til:

> *"Men med annen vinkel på svingen da liksom fordi den skal jo svinge en annen vei."*

5.2 Hvilke utfordringer kommer frem i et undervisningsopplegg med programmering i naturfag på mellomtrinnet?

Funnene som er knyttet til dette forskningsspørsmålet er kategorisert etter følgende fire kategorier:

1. Kommunikasjon rundt den algoritmiske tenkeren
2. Utfordringer knyttet til behov for veiledning
3. Utfordringer med samarbeid
4. Teknologiske utfordringer

5.2.1 Kommunikasjon rundt den algoritmiske tenkeren

Denne kategorien handler om hvordan den algoritmiske tenkeren har blitt kommunisert mellom planlegger og lærer, og lærer og elever.

Planleggeren fortalte at den algoritmiske tenkeren hadde stått sentralt i utviklingen av undervisningsopplegget, med vekt på arbeidsmåtene i modellen. Videre kom det frem at den algoritmiske tenkeren ikke hadde blitt kommunisert til lærerne da planleggeren introduserte dem for undervisningsopplegget. Dette kan gjøre det vanskelig for lærerne å vite at det skal vektlegges i undervisningen. Det ble observert at læreren ikke kommuniserte strategiene fra den algoritmiske tenkeren til elevene. Dette kan både skyldes den manglende kommunikasjonen mellom planleggeren og lærerne, men også lærerens forståelse av den algoritmiske tenkeren:

> *"Ja, for jeg har ikke hatt sånn bevisst forhold til, eh jeg vet ikke hva den algoritmiske tenkeren...jo jeg vet jo på en måte hva det er, men jeg har ikke satt meg så veldig inn i akkurat det begrepet...Jeg syntes det er vanskelig å svare på spørsmål rundt det begrepet. Jeg må jo bryte opp begrepet først og så sette det inn i en ny kontekst og nytt spørsmål så blir jeg sånn SHIT det er vanskelig."*

Under observasjonen kom det frem at gruppe 1 spurte læreren om hjelp da de ikke forstod koden de hadde kopiert fra en annen gruppe. Læreren svarte at de måtte spørre noen andre

grupper. Dette kan tolkes som at læreren ønsket at elevene skulle samarbeide og lære av hverandre, noe som støttes av at læreren trekker frem viktigheten av samarbeid i intervjuet. Likevel kan lærerens rolle i en slik situasjon være å bidra med veiledning av hvilke strategier elevene kan benytte seg av. Det så imidlertid ikke ut til at læreren veiledet elevene og dette kan skyldes lærerens faglige begrensinger:

> *"En utfordring er jo egne begrensinger da. Egentlig at det føles, jeg går alltid med en liten sånn sommerfugl i magen mer enn alle andre undervisningsøkter jeg har, for det er sånn jeg vet jo at jeg ikke kan nok. Når elevene stiller spørsmål så må vi jo ofte løse det sammen, og det funker jo helt fint det, men det er jo alltid en elev som fant ut av det fordi de har jo mer teknologisk intuisjon enn meg på en måte."*

Ettersom intuisjon handler om å ha en direkte forståelse eller fornemmelse av en situasjon kan det virke som at læreren mener at elevene i større grad har en følelse hvordan de skal løse et problem når teknologi er involvert. Dette kan komme av at elevene er vant til å bruke teknologi i hverdagen og har flere ulike teknologiske plattformer de benytter seg av, for eksempel PC, iPad og spillkonsoller.

Det ble observert at det var liten grad av kommunikasjon mellom lærer og elever i undervisningen. Læreren nevnte ingenting om strategier elevene kunne bruke verken i starten av undervisningen eller underveis i undervisningen. Vi så heller ikke noe oppsummerende avslutning der elevene delte deres fremgangsmåte.

5.2.2 Utfordringer knyttet til behov for veiledning

Det ble observert at flere elever synes det var utfordrende å komme frem til en løsning på egenhånd. Et eksempel på dette er elev A1 som spurte læreren om hjelp og fikk beskjed om å prøve seg frem eller spørre andre elever. Videre spurte elev A1 andre elever:

> *A1: "Hvordan starter man, hvordan styrer man?"*
>
> *Ser en annen elev som har fått ballen til å bytte farge*
>
> *A1: "Hva skal jeg trykke på for å få den til å gå sånn? Hva trykker jeg på for å bytte farge?"*

Dette kan tolkes som at eleven ikke har tilstrekkelige ferdigheter eller/og erfaring med å løse utfordringer i programmering og dermed fører det til en utfordring med å sette sammen koder på egenhånd.

I intervjuet ga læreren uttrykk for at det var et bevist valg at elevene prøve seg fram på egenhånd:

> *"Jeg føler veldig sånn elevstyrt. De er såpass drilla og gode på sånne type øvelser at jeg holder meg veldig i bakgrunnen egentlig."*

Vi så imidlertid at mange elever hadde utfordringer med å løse oppgaven og hadde behov for hjelp underveis. Som nevnt i delkapittelet over (5.2.1) valgte noen elever å spørre læreren og fikk til svar at de måtte spørre andre elever eller grupper. Flere grupper valgte å spørre andre grupper om løsningen på oppgaven fremfor å prøve seg frem på egenhånd. Da gruppe 1 ikke fikk ballen til å bevege seg kopierte det de andre grupper. Det så imidlertid ikke ut som at elevene utforsket løsningen de kopierte og dette kan være årsaken til at elevene ga utrykk for at de ikke forstod hva som hadde blitt gjort:

> *"Jeg forstår ikke? Hva gjorde vi annerledes nå?"*

Senere i undervisningen ga elevene utrykk for at de var usikre på hva de skulle gjøre videre. Dette kan skyldes at de ikke forstod hva koden de hadde kopiert innebar. Dette viste også gruppe 1 som kopierte andres løsninger og opplevde utfordringer videre i prosessen:

> *"Vi klarte jo svingen i stad, men jeg forstår ikke hvordan vi skal klare svingen nå? Hva gjorde vi i stad egentlig?"*

Dette ser ut til å skyldes at elevene ikke forstod hva de har gjort tidligere og kunne dermed ikke benytte tidligere løsninger videre i oppgaven. Det virket som at elevene ikke så likheter og ulikheter i svingene og kunne ikke benytte dette til å se mønstre i programmeringen. Videre vil dette kunne føre til at elevene er avhengig av andres hjelp til å finne en løsning, eller at de må sette seg inn i og utforske hvordan de ulike kodene fungerer.

5.2.3 Utfordring med samarbeid

I intervjuet kom det frem at både læreren og planleggeren så på samarbeid som grunnleggende i programmering. Begge mente at det å være mellom to til tre på gruppen kan gjøre at alle på gruppen bidrar. Som nevnt tidligere satt læreren en elev som hadde lite erfaring med programmering sammen med en elev som hadde erfaring med programmering for at den ene skulle lære den andre. I flere grupper virket det som at et slikt samarbeid fungerte godt, mens for andre grupper fungerte det ikke like godt. Dette viste seg i gruppe 1 der elev 2 ikke hadde vært til stedet i introduksjonsundervisningen:

> *Elev 1 holder iPaden.*
>
> *Elev 2: "Hva gjorde vi nå? Hvorfor gikk den den veien?"*
>
> *Elev 1 svarer ikke.*

Mangelen på kommunikasjon mellom elevene fører til at elev 2 ikke blir inkludert i programmeringen. Dermed faller lærerens hensikt, om at den ene eleven kan lære av den andre, bort. Ved mangel på forklaring fra medeleven sluttet elev 2 å stille spørsmål og brukte heller tiden på å se i luften eller bort på andre grupper. Dermed endte elev 2 opp med å være passiv i samarbeidet og det var kun elev 1 som aktivt programmerte.

I tillegg ble det observert flere grupper som avviste medelevenes løsningsforslag. Dette kan ha påvirket samarbeidet ettersom elevene ikke lyttet til hverandre og løste problemet sammen. Videre kan det diskuteres om elevens faglige utvikling påvirkes ettersom elevene ikke får mulighet til å teste teoriene sine. Dette kan føre til at elevene mister muligheten til å bli en del av den direkte utforskningen og kan bli, som elev 2 i eksempelet over, en ufrivillig passiv deltaker.

For å sørge for at begge elevene var aktive i programmeringen ga læreren beskjed om at de skulle bytte på hvem som styrte iPaden. Dette ble sagt, men ikke fulgt opp. Dette kan skyldes at læreren mente at elevene var selvstendige i programmeringen:

"Jeg føler veldig sånn elevstyrt. De er såpass drilla og gode på sånne type øvelser at jeg holder meg veldig i bakgrunnen egentlig."

Det var imidlertid flere grupper som ikke byttet på hvem som styrte iPaden. Dette kunne føre til at elevene som ikke fikk styre iPaden ikke fikk et innblikk i hvordan de ulike kommandoene fungerte og hvordan de ble satt sammen til en helhetlig kode.

5.2.4 Teknologiske utfordringer

Teknologiske utfordringer innebærer utfordringer knyttet til iPad, Spheroballene og internett. Planleggeren trakk frem at teknologi kan være en utfordring i programmeringsundervisning. Videre påpeker planleggeren at bruk av teknologi i undervisningen krever at lærere forbereder seg på de teknologiske utfordringene som kan oppstå.

I undervisningen ble det observert at Spheroballene ikke var oppladet og elevene måtte derfor ta en pause. Pausen så ut til å komme på et tidspunkt hvor elevene var engasjerte fordi elevene kom med utsagn som *"Vi har jo nesten klart det!"* og *"Åååå, vi skulle jo akkurat til å teste på banen. Da gidder jeg ikke mer."* For å unngå at elevene skulle gi opp foreslo læreren at elevene kunne dele seg opp å gå sammen med andre grupper som hadde en Spheroball med strøm. En elev responderte med å sette seg ned på gulvet og sukke, en annen så utover rommet og gjespet og en tredje elev svarte: *"Nei, det gidder jeg ikke."* Dette viser at noen elever gir opp når teknologien svikter.

6 Diskusjon

Hensikten med denne studien er å undersøke hvordan elever på mellomtrinnet arbeider med programmering i naturfagundervisning, og utfordringer som kan oppstå i planlegging og gjennomføring av slike undervisningsopplegg. I diskusjonskapittelet vil resultatene fra analysen drøftes, i lys av tidligere presentert teori og empiri. I tillegg vil vi drøfte hvordan funnene våre kan bidra til å utvikle programmeringsundervisning i naturfagklasserom. Vi har strukturert diskusjonen etter forskningsspørsmålene:

- Hvilke kjennetegn på algoritmisk tenkning kommer til utrykk i elevers arbeid med programmering i naturfag på mellomtrinnet?
- Hvilke utfordringer kommer frem i et undervisningsopplegg med programmering i naturfag på mellomtrinnet?

6.1 Kjennetegn på algoritmisk tenkning

Resultatene viser at elevene benytter seg av flere av strategiene som ligger innenfor den algoritmiske tenkeren. Tre av strategiene fra den algoritmiske tenkeren kom frem i arbeidet til alle elevene ettersom det var en forutsetning for undervisningen. Derfor er disse tre strategiene samlet under overskriften *overordnede strategier – skape, algoritme og abstraksjon* (6.1.1). I tillegg er strategiene dekomposisjon og mønstre slått sammen. De ytterligere strategiene er i hovedsak drøftet hver for seg, men noen strategier blir likevel nevnt i drøftingene til andre strategier der det er naturlig. Til slutt gjengir vi de viktigste poengene fra drøftingen knyttet til det første forskningsspørsmålet i en oppsummering før vi går videre til drøftingen knyttet til funnene innenfor det andre forskningsspørsmålet.

6.1.1 Overordnede strategier – skape, algoritme og abstraksjon

Skape blir beskrevet som en sentral del av programmeringen der elever enten kan forbedre et allerede eksisterende produkt eller skape noe nytt (Barefoot computing, u.å-e). Undervisningen vi observerte gikk ut på at elevene skulle lage en kode ut ifra blokkene som var tilgjengelig i programmet de programmerte i. Dermed kan det å skape ses på som en strategi som lå til grunn for hele undervisningen ved at elevene skapte en kode.

Hvor langt elevene kom i arbeidet med å skape en kode ble påvirket av hvilke andre strategier elevene brukte. På den ene siden så vi blant annet at flere av elevene som evnet å feilsøke og forutse utfall basert på logiske resonnementer klarte å lage en ferdigutviklet kode. I tillegg så vi en tendens til at elevene som samarbeidet kom lengre i programmeringen enn de elevene som arbeidet individuelt. Dette vil bli drøftet mer under delkapittel 6.1.4. På den andre siden så vi elever som kopierte deler av andre elevers kode uten at de så ut til å forstå hva koden betydde. Selv om disse elevene lagde deler av koden så vi at de ikke utviklet den ferdig. Betydningen av å kopiere andres løsning vil bli drøftet nærmere under delkapittel 6.1.6. Resultatene viser at kunnskap om strategiene innenfor den algoritmiske tenkeren kan bidra til at elevene mestrer å skape koden. Dette samsvarer med studien til Sentance og Csizmadia (2017) som understreker viktigheten av å utvikle elevenes algoritmiske tenkning ved å lære dem ulike strategier de kan bruke. Slike strategier finner vi i den algoritmiske tenkeren.

Algoritme er en annen strategi som kan ses på som overordnet for undervisningen i denne studien. I likhet med strategien *skape*, er *algoritme* en strategi som elevene arbeidet med parallelt med de andre strategiene. Gjennom at elevene blant annet brøt ned utfordringen, utforsket blokkene, identifiserte og rettet opp i feil og vurderte løsningen satt elevene sammen blokkene til en algoritme. Algoritmen instruerte hvordan Spheroballen skulle bevege seg. Introduksjonsøkten elevene hadde bidro til at elevene ble kjent med programmet de skulle programmere i og programmeringsspråket slik at det skulle bli lettere for dem å lage algoritmen. En slik opplæring i programmet elever skal bruke støttes av studien til Tyrén et al. (2018) som trekker frem elevers behov for å være kjent med programmet.

Algoritmen elevene lagde kan ses på som en abstraksjon av den opprinnelige banen. Ved å se bort ifra irrelevant informasjon lagde elevene en kode som var en forenklet versjon av banen. Dette samsvarer med både Barefoot computing (u.å-a) og Wing (2017) sin beskrivelse av abstraksjon. Kodene elevene lagde var forenklet i ulik grad og inneholdt dermed ulik mengde informasjon. Koden til gruppe 3, som programmerte Spheroballen til å gå på skrå fremfor å svinge i rette vinkler, hadde andre detaljer enn kodene til de andre gruppene. Dette gjør at

dersom en hadde sett kodene ved siden av hverandre uten å vite hvordan banen så ut, kan det være at det dannes to ulike bilder av den samme banens form.

Elevene i gruppe 1 som er trukket frem under *utfordringer knyttet til behov for veiledning* (5.2.2) i resultatene kopierte andres koder fremfor å lage kodene på egenhånd, og stod fast i neste del av programmeringen. Dette kan skyldes at de ikke har sett sammenhengen mellom abstraksjonen og den virkelige banen. På samme måte som elever kan forstå et fenomen i naturfag ved å forstå en modell av fenomenet, kan det å sette seg inn i en kode og forstå den utvikle elevenes forståelse for hvordan programmeringen foregår.

6.1.2 Feilsøking – identifisering og retting av feil

Flere elever viste evne til å oppdage feil ved at de testet koden og dersom Spheroballen ikke gikk som ønsket konstaterte de at det hadde oppstått en feil. Gruppe 5 gikk videre ved å definere hva feilen var for så å starte og rette opp i den. Dermed gikk de inn i koden for å undersøke hvilke justeringer som måtte gjøres for å oppnå ønsket resultat. Mellom hver justering kjørte de Spheroballen gjennom banen som en test.

Denne formen for feilsøking passer inn under Dong et al. (2019) sin definisjon av test-basert fikling. Gjennom å oppdage feil underveis ble elevene i gruppe 5 bedre kjent med forskjellen mellom ønsket utfall og faktisk utfall. Dermed ble de nødt til å gjøre endringer i koden for å oppnå det ønskede utfallet. Elevenes evne til å gjøre små justeringer i koden viste at de benyttet seg av systematisk testing i feilsøkingen, som ifølge Barefoot computing (u.å-f) er en viktig del av feilsøking. I tillegg feilsøkte elevene underveis i programmeringen, noe som er anbefalt av Barefoot computing (u.å-f) ettersom det kan gjøre at feilsøkingen bli mer overkommelig.

Den systematiske feilsøkingen elevene i gruppe 5 foretok seg kan ses i sammenheng med trinnene i Barefoot computing (u.å-f) sin feilsøkingsprosess. Elevene lagde en kode som

fortalte hvordan Spheroballen skulle oppføre seg. Deretter testet de den for å finne ut om det de hadde forutsett at skulle skje faktisk skjedde. I dette trinnet ble det oppdaget en feil, og elevene måtte derfor finne ut hva i koden som forårsaket feilen slik at de kunne rette opp i den. Det å følge disse trinnene kan ifølge Barefoot computing (u.å-f) gjøre feilsøkingen mer effektiv, noe som igjen kan føre til at elever i større grad holder ut i programmeringsarbeidet (Barefoot computing, u.å-k).

Arbeid med fysisk programmering ved bruk av Spheroballer kan føre til at det i utgangspunktet blir lettere å oppdage feilene. Grunnen til det er at elevene får mulighet til å visualisere koden de har lagd når Spheroballen gjennomfører kodens handling, noe som ifølge Flowers og Gossett (2002) er en av fordelene med fysisk programmering. Dersom Spheroballens gjennomføring av koden ikke gjenspeiler elevens hensikt for koden, fører dette til at elevene kan konkludere med at det ligger en feil i koden.

Når det blir konstatert at det ligger en feil i koden er det videre nødvendig å undersøke hvor i koden feilen ligger. Elev A2, som arbeidet alene, søkte etter feilen ved å sammenligne den fysiske banen med den programmerte koden. På denne måten kan det virke som at eleven viser forståelse for sammenhengen mellom abstraksjonen og banen, og kan dermed lettere gå inn i koden og rette feilene. En slik oversikt over de ulike delene av koden kan være gunstig ettersom eleven arbeidet alene. Tsai et al. (2015) peker på elevers ulike roller i parprogrammering som en fordel ettersom en elev har i oppgave å se etter feil og sørge for at programmering går fremover. Denne hjelpen mister elever som arbeider individuelt og de må dermed både oppdage og rette feilene på egenhånd. Derfor kan elev A2 sin måte å holde oversikt på bidra til at feilsøkingen blir mer overkommelig.

Gruppe 8 viser at de ikke gikk systematisk til verks da de feilsøkte fordi de prøvde seg frem med tilfeldige koder uten å reflektere rundt hvilke koder som kunne være hensiktsmessig å benytte for å få ønsket utfall. Selv om elev 2 påpekte at de måtte velge koder som påvirket svingen Spheroballen skulle ta, endte gruppen opp med å prøve tilfeldige koder. Gruppe 6

brøyt inn etter at gruppe 8 hadde prøvd seg frem en stund og dette så ut til å skyldes at de så at gruppe 8 stod fast. Dette viser at feilsøkingsstrategien som gruppe 8 benyttet seg av var lite effektiv. Dette stemmer overens med studien til Dong et al. (2019) som viser at en utfordring blant nybegynnere er at de ikke feilsøker på en systematisk måte og dermed har utfordringer med å oppnå ønsket utfall.

Dersom elever fortsetter med en slik prøve-og-feile-strategi kan dette begrense deres mulighet til å utvikle andre strategier innen algoritmisk tenkning (Chevalier et al., 2020). I studien til Chevalier et al. (2020) kom det frem at elever som endte opp med å prøve og feile i programmeringen var elever som fikk fri tilgang på programmet de skulle programmere i og få instruksjoner underveis. Dette var til en viss grad tilfellet i vår studie også. Elevene hadde blitt veiledet i noen strategier på forhånd og underveis. Likevel hadde de fri tilgang på Spheroappen under hele undervisningen, og de ble ikke låst til en fremgangsmåte for å løse utfordringen. Dermed startet elevene rett på programmeringen og brukte mye tid på å prøve og feile.

Fremgangsmåten til gruppe 8 så ut til å være lite effektiv ettersom det var tidkrevende. Dersom elever står fast ved hvordan de skal løse utfordringer vil det kunne svekke elevenes utholdenhet (Barefoot computing, u.å-k). Vi vet derimot ikke om det førte til at de ble mer kjent med kodene og dermed lettere finner en løsning på fremtidige oppgaver.

Resultatene våre viser at flere av gruppene benyttet ulike feilsøkingsstrategier. Dette kan skyldes at det har blitt lagt vekt på feilsøking i introduksjonsøkten. Likevel ser vi at noen grupper benyttet seg av mindre effektive strategier. På en side kunne det vært en fordel for disse gruppene og lære strategier for å feilsøke på en mer systematisk måte, slik som Dong et al. (2019) trekker frem. Det å lære strategier for feilsøking kan videre påvirke elevers utholdenhet ved at strategiene fungerer som en veiledning dersom de står fast, som Barefoot computing (u.å-k) påpeker. På en annen side kan utforskingen elevene gjorde bidra til at

gruppene blir bedre kjent med de ulike kommandoene, som ifølge Dong et al. (2019) kan være en fordel.

For å veilede elevene i feilsøking kan det være nødvendig å understreke at det ikke kun handler om å søke etter riktige svar. Dette vil være hensiktsmessig for læreren å vektlegge i undervisningen, da det å oppdage egne feil og undersøke disse er en viktig del av feilsøkingen (Barefoot computing, u.å-f; Hazzan et al., 2015). Videre kan det planleggeren sa om at det å komme frem til en løsning ofte kreve flere forsøk være viktig å informere elevene om. På den måten formidler læreren til elevene at det å gjøre feil ikke trenger å være negativt, men at det er en viktig del av læringsprosessen.

6.1.3 Holde ut – stå i et problem

Gruppe 5 møtte på en utfordring ved at Spheroballen gikk for langt og dermed havnet utenfor banen. Elevenes løsning var å gjøre flere målrettede justeringer av tiden for å påvirke lengden. Dette gjorde at Spheroballen traff målskiven litt til venstre for 100. Dermed måtte elevene finne en løsning på den nye utfordringen. Gruppe 6 møtte også utfordringer med lengden som de rettet opp i, og deretter landet Spheroballen på 100. De ønsket å prøve en gang til og oppdaget at Spheroballen ikke traff 100 denne gangen. Dermed gjorde gruppen nye endringer for å komme til 100 igjen.

Dette viser at elevene ikke ga opp da de møtte utfordringer, men at de heller fokuserte på hvilke endringer de kunne gjøre i koden. Gruppe 6 visste hvor de skulle gjøre endringer, noe som ifølge Barefoot computing (u.å-k) kan bidra til at elevene holder ut. Gruppene viser også at de evnet å prøve ut flere ulike løsninger da de oppdaget at endringen av lengden førte til en ny utfordring. Det å ikke gi opp selv om elevene møter flere utfordringer viser utholdenhet slik Barefoot computing (u.å-k) beskriver det. Dette kan skyldes at planleggeren hadde en introduksjonsøkt med fokus på at elevene skulle finne ut av hvor feilen skjedde og hvordan de kunne løse feilen slik at elevene ikke skulle gi opp med en gang de møtte utfordringer.

En annen årsak til at elevene holdt ut kan være at elevene synes det var morsomt med fysisk programmering. Dette støttes av (2015) som hevder at elevers positive opplevelse av programmering bidrar til bedre utholdenhet i arbeidet. Det at elever får mulighet til å teste løsningen sin når som helst kan videre være en fordel, da dette vil gi dem en direkte visuell respons som indikerer om løsningen er riktig eller ikke (Flowers & Gossett, 2002). I tillegg trekker Hodges et al. (2013) frem at når elever lager noe kan det gi dem en opplevelse av belønning. I denne studien lagde elevene en kode for å få Spheroballen til å treffe målskiven, og kan dermed ha opplevd det som en belønning dersom de mestret å lage en kode med ønsket utfall.

Vår studie viser at flere av elevene holdt ut og rettet opp i feilene da de møtte på utfordringer. Som nevnt tidligere kan dette skyldes at planleggeren har vektlagt dette i introduksjonsøkten. I tillegg kan det at elevene så hvordan koden deres fungerte visuelt ha påvirket deres utholdenhet og lyst til å gjøre endringer for å oppnå ønsket resultat. Samtidig ser vi at flere av gruppene som holdt ut gjorde målrettete justeringer som forteller oss at de visste hvilken strategi de skulle bruke for å finne en løsning på utfordringene. Som vi videre trekker frem under utfordringer med manglende veiledning (6.2.2) vil det at elevene stod fast kunne påvirke utholdenheten deres i negativ retning.

6.1.4 Samarbeid – bruker medelever som ressurs

Det ble observert at effektiviteten i form av å finne en løsning raskt og kvaliteten i form av resultatet på løsningen til de to klassene som arbeidet individuelt var dårligere enn de to klassene som arbeidet i grupper. Studien til Dybå et al. (2007) kom frem til at en av fordelene med parprogrammering er at effektiviteten med tanke på varighet og kvalitet forbedres. Selv om det ble observert at elevene ikke fikk beskjed om hvordan de skulle samarbeide ser vi likheter mellom fordelene som kom frem i samarbeidet mellom elevene, og fordelene Dybå et al. (2007) presenterer ved parprogrammering. I tillegg viser studien til Tsai et al. (2015) at elevers kognitive belastning er betydelig høyere når elever arbeider individuelt. Dermed kan

dette være årsaker til at elevene som arbeidet individuelt ikke fant løsninger like raskt, og at færre Spheroballer fra de elevene som arbeidet individuelt fullførte banen.

Læreren i de klassene som ble plassert i grupper hadde lagt til rette for samarbeid på forhånd. Læreren hadde satt sammen en elev som hadde erfaring med programmering med en elev som hadde mindre erfaring med programmering, slik at den ene eleven kunne lære den andre. Dette er et eksempel på bruk av jevnaldrende veileder som Sentance og Csizmadia (2017) anbefaler som samarbeidsstrategi. I tillegg hadde læreren satt elevene i gruppe på to til tre og begrunnet dette med at det var for at alle på gruppa skulle bidra. Dette så ut til å fungere etter hensikt for gruppe 5 der elev 1 med erfaring stilte elev 2 uten erfaring spørsmål og ga forklaring på hvilken påvirkning grader hadde på koden deres. Dette viser at elev 2 dro nytte av elev 1 sin kunnskap og erfaringer, som ifølge lærerens hensikt med gruppesammensetningen, Sevik (2016) og Sentance og Csizmadia (2017), er en fordel ved samarbeid i programmering. Videre lyttet elev 1 til elev 2 og fylte inn forslagene elev 2 kom med i koden. Dette viser at elevene i denne gruppen lyttet til hverandre og var villig til å teste hverandres løsninger, slik Ludvigsen et al. (2015) hevder er grunnleggende i samarbeid. Selv om elev 2 hadde mer erfaring viste eleven respekt for elev 1 sitt løsningsforslag ved å sette det inn i koden.

Selv om det ikke er observert at læreren snakket om hvordan elevene skulle samarbeide vil gruppe 5 sin måte å samarbeide på kunne knyttes til parprogrammering. Dette fordi elev 1 virket å være sjåfør som satt inn koder og styrte programmet og elev 2 var kartleseren som passet på at Spheroballen beveget seg i riktig retning. Da elev 2 oppdaget at Spheroballen stoppet for tidlig ga eleven beskjed til elev 1. Rollene elevene hadde i denne gruppen stemmer overens med Sevik (2016) og Barefoot computing (u.å-c) sin beskrivelse av parprogrammering. I tillegg til at kartleseren (elev 2) oppdager feil kommer eleven med en mulig endring på utfordringen, som Beck et al. (2005) og Tsai et al. (2015) hevder er en del av jobben til kartleseren.

Det at begge elevene i gruppe 5 fikk mulighet til å bidra i fellesskapet kan videre være årsaken til at elev 2 ikke ønsket å bytte på hvem som holdt iPaden da læreren ga beskjed om at elevene skulle bytte. Dette fordi eleven opplevde å bli lyttet til og få være med å bidra selv uten å styre iPaden. Selv om det ser ut til at begge elevene fikk utbytte av programmeringen og viste forståelse for feil i koden og løsninger på feilene ville det vært hensiktsmessig for elev 2 å skrive inn koder i selve programmet. Dette støttes av Barefoot computing (u.å-c) som mener at elevene må bytte rolle underveis for at begge skal få erfaring med de to rollene.

Gruppe 7 er et eksempel på en gruppe der den ene eleven stod for kodingen og kan ses på som en sjåfør, mens den andre eleven hverken så ut til å oppdage feil eller komme med løsningsforslag, noe som ville vært kartleserens ansvar ved parprogrammering (Beck et al., 2005; Tsai et al., 2015). I stedet for å ha rollen som kartleser ser det ut til at eleven hadde rollen som motivator ved å komme med heiarop og klapping underveis. På en side kan motivatoren ha påvirket prestasjonen til eleven som utarbeidet algoritmen fordi heiaropene kunne gjøre at eleven trivdes bedre i samarbeidet. På den andre siden kan det hende at motivatoren heier fordi gruppen presterte bra og at dette bidro til at de trivdes i samarbeidet. Dette støttes av Muller og Padberg (2004) som hevder at det er en korrelasjon mellom trivsel og prestasjon i samarbeid.

Det ser imidlertid ikke ut til at motivatoren lærte hvordan selve programmeringen fungerte. For at begge elevene på gruppen skulle ha utviklet ferdigheter i programmering vil lærerens rolle være sentral i dette samarbeidet. Blatchford et al. (2003) hevder at det en fordel at læreren kjenner elevene for å kunne sette sammen grupper som har grunnlag for å fungere godt. I intervjuet kom det frem at læreren hadde gjort dette ettersom læreren hadde satt sammen grupper bestående av en elev med mer erfaringer med programmering og en med mindre erfaring, på bakgrunn av lærerens kjennskap til elevene. Imidlertid ser det ikke ut til at dette bidro til at elevene lærte av hverandres erfaringer slik som læreren mente var hensikten med denne sammensetningen.

Våre resultater viser at flere av gruppene fikk et positivt utbytte av samarbeidet. Noen grupper benyttet seg av hverandres ulike erfaring med programmering og fordelte ansvaret i

programmeringen. Denne fordelingen av ansvar innad i gruppen så ut til å fungere godt for elevene, og samarbeidsstrategien ligner på hvordan roller fordeles i parprogrammering. Dermed kan det være hensiktsmessig å benytte parprogrammering i programmeringsundervisning slik at flere elever deltar aktivt i samarbeidet og utviklingen av koden.

6.1.5 Dekomposisjon og mønstre - dele opp problemet og identifisere mønstre

I undervisningen til de to klassene der elevene samarbeidet ga læreren beskjed om at elevene kunne bruke to bord hver som testbane mens de programmerte. Elevene i gruppe 4 kom frem til at ettersom bordene var like lange kunne de bruke de to bordene de hadde fått utdelt til å programmere alle delene av banen, uten at det var nødvendig å flytte på bordene. Andre elever flyttet de to bordene slik at de samsvarte med svingene i hovedbanen.

Ved å bruke to bord delte elevene opp hovedbanen til flere små deler, og de kunne dermed konsentrere seg om et delproblem av gangen. En slik oppdeling av problemet er et eksempel på dekomposisjon, og det kan gjøre det lettere å finne en løsning på det overordnede problemet (Anderson, 2016). I tillegg kunne bordene, som fungerte som konkreter, gjøre det lettere for elevene å visualisere hver del av banen og dermed gjøre det lettere å se koblingen mellom banedelen og koden de lagde.

Dekomposisjonen elevene gjorde kan knyttes til forberedelsesfasen i Schulz og Pinkwart (2016) sin modell for problemløsning i fysisk programmering. Elevene i gruppe 4 brukte bordene til å dele opp problemet. Videre brukte de observasjonen om at alle bordene var like lange til å lage en plan for hvordan de skulle lage koden. Det å dele opp problemet og lage en plan videre er viktige deler av forberedelsesfasen (Schulz & Pinkwart, 2016). Videre viser elevene i gruppe 4 at de mestret implementeringsfasen, ved at de klarte å lage koder som samsvarte med banens form. Det at de kodet uten å flytte på bordene kan gi inntrykk av at de så mønstrene i banen. Elevene forstod at strekningene rett frem kunne kodes likt, samtidig som svingene brukte samme kommando, men ulik svinggrad.

En annen gruppe som viser at de hadde funnet repetitive mønstre i banen var gruppe 9. Elevene i denne gruppen startet med å dele opp problemet, og testet koden for å få Spheroballen til å kjøre rett frem og svinge. Elevene oppdaget raskt at de ikke trengte å teste alle de videre delene av algoritmen. De brukte kunnskapen de hadde tilegnet seg om mønstre til å bygge hele koden. Dette kan ses på som en form for prototype-basert fikling (Dong et al., 2019). Gruppe 9 lagde en prototype for en spesifikk del av koden og brukte denne videre i arbeidet med å skape algoritmen. Det kommer imidlertid ikke tydelig frem om elevene kopierte opp prototypen av koden eller bygget en lik kode gjentatte ganger. Uansett har elevene vist at de ser mønstrene i banen og kan gjenskape disse mønstrene i koden.

Gruppe 6 lagde kodene separat i programmet på iPaden for så å sette de sammen til en algoritme. Dette ga inntrykk av manglende evne til å forutse utfordringer som kunne oppstå underveis som er en sentral del av forberedelsesfasen, ifølge Schulz og Pinkwart (2016). Ved å diskutere mulige utfordringer kan det ifølge Schulz og Pinkwart (2016) bli mindre kognitivt belastende å møte på disse utfordringene senere, enn dersom de kommer uforutsett. Elevene i gruppe 6 tok ikke høyde for at Spheroballen måtte trille ekstra langt for å sikre overgangen fra et bord til et annet. Dermed trillet Spheroballen annerledes enn det elevene hadde sett for seg da de kodet, og det faktiske utfallet ble annerledes fra det ønskede utfallet. Elevene oppdaget imidlertid raskt at noe var feil, og ettersom de hadde brukt tid på å lage koden i separate deler kunne de raskt gå inn i koden og undersøke hvor feilen lå. Elevenes feilsøking gir inntrykk av at de hadde forstått at de måtte legge til lengde på noen av kodene. Deretter testet de hele koden etter hvert som de gjorde endringer. Ettersom elevene både mestret å finne feilene og rette dem opp kan det ifølge Sentance og Csizmadia (2017) føre til at de utvikler programmeringsferdighetene sine, samt evnen til å stå i et problem.

I de to klassene der elevene arbeidet individuelt informerte ikke læreren om elevenes mulighet til å bruke bordene til å dele opp den opprinnelige banen. Dette ser ut til å påvirke hvordan elevene gikk frem i programmeringen. Noen elever programmerte ved å endre farge på Spheroballen. Disse elevene brukte tiden på fargevalg fremfor å utvikle en kode som skulle få

Spheroballen til å gå fra start til mål på banen. Dette kan gi inntrykk av at elevene mistet fokus på oppgaven. For å hjelpe elevene i gang med oppgaven kunne læreren introdusert dem for dekomposisjon, slik som ble gjort i de klassene der elevene samarbeidet. Ifølge Anderson (2016) og Barefoot computing (u.å-g) kan dekomposisjon gjøre programmeringen mer håndterlig, og i dette tilfelle kan det bidra til å veilede elevene til å konsentrere seg om oppgavens hensikt. På tross av at læreren ikke hadde oppfordret til dekomposisjon så vi at elev A4 delte opp koden på egenhånd ved å legge til koder underveis. Dette viser behovet eleven hadde for å bryte ned problemet i mindre deler.

Flere av elevene i vår studie benyttet seg av dekomposisjon i starten av undervisningen. Elevene valgte imidlertid ulike veier videre i programmeringen. Noen elever fortsatte med dekomposisjon og brukte bordene til å lage en og en del av koden. Andre lagde resten av koden basert på det de lærte av å lage første del av koden. På den ene siden kan det at elevene stod fritt til å velge veien videre bidra til å ivareta elevenes frihet til å ta valg og mulighet til å utforske, noe som ifølge lærerne i studien til Sentance og Csizmadia (2017) er en viktig forutsetning i programmeringsoppgaver. På den andre siden kan mye frihet og få føringer gjøre programmeringen mer krevende dersom elevene ikke klarer å sette sammen det de har lært til en kode, noe som elevene i studien til Sentance og Csizmadia (2017) opplevde som krevende. Dette kom også frem i vår studie der vi så elever som stod fast i programmeringen på ulike måter, og dermed så ut til å ha behov for videre veiledning av læreren. Dette vil bli drøftet videre i delkapittel 6.2.2.

Resultatene i denne studien viser at flere av elevene fikk utbytte av oppfordringen til dekomposisjon, da dekomposisjon fører til at programmeringen blir mer håndterlig (Anderson, 2016; Barefoot computing, u.å-g). For elever som kjenner igjen dekomposisjon fra arbeid med problemløsning kan dette være en måte å knytte sammen problemløsning og programmering, og dermed kan elevene bruke tidligere kunnskap i møte med programmering. Behovet for opplæring i dekomposisjon støttes av studien til Sentance og Csizmadia (2017) som understreker at elevene slet med nettopp det å bryte problemet ned i mindre deler.

6.1.6 Evaluering – vurdere løsningen

Elevene er avhengig av å evaluere gjennomføringen opp mot kriteriet for å løse problemet, slik Sevik (2016) påpeker. Vi observerte elever som evaluerte løsningen sin slik at de kunne rette opp i feil underveis og forbedre løsningen. I delkapittel 6.1.3 er det beskrevet at det ser ut til at selv om gruppe 5 fikk Spheroballen til å lande på målskiven ville gruppen øke kvaliteten på løsningen ved å få ballen til å lande på 100. Det å vurdere kvaliteten på løsningen er en del av det å evaluere (Barefoot computing, u.å-h).

Gruppe 3 vurderte effektiviteten, som er et annet element Barefoot computing (u.å-h) trekker frem at elevene skal vurdere. De kom frem til at det å få Spheroballen til å gå på skrå fører til at den *"slipper å gå rundt hele svingen"*, som elev 1 sier. Videre sier eleven at dette vil gjøre at Spheroballen går raskere, og kommer med forslag til hvordan dette kan gjøres gjennom å endre gradene. Dette viser at eleven vurderer løsningens effektivitet opp mot formålet, slik Barefoot computing (u.å-h) trekker frem at er en del av evalueringsprosessen.

Elev A3 kom frem til to løsninger for å få Spheroballen til å gå lenger. Dette gjorde at eleven måtte vurdere hvilke av løsningene som var hensiktsmessig å benytte. Den første løsningen ser ut til å handle om å endre koden for å få Spheroballen til å gå lenger, og dermed kan eleven gjøre et logisk resonnement ut ifra erfaringen om hva som påvirker lengden i koden som er utarbeidet. Den andre løsningen handler om å flytte startpunktet til Spheroballen og dermed slipper eleven å endre kode. Ulempen med å flytte startpunktet er at det kan være utfordrende å huske hvor eleven skal starte Spheroballen fordi den da skal plasseres lenger frem enn "startmerket".

Gruppe 1 kopierer andres koder uten å sette seg inn i kodens funksjon og dette vises i elevenes utsagn: *"Jeg forstår ikke? Hva gjorde vi annerledes nå?"* og *"Hva gjorde vi i stad egentlig?"* Denne strategien viste seg å være lite effektiv ettersom gruppen stod fast ved neste del av programmeringen. En årsak til at elevene kopierte andre elevers koder kan være mangel på grunnleggende forståelse i programmering. Dette trekker lærerne i Sentance og Csizmadia (2017) sin studie frem som en vanlig utfordring for elevene. Samtidig trekker Lee

et al. (2011) frem strategien å kopiere andres løsning som en grunnleggende del av det pedagogiske rammeverket Use-Modify-Create. I denne modellen skal programmeringen starte med å kopiere andres koder, og det skal kunne bidra til å utvikle elevene forståelse i programmering. Da er det imidlertid viktig å sette seg inn i hvordan koden fungerer, fremfor å kun kopiere den (Lee et al., 2011). Dette gjør ikke elevene i gruppe 1 og en grunn til dette kan være at de var fornøyd så lenge de fikk Spheroballen i mål.

Den største begrensingen elevene i vår studie fikk var at de kun fikk lov til å teste Spheroballen på hovedbanen to ganger. Dette kan ligne på en av begrensningene elevene i testgruppen til Chevalier et al. (2020) fikk, der de på et tidspunkt ikke fikk lov til å kjøre koden på roboten. Dette kan oppmuntre elevene til å gå igjennom hver del av koden før de tester. På denne måten må elevene sette seg inn i hvordan og hvorfor de har kodet og diskutere hva utfallet vil bli. Dermed kan en slik begrensing bidra til at elevene utvikler og bruker andre strategier fra den algoritmiske tenkeren.

For å bidra til at elevene setter seg inn i koden og forstår hva de har gjort kunne læreren lagt vekt på at elevene skulle evaluere underveis slik at de måtte sette seg inn i hvordan koden fungerte eller ikke fungerte. Dette kunne blitt gjort ved at elevene skulle legge inn kommentarer i koden som fortalte hva de ulike delene gjorde, lagt til lyd der elevene beskrev hvordan de hadde utviklet koden, eller fortalt andre hva de hadde lært, som er noen av forslagene Brennan og Resnick (2012) trekker frem for at elevene skal sette seg inn i egen tankegang.

Våre funn viser at flere elever evaluerte løsningene sine underveis, mens andre elever hadde utfordringer med dette. Elevene så hvordan løsningene endte opp i form av Spheroballens plassering i forhold til målskiven. Likevel var det ikke alle som vurderte hva de hadde gjort for å komme frem til ønsket resultatet, eller hvordan de kunne endre koden for å oppnå dette. Ved å vektlegge evaluering underveis og som en avslutning på undervisningen kan elever bevisstgjøres på hva, hvordan og hvorfor de har programmert, slik som Sevik (2016) trekker

frem som et viktig element i programmering. I en slik evaluering kan lærere bidra med spørsmål rundt hva som kan være årsaken til at noen av Spheroballene kom eller ikke kom frem til målskiven.

6.1.7 Fikle – utforsking av kodene

Resultatene fra observasjonen viser at flere elever startet programmeringen med å utforske hvordan de ulike kommandoene påvirket Spheroballens bevegelser. Elevene i gruppe 7 ga uttrykk for at de var usikre på hvordan de skulle angripe utfordringen, men ble enige om å prøve seg frem. Elevene benyttet fikling som strategi da de testet de ulike kommandoen. Det at elevene utforsket de ulike blokkene kan ha ført til at usikkerheten rundt blokkenes funksjon stadig ble mindre og at elevene dermed kom i gang med programmeringen. Det å prøve seg frem dersom en er usikker kan ifølge Barefoot computing (u.å-l) bidra til å styrke fiklingen. Dette samsvarer med Dong et al. (2019) som hevder at elevenes uttrykk for usikkerhet er en viktig del av utforskingen.

Elevenes videre utforsking i programmeringen viser at fiklingen pågikk gjennom hele undervisningen. Ettersom fikling består av utforsking (Barefoot computing, u.å-l), og utforsking er en sentral del av programmering, kan dette støtte påstanden om at elevene fiklet gjennom hele undervisningen. Utforskingens sentrale del i programmering kommer til syne i kompetansemålene i læreplanen der verbet *utforske* står i forbindelse med programmering (Utdanningsdirektoratet, 2020b, 2020c). I tillegg kan det at elevene arbeider med Spheroballer, som er en form for fysisk programmering, bidra til at de gjennom hele undervisningen er i kontakt med roboten som skal utføre algoritmen de koder. Dermed kan det ses på som en form for direkte utforsking, som ifølge Barefoot computing (u.å-l) er en del av definisjonen på fikling.

Fikling kan komme til syne i flere av de andre strategiene i den algoritmiske tenkeren. I dekomposisjon kan fikling gjenkjennes ved at elevene deler opp problemet og utforsker én del av gangen. Gjennom å oppfordre til dekomposisjon kan det i denne studien se ut til at elevens utforsking ble påvirket ved at flere elever utforsket deler av problemet fremfor å gape over en

for stor del av programmeringen fra starten av. Videre kan fikling gjenkjennes i feilsøking ved at det å identifisere og rette opp feil i koden kan ses på som en måte å utforske og evaluere kodens funksjonalitet. Det at fikling inngår i andre strategier støttes av studien til Dong et al. (2019) der feilsøking og dekomposisjon kom frem som viktige elementer i elevenes fikling. Det vil derfor være nyttig for naturfagslærere å sette seg inn i alle strategiene i den algoritmiske tenkeren. En slik helhetlig forståelse kan gjøre det lettere å oppmuntre elevene til utforsking ved at elevene kan benytte seg av andre strategier. Dersom utforskingen veiledes med effektive strategier, kan programmeringen virke mer overkommelig for elevene.

Videre er blokkbasert programmering designet for å oppmuntre til fikling gjennom å utforske de ulike blokkene (Maloney et al., 2010; Taraldsen & Myhra, 2019). Ettersom utforsking er nært knyttet til programmering i kompetansemålene vil det være hensiktsmessig for lærere å inkludere blokkbasert programmering i naturfagundervisningen. I tillegg kan dette med fordel utvides til fysisk programmering slik at læringen støttes gjennom bruk av fysiske gjenstander. Dette samsvarer med Hodges et al. (2020) som hevder at fysisk programmering kan bidra til økt engasjement og motivasjon gjennom å synliggjøre programmeringsresultatene i en fysisk gjenstand.

6.1.8 Logikk – forutsi og forklare hvorfor noe skjer

Etter at gruppe 2 hadde fått Spheroballen til å gå rett frem kom de frem til at de måtte få Spheroballen til å svinge for å følge banen. De begrunnet løsningen på denne utfordringen med et matematisk resonnement ved å si at for å få til en skarp sving måtte de legge inn 90 grader i koden. Elevene ga uttrykk for at de klarte å forutse hva som kom til å skje, noe som stemmer overens med Barefoot computing (u.å-i) sin definisjon av logikk. Videre skulle gruppe 2 få Spheroballen til å svinge motsatt vei og gjorde et logisk resonnement på bakgrunn av erfaringen med gradene fra første sving. De forklarte hva som fikk ballen til å gå til høyre og forutså videre at løsningen på en venstresving da måtte være å sette inn 270 grader i koden. Dette viser at elevene både klarte å forklare hva som skjedde og hva som kom til å skje ved bruk av elevenes matematiske kunnskap og erfaring med en forutsigbar respons i koden.

Evnen til å forklare og forutse på bakgrunn av forutsigbare responser er ifølge Barefoot computing (u.å-i) grunnleggende ved logiske resonnementer.

Som nevnt i delkapittel 6.1.2 brukte gruppe 8 mindre effektive feilsøkingsstrategier. Elevene testet nye tilfeldige blokker for å forsøke å få Spheroballen til å svinge riktig vei. Ved bruk av nye blokker kunne ikke elevene forutse hva som kom til å skje ettersom de ikke hadde testet ut responsen disse blokkene ga. På en side kan det argumenteres for at gruppen brukte erfaringene, fra responsen de hadde observert da de testet koden, til å videreutvikle koden ettersom de så at koden ikke fungerte som ønsket. På den andre siden virket det likevel som at elevene ikke forstod hva blokkene de i utgangspunktet hadde brukt innebar. Elev 2 ga imidlertid uttrykk for at eleven visste at de rosa blokkene ikke ville påvirke svingen, men elev 1 ville likevel teste blokken. Det kan virke som elevene ikke utnytter hverandres kunnskaper, som er en av fordelene Sevik (2016) trekker frem med samarbeid. Dermed valgte de å bytte ut den ene blokken fremfor å gjøre justeringer på de blokkene som egentlig ga riktig respons. Det kan dermed virke som at gruppe 8 ikke evnet å forutse og gjør endringer basert på erfaringer i like stor grad som elevene i gruppe 2.

Resultatene i vår studie viser at elever kan få utbytte av å bruke kunnskap fra matematikk når de koder i naturfag, og programmeringens tverrfaglighet kommer dermed frem. Elevenes bruk av kunnskap fra andre fag styrket den logiske tenkningen ved at de evnet å forutse kodens utfall da de gjorde endringer. Samtidig kommer behovet for logisk tenkning i programmering frem ved at elevene som ikke evnet å gjøre logiske resonnementer arbeidet mindre effektivt i programmeringen fordi de testet koder som ikke var relevant for løsningen.

6.1.9 Oppsummering av kjennetegn på algoritmisk tenkning

Ut ifra denne studiens resultater ser vi at flere av strategiene fra den algoritmiske tenkeren kommer til uttrykk i elevenes arbeid med programmering i naturfag. Noen av strategiene var det lagt opp til at elevene skulle kjenne til, som feilsøking og holde ut som det hadde vært fokus på i introduksjonsøkten. Dette kan være årsaken til at flere elever mestret disse

strategiene og det viser effekten undervisningen i strategiene fra den algoritmiske tenkeren kan ha. Videre oppfordret læreren elevene til å dele opp programmeringen ved å bruke bord til å programmere en og en del av hovedbanen. På denne måten kom dekomposisjon frem. Oppfordringen til dekomponering sammen med fokuset på feilsøking i introduksjonsøkten kan ha påvirket elevenes evne til utholdenhet.

Det var lagt opp til samarbeid i to av klassene, og strategiene skape, algoritme og abstraksjon kom til syne som overordnede strategier for undervisningen. Samarbeidet ble påvirket av lærerne ved at gruppestørrelsen var begrenset til to-tre elever. De små gruppene kan ha ført til at begge gruppemedlemmene bidro i større grad. I tillegg ble elever med lite erfaring plassert sammen med elever med mer erfaring i programmering og elevene fikk dermed jobbe med en jevnaldrende veileder.

Utforsking og evaluering var strategier elevene benyttet seg av gjennom hele undervisningen ved at de utforsket hvordan de kunne lage kodene og vurderte hvordan kodene kunne forbedres. På denne måten kom disse strategiene frem samtidig som elevene arbeidet med andre strategier. Elevers evaluering kan styrkes dersom det legges opp til en form for evaluering på slutten av timen. I tillegg kan fikling styrkes gjennom bruk av blokkbasert programmering som er designet for å oppmuntre til fikling. Elevenes evne til å gjøre logiske resonnementer kom også frem gjennom hele programmeringen. Her så vi utbyttet elevene fikk av matematiske kunnskaper i programmeringen og hvordan dette bidro til å gjøre programmeringen mer effektiv.

Vi ser at lærerens påvirkning kan ha noe å si for hvilke strategier elevene benytter seg av. Ved å lære elever strategier innenfor den algoritmiske tenkeren kan dette videre bidra til å utvikle elevenes ferdigheter i programmering. Dermed understrekes behovet for lærerens veiledning i programmeringsundervisning. Dette drøftes videre i delkapittel 6.2.2.

6.2 Utfordringer i programmering

Resultatene fra analysen av intervjuene og observasjonene viser at både lærerne og elevene møtte på utfordringer knyttet til programmering. Vi har valgt å drøfte disse utfordringene under de fire induktive kategoriene fra analysen. I tillegg blir drøftingen av utfordringene knyttet opp til drøftingen av strategiene fra den algoritmiske tenkeren i delkapitlene over. Avslutningsvis oppsummerer vi de viktigste poengene innenfor diskusjonen rundt utfordringer i programmering.

6.2.1 Utfordringer rundt kommunikasjon om den algoritmiske tenkeren

Undervisningsøkten som har blitt fulgt i denne studien ble planlagt av en lærer og gjennomført av andre lærere. Planleggeren trakk frem at utviklingen av undervisningsopplegget var basert på flere av strategiene fra den algoritmiske tenkeren, med vekt på arbeidsmåtene.

Det at lærerne som skulle undervise i programmering brukte en annen lærer, med god kunnskap innenfor programmering, som en ressurs støttes av Sentance et al. (2017) sine råd for implementering av programmering i skolen. I tillegg viser Stenseth et al. (2019) at det er i planleggingen av undervisningen at lærerens kunnskaper er viktigst. Dermed vil det være hensiktsmessig å inkludere en lærer med mye kunnskap i denne prosessen.

Læreren skal imidlertid også veilede elevene underveis i undervisningen. Derfor er det viktig at læreren er godt kjent med innholdet i undervisningen. I vår studie kom det frem at kommunikasjonen rundt den algoritmiske tenkeren var mangelfull mellom planleggeren og læreren. I tillegg syntes læreren at algoritmisk tenkning var et vanskelig begrep og mente egne faglige begrensninger var en utfordring i programmeringsundervisning. Disse begrensingene kan være årsaken til at læreren oppfordret elevene til å spørre hverandre om hjelp, og at læreren holdt seg i bakgrunnen.

Det at lærere er utrygge på programmering kom dessuten frem i studien til Sentance og Csizmadia (2017). Med de mange definisjonene av computational thinking (Barr et al., 2011; Coulter et al., 2010; Council, 2010; Voogt et al., 2015) og det faktum at programmering har en kompleks sammensetning, kan det være vanskelig for lærere å få tak i hva begrepet egentlig innebærer. Utdanningsdirektoratet (2019) sin beskrivelse av den algoritmiske tenkeren kan imidlertid ses på som en måte å gjøre algoritmisk tenkning mer håndgripelig gjennom å gjøre kunnskapen tilgjengelig for lærere. Da vi leste om den algoritmiske tenkeren på Utdanningsdirektoratets nettsider opplevde vi at informasjonen om de ulike strategiene var lite utdypende. Dette førte til at vi hentet mer informasjon fra Barefoot computing (u.å-d), der en finner den opprinnelige modellen som den algoritmiske tenkeren er basert på. Det er imidlertid ikke en selvfølge at lærere setter seg inn i modellen utover det som står om den algoritmiske tenkeren hos Utdanningsdirektoratet (2019). Dermed kan det påvirke i hvor stor grad lærere setter seg inn i modellen. Ifølge Sentance et al. (2017) er det å gi lærere profesjonell opplæring i programmering et viktig grep som bør gjøres for å bidra til en programmeringsundervisning preget av motivasjon og effektivitet. Ettersom den algoritmiske tenkeren kan kobles til programmering kan profesjonell opplæring i denne modellen være hensiktsmessig. Ved å sikre en god opplæring av lærere i den algoritmiske tenkeren, kan det bidra til at det blir lettere å videreføre modellens strategier til elever.

En studie gjort av Medeiros et al. (2018) viser at elever ofte viser manglende strategier i problemløsning. Ettersom den algoritmiske tenkeren er en modell for problemløsning, understreker dette nytten av at lærere setter seg inn i modellen. Sørby og Angell (2012) peker på viktigheten av at lærere følger opp elevene gjennom å bevisstgjøre dem på hvilke strategier de bruker. Når dette ikke blir gjort på grunn av manglende kunnskap hos lærere kan dette ifølge Medeiros et al. (2018) føre til at elever utvikler egne, mindre effektive strategier. Dette så vi tilfeller av i undervisningen og utfordringer rundt dette, samt tiltak som kan gjøres, vil bli drøftet i neste delkapittel (6.2.2).

6.2.2 Utfordringer knyttet til behov for veiledning

I resultatene kom det frem at flere elever ga uttrykk for at programmeringen var krevende, og de hadde dermed behov for veiledning. Noen av elevene arbeidet individuelt, og brukte mye tid på å stille sprømål til både medelever og læreren da de stod fast. Andre elever kopierte andre gruppers løsninger fremfor å prøve seg frem på egenhånd.

Ifølge lærerne i studien til Sentance og Csizmadia (2017) bør lærere oppfordre elever til å finne ut av feil på egenhånd fremfor at lærere gir dem svaret. Dette skal gi elevene øving i å stå i et problem. I vår studie fikk elevene rom til å utforske utfordringen på egenhånd, da læreren holdt seg i bakgrunnen. Likevel så vi at elev A1 hadde utfordringer med programmeringen ettersom eleven spurte medelever om hjelp gjennom hele undervisningen. Vi har tolket dette som at eleven ikke hadde tilstrekkelige ferdigheter og erfaring med å løse utfordringer i programmering, og det kan virke som at elev A1 spurte andre elever for å ikke stå fast. Dette vil imidlertid ikke bidra til å utvikle elevens programmeringsferdigheter med mindre eleven forstår hva og hvorfor de andre elevene har programmert slik de har gjort. Derfor kan det se ut til at elev A1 trenger å utvikle andre strategier som kan brukes når eleven står fast. Både læreren i vår studie og lærerne i studien til Sentance og Csizmadia (2017) sa at utholdenhet var en gjennomgående utfordring i programmeringsundervisning. Dette understeker viktigheten av behovet for veiledning i strategier, noe som videre kan bidra til å utvikle elevers algoritmiske tenkning (Sentance & Csizmadia, 2017).

Det at noen elever i vår studie utviklet egne mindre effektive strategier viser at det var et behov for lærerens veiledning. I undervisningen var det et bevisst valg av læreren å holde seg i bakgrunnen ettersom læreren mente elevene var drillet i programmeringen. Dette kan gi elevene mulighet til å utforske på egenhånd. Likevel har elevene ofte et behov for støttestrukturer. Elever har krav på differensiering, men i studien til Sentance og Csizmadia (2017) trekker lærere frem at de syntes det er spesielt krevende å differensiere i programmering ettersom elevene ofte ligger på svært ulike nivå. Det finnes likevel noen tiltak som kan bidra til differensiering.

Først og fremst kan lærere benytte seg av rammeverket Use-Modify-Create, utviklet av Lee et al. (2011). Programmeringsoppgaven kan dermed differensieres ved at noen elever får i oppgave å skape en helt ny kode, samtidig som andre elever kan ta utgangspunkt i en allerede eksisterende kode, for så å modifisere den. Det å ta utgangspunkt i en allerede eksisterende kode kan bidra til at elevene utvikler mer komplekse koder enn de hadde klart å utvikle på egenhånd, ifølge Brennan og Resnick (2012). Dermed kan det å bruke en eksisterende kode differensieres videre ved at noen elever utvikler komplekse koder med krevende endringer, mens andre elever kan gjøre mindre krevende endringer av den eksisterende koden. Dette kan ses i sammenheng med Barefoot computing (u.å-e) sin beskrivelse av å skape, som både kan handle om å skape noe fra bunnen av, men også det å forbedre et allerede eksisterende produkt.

En annen måte lærere kan differensiere på er ved å gi elevene ulik veiledning underveis. Studien til Tyrén et al. (2018) viser at elever foretrekker muntlige svar når de står fast. Likevel vil dette være krevende dersom det er en stor andel av elevene som trenger hjelp samtidig. Dermed kan lærere differensiere ved at elevene bruker medelever og/eller videoinstruksjoner som veiledere, der sistnevnte også blir trukket frem som en foretrukken veiledning av elever i Tyrén et al. (2018) sin studie. Dersom flere av elevene kan få utbytte av slik veiledning kan lærere sette av mer tid til muntlig tilbakemelding til elever som for eksempel begrenses av manglende programmeringsferdigheter.

6.2.3 Utfordringer med samarbeid

Som nevnt tidligere under delkapittel 6.1.4 la læreren som ble intervjuet til rette for samarbeid gjennom gruppestørrelse på to til tre elever med hensikt om at alle parter skulle delta aktivt i samarbeidet. I tillegg ble elevene satt sammen etter erfaring og kunnskap slik at de kunne få utbytte av hverandres kunnskap og hjelpe hverandre. Dette samsvarer med det studien til Sentance og Csizmadia (2017) kaller jevnaldrende veileder, som er en samarbeidsstrategi flere av lærerne i studien hadde god erfaring med. Dermed vil samarbeidsstrategien læreren la til rette for i den observerte undervisningen videre bli omtalt som jevnaldrende veileder.

Selv om læreren la til rette for jevnaldrende veileder ble det likevel observert at dette ikke fungerte etter hensikt for gruppe 1. Eleven som hadde vært på introduksjonsøkten og skulle fungere som en veileder ignorerte spørsmålene til medeleven som ikke hadde vært til stede på introduksjonsøkten. Det ble videre observert grupper der elevene avviste hverandres løsningsforslag. Dermed så det ikke ut til at lærerens hensikt med samarbeidet ble oppnådd av disse gruppene selv om læreren gjorde tilpasninger for å påvirke samarbeidet. Dette kan skyldes at det er flere faktorer enn gruppestørrelse og sammensetning av elever som påvirker samarbeidet.

Säljö (2001) trekker frem kommunikasjon som en viktig faktor for et godt samarbeid. Videre trekkes det frem at kommunikasjon om forståelse er grunnleggende ettersom mennesker forstår ting ulikt (Säljö, 2001). Ettersom gruppene bestod av elever med ulikt utgangspunkt og at det ble observert at elevene hadde ulik forståelse, vil kommunikasjon være relevant for gruppene i vår studie for å sørge for god kunnskapsoverføring mellom elevene. I gruppe 1 kunne elev 2, som ikke hadde deltatt på introduksjonsøkten, fått en økt forståelse for programmeringen dersom elev 1 hadde satt seg inn i elev 2 sin forståelse og besvart spørsmålene. Dette fordi elever med lite erfaring kan ha utfordringer med å forstå hvordan programmet fungerer og hvordan de kan bruke programmet (Tyrén et al., 2018). Dette kan være årsaken til at det er ulikt i hvor stor grad elevene i gruppe 1 er delaktige i programmeringen. I tillegg kunne det at elev 1 hadde forklart til elev 2 gitt elev 1 en mulighet til å teste egen kunnskap. Dette kunne ført til at elevene hadde fått et bedre utbytte av samarbeidet. Dette støttes av Sevik (2016) som hevder at hensikten med samarbeid er å få utbytte av hverandres kunnskaper og erfaringer.

Studien til Muller og Padberg (2004) viste at det var en korrelasjon mellom gruppens prestasjon og hvor komfortable elevene i gruppen var med samarbeidet. McDowell et al. (2003) kom frem til at samarbeidet også kan påvirkes av i hvor stor grad elevene ser på samarbeidspartneren sin som dyktig. Årsaken til at elev 1 i gruppe 1 ignorerte spørsmålene fra elev 2 kan dermed ha vært at elev 1 så på det å lære opp medeleven som mer tidkrevende enn å gjennomføre programmeringen selv.

Et annet tiltak læreren gjorde for å legge til rette for at begge elevene i læringsparet skulle prøve seg på programmeringen var å be elevene om å bytte på hvem som holdt iPaden. Dette ble imidlertid ikke fulgt opp, og det ble observert at noen elever unngikk å bytte iPad. For å unngå at elever er passive i programmeringen kunne læreren ha benyttet parprogrammering som en samarbeidsstrategi og gitt elevene innføring i de ulike rollene, der sjåføren skal skrive koder og styre programmet og kartleseren skal se etter feil og mulige endringer, slik Barefoot computing (u.å-c), Beck et al. (2005), Sevik (2016) og Tsai et al. (2015) beskriver det. På denne måten kan en unngå at en elev påtar seg en mer observerende og passiv rolle i samarbeidet, som vi så tilfeller av i undervisningen. Ved å ha tydelige roller vil elever lettere kunne bidra i samarbeidet, og dette kan føre til at samarbeidet fungerer bedre (Barefoot computing, u.å-c).

Dette viser at selv om læreren gjør bevisste valg for å påvirke samarbeidet, som gruppestørrelse, elevers erfaring og det å gi elevene beskjed om å bytte, fører det ikke nødvendigvis til at hensikten med samarbeidet oppnås. Elever bør få en innføring i hvordan de skal samarbeide for at de skal få utbytte av samarbeidet (Ludvigsen et al., 2015). Dermed bør lærere legge vekt på hvordan elevene skal samarbeide i naturfagundervisning med programmering.

I denne studien kom det frem at flere av elevene har utfordringer med kommunikasjon i form av å lytte til hverandre, forklare og dele ideer. I følge Blatchford et al. (2003) er dette noe læreren må lære elevene. Dette kan for eksempel gjøres gjennom å benytte parprogrammering der elevene bevisstgjøres over egen rolle. Videre vil det være behov for at læreren følger opp og veileder elevene dersom de er usikre på egen rolle slik at alle på gruppen bidrar til å utvikle algoritmen. Dette stemmer overens med Aakervik et al. (2006) som hevder at læreren skal fungere som en organisator og veileder i samarbeidet.

6.2.4 Teknologiske utfordringer

Den teknologiske utfordringen som oppstod i undervisningen, var at Spheroballene ikke hadde nok strøm. Dette førte til at elevene fikk mindre tid til programmeringen og dette kan være et hinder for utviklingen av elevenes forståelse. Det så ut til at elevene ble avbrutt da de var engasjerte i læringen fordi elevene kom med utsagn som: "*Vi har jo nesten klart det!*". I tillegg viser utsagn som: *"Åååå, vi skulle jo akkurat til å teste på banen. Da gidder jeg ikke mer."*, at elevene gir opp. Selv om læreren prøvde å få elevene til å gå sammen med andre grupper som hadde en Spheroball med mer strøm, ga elevene uttrykk for at de ikke gadd å fortsette. Dermed stoppet elevenes læringsprosess. En tilsvarende utfordring kom frem i studien til Tyrén et al. (2018) der det trekkes frem at teknologiske utfordringer, som manglende internettforbindelse, fører til frustrasjon for elevene og at læringstid går bort. I studien til Sentance og Csizmadia (2017) trekker det også frem tekniske problemer som en utfordring i arbeidet med programmering. På bakgrunn av vår studie, studien til Tyrén et al. (2018) og studien til Sentance og Csizmadia (2017) vil det være viktig at lærere er godt forberedt på de teknologiske utfordringene som kan oppstå og hvordan disse utfordringene kan forhindres.

6.2.5 Oppsummering av utfordring i programmering

Utfordringene knyttet til programmering i naturfag som kommer frem i denne studien er manglende kommunikasjon om den algoritmiske tenkeren, elevenes behov for veiledning, utfordringer med samarbeid og teknologiske utfordringer.

For at læreren skal veilede elevene underveis i undervisningen er det viktig at læreren er godt kjent med undervisningsopplegget slik at en er trygg i gjennomføringen. Dette krever god kommunikasjon mellom den som har planlagt undervisningen og de som skal gjennomføre undervisningen. Lærerens manglende kunnskap og erfaring med den algoritmiske tenkeren førte til manglende veiledning av strategier elevene kunne bruke. Dette skaper utfordringer fordi mange elever ikke vet hvilke strategier de skal bruke eller hvilke strategier som er effektive for å løse utfordringer de møter i programmeringen. Den algoritmiske tenkeren kan bidra til å gjøre kunnskapen mer håndterlig for læreren. Likevel så vi at Utdanningsdirektoratet (2019) sin beskrivelse av den algoritmiske tenkeren er lite utdypende

og det kan derfor være utfordrende for læreren å forstå hvordan en kan kjenne igjen strategier som elever bruker og å veilede elever i bruk av strategier i undervisning.

Det er viktig at læreren er bevisst på hva som er hensikten med samarbeid og hva som forventes av elevene slik at læreren kan formidle dette til elevene og på denne måten sørge for et hensiktsmessig samarbeid der elevene lærer av hverandre. Ved bruk av jevnaldrende veileder som samarbeidsstrategi kan dette gjøres ved å vektlegge kommunikasjon og det å lytte til hverandres forslag. Parprogrammering er en annen samarbeidsstrategi som kan benyttes. Dette krever imidlertid at elevene forstår hva de ulike rollene innebærer. Videre krever samarbeidsstrategiene at lærere følger opp elevene underveis i samarbeidet.

I tillegg vil det være hensiktsmessig at lærere har satt seg godt inn i hvilke teknologiske utfordringer som kan oppstå, og hva som eventuelt kan gjøres i møte med disse utfordringene, for å unngå at elevens læringsprosess og engasjement svekkes.

7 Oppsummering og konklusjon

Hensikten med denne studien var å undersøke hvordan elever på mellomtrinnet arbeider med programmering i naturfagundervisning, og utfordringer som kan oppstå i planlegging og gjennomføring av slike undervisningsopplegg.

For å undersøke dette har vi formulert to forskningsspørsmål:

- Hvilke kjennetegn på algoritmisk tenkning kommer til utrykk i elevers arbeid med programmering i naturfag på mellomtrinnet?
- Hvilke utfordringer kommer frem i et undervisningsopplegg med programmering i naturfag på mellomtrinnet?

Studiens resultater viser at alle strategiene fra den algoritmiske tenkeren kom til syne i elevenes arbeid. Samtidig så vi flere utfordringer knyttet til disse strategiene. Under vil vi sammenfatte de viktigste funnene som er drøftet i diskusjonen, samt trekke frem hvordan kunnskap rundt dette kan bidra til å utvikle programmeringsundervisning i naturfag.

Planleggeren hadde lagt til rette for strategiene holde ut og feilsøking i introduksjonsøkten. Dette ble gjort for å lære elevene at programmering i stor grad innebærer det å se etter feil og lære av feilene. Videre skulle dette bidra til at elevene klarte å stå i problemet uten å gi opp. Dette samsvarer med Barefoot computing (u.å-k) som sier at feilsøking påvirker elevenes utholdenhet. Under observasjonen kom det frem at mange elever brukte strategiene de hadde lært i introduksjonsøkten flittig. Dermed kan det å lære elever strategier fra den algoritmiske tenkeren veilede dem gjennom programmeringen ved at de vet hvilke strategier de kan benytte seg av når de møter utfordringer.

For å lære elever strategier fra den algoritmiske tenkeren krever det at læreren er kjent med strategiene selv. I vår studie kom det frem at planleggeren hadde studert programmering og var godt kjent med den algoritmiske tenkeren. Å bruke en lærer med mye kunnskap kan være hensiktsmessig i planleggingen av undervisningen slik at undervisningen er basert på både programmeringsfaglige og programmeringsdidaktiske valg (Sentance et al., 2017). Vi så imidlertid at det var manglende kommunikasjon om den algoritmiske tenkeren mellom planleggeren og lærerne som underviste. Dette kan føre til at oppleggets programmeringsfaglige og programmeringsdidaktiske hensikt går tapt. I tillegg så den manglende kommunikasjonen ut til å påvirke lærernes evne til å videreføre kunnskapen til elevene. Dette understreker viktigheten av kommunikasjon mellom ressurspersonen og læreren.

I undervisningen i to av klassene ble det lagt opp til samarbeid ved at gruppene ble satt sammen av elever med ulik erfaring slik at de skulle lære av hverandre. Dette kom frem i flere av gruppene der samarbeidet bestod av elever som lyttet til hverandres forslag, brukte

hverandres kunnskap og kommuniserte godt. Vi så likevel grupper der elevene ikke var like aktive i utviklingen av algoritmen. Dette kan komme av at elevene ikke har blitt informert om hvordan de skal samarbeide, som ifølge Ludvigsen et al. (2015) er grunnleggende for et godt samarbeid. Et tiltak lærere kan gjøre for å tilrettelegge for samarbeid i programmering er å introdusere elever for parprogrammering. I et slik samarbeid får elevene roller som innebærer ulike ansvar, kalt sjåfør og kartleser (Barefoot computing, u.å-c; Sevik, 2016). På den måten har elevene hver sin oppgave i samarbeidet og er avhengig av hverandre når de programmerer.

Læreren i de to klassene der elevene samarbeidet la til rette for dekomposisjon ved å gi elevene beskjed om at de kunne bruke to bord til å utvikle en kode. Ifølge Anderson (2016) og Barefoot computing (u.å-g) bidrar dekomposisjon til å gjøre programmering mer håndterlig for elever og dette så vi også i vår studie. Vi så at flere av elevene som ikke dekomponerte heller ikke klarte å utvikle en fullstendig algoritme. Dette gir inntrykk av at samarbeid og det å dele opp banen kan påvirke elevenes evne til å fullføre løsningen.

Imidlertid er det ikke nødvendigvis det at elevene lager en fullstendig løsning som viser at elevene har forstått hva, hvordan og hvorfor de programmerer. Vi så at de elevene som kopierte andres løsninger uten å evaluere løsningen slet med å løse de neste utfordringene de møtte på i programmeringen. Derfor bør lærere legge vekt på evaluering for å utvikle elevenes forståelse for programmering. Dette kan bli gjort ved at lærere stiller spørsmål for å få elever til å reflektere rundt egen løsning, samt legge opp til evaluering på slutten av undervisningsøkten.

I tillegg til utfordringer knyttet til den algoritmiske tenkeren og lærerens kunnskap om programmering, kom det frem en teknologisk utfordring. Dette gikk ut på at Spheroballene var utladet, noe som så ut til å påvirke elevenes utholdenhet i programmeringen. For å unngå slike utfordringer er det viktig at lærere vurderer hvilke teknologiske utfordringer som kan oppstå og hvordan de kan forhindres.

Denne studien viser hvordan strategier fra den algoritmiske tenkeren og utfordringer kan se ut i et undervisningsopplegg i programmering i naturfag. Det at lærere setter seg inn i den algoritmiske tenkeren og hvilke utfordringer elevene kan møte på er hensiktsmessig for å få til god programmeringsundervisning. Vi så imidlertid at informasjonen om den algoritmiske tenkeren på Utdanningsdirektoratet (2019) sine nettsider var lite utfyllende og for å få en dypere forståelse av de ulike strategiene var det nødvendig med påfyll fra den opprinnelige modellen til Barefoot computing (u.å-d). Vår studie kan bidra til å vise hvordan algoritmisk tenkning kan se ut i norske klasserom, og hjelpe lærere til å kjenne igjen de ulike strategiene i egen undervisning. Ved å kjenne igjen og forstå de ulike strategiene, samt utfordringer som kan oppstå, kan lærere bruke dette til å utvikle programmeringsundervisning i naturfag.

7.1 Studiens begrensninger og veien videre

I denne studien har det blitt samlet observasjonsdata fra fire ulike klasser. Ettersom klassene var nokså store, hadde vi ikke mulighet til å observere alle elevene i hver klasse. Vi måtte rette oppmerksomheten vår mot noen få elever for å få mest mulig detaljerte observasjoner. Derfor har vi i senere tid reflektert rundt at det å observere flere klasser kunne ha ført til en større datamengde og vi kan ikke utelukke at enda flere informanter ville nyansert funnene.

Intervjudataene er innhentet fra to lærere. I tillegg til å intervjue en lærer som underviste i opplegget var det interessant å inkludere et intervju av læreren som hadde planlagt undervisningen ettersom dette kunne gi oss en økt forståelse av hensikten med undervisningsopplegget. Imidlertid kunne det å inkludere flere lærere i studien bidratt til å gi oss et dypere innblikk i læreres forståelse av programmering, den algoritmiske tenkeren og utfordringer de møter på i naturfagundervisning i programmering. Ettersom vi i denne studien ønsket å gjøre intervjuer knyttet til undervisningen vi observerte var det naturlig å intervjue de utvalgte lærerne.

Videre hadde det vært interessant å intervjue elever for å få et innblikk i deres opplevelse rundt bruken av strategier og utfordringer elever møter på i programmering i naturfag. Undervisningen skal lages for elevene og derfor vil informasjon om hva elevene mener er krevende viktig kunnskap som kan bidra til utvikling av undervisningsopplegg.

Ettersom strategiene i den algoritmiske tenkeren ikke er knyttet til et spesifikt program eller undervisningsopplegg, kan disse strategiene overføres til nye programmeringssituasjoner. Derfor kan det være interessant å følge elever videre fra et undervisningsopplegg med programmering i naturfag til et annet for å undersøke om og på hvilke måter elever tar med seg strategiene fra den algoritmiske tenkeren videre og eventuelt hvorfor de gjør det eller ikke.

Vedlegg 1 - Observasjonsskjema

Klasse: …… dato: ……

Observasjoner	Egne tolkninger

Vedlegg 2 – Intervjuguide til lærer

Intervjuguide til lærer

"Programmering i naturfag"

OsloMet – storbyuniversitetet

Innledning

Hei, og velkommen til denne uformelle samtalen om bruk av programmering i naturfagundervisningen.

Som du kjenner til, ettersom du har samtykket til å delta, ønsker vi å undersøke hvordan programmering blir brukt i naturfag ved å se på lærerens tilrettelegging, elevenes utfordringer og strategiene som blir benyttet.

Vi ønsker å intervjue deg om dine og elevenes tidligere erfaringer med programmering og erfaringene du gjorde deg i denne undervisningen. Det er frivillig å delta i intervjuet, og du kan når som helst, uten begrunnelse, trekke ditt samtykke.

Vi vil gjøre et lydopptak av intervjuet for å sørge for at alle detaljer kommer med i transkripsjonen. Opptaket skal kun brukes til forsknings- og dokumentasjonsformål. Det vil blir behandlet konfidensielt som vil si at du ikke kan identifiseres med navn eller kunne gjenkjennes på annen måte i masteroppgaven.

Ønsker du å trekke deg? Hvis ikke, starter vi intervjuet og lydopptaket.

– Start opptaket –

Om informanten

Hvilken utdannelse har du?

Hvilke fag har du fra lærerutdanningen?

Hvor mange år har du jobbet som lærer?

Tilrettelegging

Har elevene hatt en forberedelses økt til denne programmeringsundervisningen, og hva var eventuelt fokuset i forberedelsesøkten?

- Tilleggsspørsmål: Men det var med blokkprogrammering, det var ikke sånn at de tegnet for å få den til å kjøre fremover?

Hva tenkte du at målet med undervisningen var?

Hvor stor grad av frihet fikk elevene til å løse oppgaven?

- Måtte du gjøre tilpasninger underveis, eventuelt hvilke?

Hvilke strategier var det ment at elevene skulle bruke?

Hvilke strategier mener du at elevene brukte i undervisningen?

Hvordan ble gruppene satt sammen?

Programmering

Ut ifra dine nåværende kunnskaper, hva legger du i begrepet programmering?

Hvilken kompetanse har du i programmering?

Hvilke tanker har du rundt det å inkludere programmering i naturfag?

Hvilke erfaringer har du med programmering i klasserommet?

Opplever du noen utfordringer når du underviser i programmering i naturfag, eventuelt hvilke?

Hvordan har elevene jobbet med programmering tidligere?

- Når og hvordan?
- Tilleggsspørsmål: når var det dere hadde om koding? Har alt vært i 7.klasse?

Har elevene fått noe trening i hvordan de skal møte utforinger i programmering?

- Opplevde du at elevene benyttet seg av det de hadde lært og på hvilken måte?
- Oppklaringsspørsmål for å tydeliggjøre: Altså du nevner at de har lært å feilsøke for eksempel, opplevde du at de benyttet seg av den strategien i denne programmeringsundervisningen?

Algoritmisk tenkning

Hva legger du i begrepet algoritmisk tenkning?

Hvordan tenker du algoritmisk tenkning var en del av undervisningen du nettopp gjennomførte?

Har du brukt den algoritmiske tenkeren i planleggingen og gjennomføring av undervisningen?

På hvilken måte er den algoritmiske tenkeren benyttet i planleggingen av undervisningen?

Hvordan kom den algoritmiske tenkeren til syne i undervisningen?

Er det noen deler av den algoritmiske tenkeren det er lagt større fokus på, eventuelt hvilke?

- Hvordan tilrettela du for dette underveis?

Var det noen utfordringer knyttet til algoritmisk tenkning du hadde tenkt ut på forhånd at elevene kom til å møte på?

Tilleggsspørsmål: Men hva er det de er bedre til?

Opplevde du at elevene møtte på utfordringer knyttet til den algoritmiske tenkeren, eventuelt hvilke?

- Hvordan møtte du disse utfordringene?

Har elevene fått trening i hvordan de skal møte utfordringer i programmering?

Avslutningsspørsmål

Er det noe du ønsker å legge til?

Vedlegg 3 – Intervjuguide til planlegger

Intervjuguide til planlegger

"Programmering i naturfag"

OsloMet – storbyuniversitetet

Innledning

Hei, og velkommen til denne uformelle samtalen om bruk av programmering i naturfagundervisningen.

Som du kjenner til, ettersom du har samtykket til å delta, ønsker vi å undersøke hvordan programmering blir brukt i naturfag ved å se på lærerens tilrettelegging, elevenes utfordringer og strategiene som blir benyttet.

Vi ønsker å intervjue deg om dine og elevenes tidligere erfaringer med programmering og erfaringene du gjorde deg i denne undervisningen. Det er frivillig å delta i intervjuet, og du kan når som helst, uten begrunnelse, trekke ditt samtykke.

Vi vil gjøre et lydopptak av intervjuet for å sørge for at alle detaljer kommer med i transkripsjonen. Opptaket skal kun brukes til forsknings- og dokumentasjonsformål. Det vil blir behandlet konfidensielt som vil si at du ikke kan identifiseres med navn eller kunne gjenkjennes på annen måte i masteroppgaven.

Ønsker du å trekke deg? Hvis ikke, starter vi intervjuet og lydopptaket.

– Start opptaket –

Om informanten

Hvilken utdannelse har du?

Hvilke fag har du fra lærerutdanningen?

Hvor mange år har du jobbet som lærer?

Programmering

Hva legger du i begrepet programmering?

Hvilken kompetanse har du i programmering?

Hvilke tanker har du rundt det å inkludere programmering i naturfag?

Hvilke erfaringer har du med programmering i naturfagundervisning?

Opplever du noen utfordringer når du underviser i programmering i naturfag, eventuelt hvilke?

- Tilleggsspørsmål: Hva tenker du angående det du sier med gruppesammensetninger? Hva må du tenke på der?

Planlegging

Stemmer det at du har vært med på å lage undervisningsopplegget i programmering med Spheroballene?
Har du planlagt undervisningen alene eller i samarbeid med lærerne på 7.trinn?

- Hvis alene: Hvordan har du formidlet opplegget til de andre lærerne?
- Hvis sammen: På hvilken måte har dere samarbeidet?

Hva tenkte du målet med undervisningen var?

Algoritmisk tenkning

Hva legger du i begrepet algoritmisk tenkning?
Stemmer det at du har brukt den algoritmiske tenkeren i planleggingen av undervisningen?

På hvilken måte er den algoritmiske tenkeren benyttet i planleggingen av undervisningen?

Hvilke strategier fra den algoritmiske tenkeren var det ment at elevene skulle bruke?

- Tilleggskommentar: Vi har lagt sammen arbeidsmåtene og nøkkelbegrepene fra modellen og kalt de strategier.

Er det noen deler av den algoritmiske tenkeren det er lagt større fokus på, eventuelt hvilke?

Var det noen utfordringer knyttet til algoritmisk tenkning du hadde tenkt ut på forhånd at elevene kom til å møte på?

- Hvordan påvirket dette planleggingen?

Er det noe du opplever som utfordrende med å bruke den algoritmiske tenkeren i naturfag?

Avslutningsspørsmål

Er det noe du ønsker å legge til?

Vedlegg 4 – Medforfattererklæring

OSLOMET

Medforfattererklæring

Om to eller tre studenter gjennomfører og/eller skriver masteroppgaven sammen, skal det legges ved et medforfatterklæring, jf. emneplan MGMO5900:

"For studenter som velger å gjennomføre masteroppgaven som gruppearbeid, skal det gå tydelig fram i egen redegjørelse hvordan arbeidet er fordelt, og hvordan hver enkelt oppfyller kravet om selvstendig vitenskapelig arbeid. Her benyttes en medforfattererklæring som begge eller alle tre parter signerer."

Masteroppgavens tittel:

Programmering i naturfag: den algoritmiske tenkeren og utfordringer på veien

Redegjørelse på hvordan arbeidet er fordelt, og hvordan den enkelte oppfyller kravet om selvstendig vitenskapelig arbeid:

I arbeidet med å søke etter relevant teori har vi fordelt vitenskapelige artikler, men vi har lest igjennom og vurdert hverandres tekst, og kommet til enighet om hva som var hensiktsmessig å inkludere i oppgaven. Planlegging og gjennomføring av datainnsamling er utført av begge studenter. Vi har analysert dataen i samarbeid og reflektert rundt funn. Noe av drøftingen er gjort i samarbeid, mens andre deler er fordelt mellom oss. Likevel har vi lest igjennom hverandres drøftinger og kommet med forslag til endringer.

Undertegnede bekrefter å ha bidratt til følgende deler av masteroppgavearbeidet:

Prosjektskisse, idé og tema for masteroppgaven	Ja
Praktisk gjennomføring av studien for eksempel innhenting av data	Ja
Analyse, drøfting og tolkning av resultatene	Ja

Undertegnede har lest og godkjent den innsendte versjonen av masteroppgaven

(sted)	(dato)	(signatur)
OSLO	13.05.22	Cathinka M. Ebert
OSLO	13.05.22	Helena Dragland

Vedlegg 5 – Vurdering fra NSD

Vurdering

Referansenummer

813074

Prosjekttittel

Programmering i naturfag

Behandlingsansvarlig institusjon

OsloMet – storbyuniversitetet / Fakultet for lærerutdanning og internasjonale studier / Institutt for grunnskole- og faglærerutdanning

Prosjektansvarlig (vitenskapelig ansatt/veileder eller stipendiat)

Per Øyvind Sollid, peroso@oslomet.no

Type prosjekt

Studentprosjekt, masterstudium

Kontaktinformasjon, student

Cathinka Mønsted Ebert, s325195@oslomet.no

Prosjektperiode

01.01.2022 - 01.07.2022

Vurdering (1)

20.01.2022 - Vurdert

Det er vår vurdering at behandlingen av personopplysninger i prosjektet vil være i samsvar med personvernlovgivningen så fremt den gjennomføres i tråd med det som er dokumentert i meldeskjemaet med vedlegg den 20.01.2022. Behandlingen kan starte.

TYPE OPPLYSNINGER OG VARIGHET Prosjektet vil behandle alminnelige kategorier av personopplysninger frem til 01.07.2022.

LOVLIG GRUNNLAG Prosjektet vil innhente samtykke fra de registrerte til behandlingen av personopplysninger. Vår vurdering er at prosjektet legger opp til et samtykke i samsvar med kravene i art. 4 og 7, ved at det er en frivillig, spesifikk, informert og utvetydig bekreftelse som kan dokumenteres, og som den registrerte kan trekke tilbake.

Lovlig grunnlag for behandlingen vil dermed være den registrertes samtykke, jf. personvernforordningen art. 6 nr. 1 bokstav a.

PERSONVERNPRINSIPPER Personverntjenester vurderer at den planlagte behandlingen av personopplysninger vil følge prinsippene i personvernforordningen om:

• lovlighet, rettferdighet og åpenhet (art. 5.1 a), ved at de registrerte får tilfredsstillende informasjon om og samtykker til behandlingen
• formålsbegrensning (art. 5.1 b), ved at personopplysninger samles inn for spesifikke, uttrykkelig angitte og berettigede formål, og ikke behandles til nye, uforenlige formål
• dataminimering (art. 5.1 c), ved at det kun behandles opplysninger som er adekvate, relevante og nødvendige for formålet med prosjektet
• lagringsbegrensning (art. 5.1 e), ved at personopplysningene ikke lagres lengre enn nødvendig for å oppfylle formålet DE

REGISTRERTES RETTIGHETER Så lenge de registrerte kan identifiseres i datamaterialet vil de ha følgende rettigheter: innsyn (art. 15), retting (art. 16), sletting (art. 17), begrensning (art. 18), og dataportabilitet (art. 20).

Personverntjenester vurderer at informasjonen om behandlingen som de registrerte vil motta oppfyller lovens krav til form og innhold, jf. art. 12.1 og art. 13. Vi minner om at hvis en registrert tar kontakt om sine rettigheter, har behandlingsansvarlig institusjon plikt til å svare innen en måned.

FØLG DIN INSTITUSJONS RETNINGSLINJER Personverntjenester legger til grunn at behandlingen oppfyller kravene i personvernforordningen om riktighet (art. 5.1 d), integritet og konfidensialitet (art. 5.1. f) og sikkerhet (art. 32).

Nettskjema og Microsoft er databehandler i prosjektet. Personverntjenester legger til grunn at behandlingen oppfyller kravene til bruk av databehandler, jf. art 28 og 29.

For å forsikre dere om at kravene oppfylles, må dere følge interne retningslinjer og/eller rådføre dere med behandlingsansvarlig institusjon.

MELD VESENTLIGE ENDRINGER Dersom det skjer vesentlige endringer i behandlingen av personopplysninger, kan det være nødvendig å melde dette til oss ved å oppdatere meldeskjemaet. Før du melder inn en endring, oppfordrer vi deg til å lese om hvilke type endringer det er nødvendig å melde:
https://www.nsd.no/personverntjenester/fylle-ut-meldeskjema-for-personopplysninger/melde-endringer-i-meldeskjema
Du må vente på svar fra oss før endringen gjennomføres.

OPPFØLGING AV PROSJEKTET Personverntjenester vil følge opp ved planlagt avslutning for å avklare om behandlingen av personopplysningene er avsluttet. Lykke til med prosjektet!

Referanseliste

Abbot, S. (2016). 21st century skills. The Glossary of Education Reform. 2014. I. https://www.edglossary.org/21st-century-skills/

Anderson, N. D. (2016). A call for computational thinking in undergraduate psychology. *Psychology Learning & Teaching*, *15*(3), 226-234.

Barefoot computing. (u.å-a). *Abstraction*. Hentet 01.03.22 fra https://www.barefootcomputing.org/concepts-and-approaches/abstraction

Barefoot computing. (u.å-b). *Algorithms*. Hentet 01.02.22 fra https://www.barefootcomputing.org/concepts-and-approaches/algorithms

Barefoot computing. (u.å-c). *Collaborating*. Hentet 04.03.22 fra https://www.barefootcomputing.org/concepts-and-approaches/collaborating

Barefoot computing. (u.å-d). *Computational Thinking Concepts and Approaches*. Hentet 01.02.22 fra https://www.barefootcomputing.org/concept-approaches/computational-thinking-concepts-and-approaches

Barefoot computing. (u.å-e). *Creating*. Hentet 02.03.22 fra https://www.barefootcomputing.org/concepts-and-approaches/creating

Barefoot computing. (u.å-f). *Debugging*. Hentet 03.03.22 fra https://www.barefootcomputing.org/concepts-and-approaches/debugging

Barefoot computing. (u.å-g). *Decomposition*. Hentet 01.03.22 fra https://www.barefootcomputing.org/concepts-and-approaches/decomposition

Barefoot computing. (u.å-h). *Evaluation*. Hentet 02.03.22 fra https://www.barefootcomputing.org/concepts-and-approaches/evaluation

Barefoot computing. (u.å-i). *Logic*. Hentet 01.03.22 fra https://www.barefootcomputing.org/concepts-and-approaches/logic

Barefoot computing. (u.å-j). *Patterns*. Hentet 01.03.22 fra https://www.barefootcomputing.org/concepts-and-approaches/patterns

Barefoot computing. (u.å-k). *Persevering*. Hentet 03.03.22 fra https://www.barefootcomputing.org/concepts-and-approaches/persevering

Barefoot computing. (u.å-l). *Tinkering*. Hentet 02.03.22 fra https://www.barefootcomputing.org/concepts-and-approaches/tinkering

Barr, D., Harrison, J. & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *Learning & Leading with Technology*, *38*(6), 20-23.

Barr, V. & Stephenson, C. (2011). Bringing computational thinking to K-12: and what is the role of the computer science education community? *Acm Inroads*, *2*(1), 48-54.

Beck, L., Chizhik, A. & McElroy, A. (2005). Cooperative learning techniques in CS1: design and experimental evaluation. I *Technical Symposium on Computer Science Education* (s. 470-474). ACM.

Begel, A. (1996). LogoBlocks: A graphical programming language for interacting with the world. *Electrical Engineering and Computer Science Department, MIT, Boston, MA*, 62-64.

Bergkastet, I., Duesund, C., Westvig, T. S. & Keeping, D. (2019). *Trygt og godt skolemiljø : inkludering faglig og sosialt* (2. utg. utg.). Gyldendal.

Bers, M. U., Flannery, L., Kazakoff, E. R. & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145-157.

Blatchford, P., Kutnick, P., Baines, E. & Galton, M. (2003). Toward a social pedagogy of classroom group work. *International journal of educational research*, *39*(1-2), 153-172.

Brennan, K. & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada,

Chevalier, M., Giang, C., Piatti, A. & Mondada, F. (2020). Fostering computational thinking through educational robotics: a model for creative computational problem solving. *INTERNATIONAL JOURNAL OF STEM EDUCATION*, *7*(1). https://www.scopus.com/inward/record.uri?eid=2-s2.0-85088860972&doi=10.1186%2fs40594-020-00238-z&partnerID=40&md5=47cfe83a7a927cc5f849faa0344acb2d

Christoffersen, L. & Johannessen, A. (2012). *Forskningsmetode for lærerutdanningene*. Abstrakt forl.

Cohen, L., Manion, L. & Morrison, K. R. B. (2018). *Research methods in education* (8th ed. utg.). Routledge.

Coulter, B., Lee, I. & Martin, F. (2010). Computational thinking for youth. I. Citeseer.

Council, N. R. (2010). *Report of a workshop on the scope and nature of computational thinking*. National Academies Press.

CSTA Standards Task Force. (2016). *[INTERIM] CSTA K-12 Computer Science Standards*. CSTA. http://steamcurriculum.weebly.com/uploads/2/5/5/8/25586003/csta_interim_standards_final_07222.pdf?fbclid=IwAR0ciJKQtZst5mTyU6OAAJGCiVdSc8BrKxcaWtMNOU_M11YDHEAEiuOp7WA

Dalland, O. (2012). *Metode og oppgaveskriving for studenter* (5. utg. utg.). Gyldendal akademisk.

Dalland, O. & Keeping, D. (2020). *Metode og oppgaveskriving* (7. utgave. utg.). Gyldendal.

Dong, Y., Marwan, S., Catete, V., Price, T. & Barnes, T. (2019). Defining tinkering behavior in open-ended block-based programming assignments. Proceedings of the 50th ACM Technical Symposium on Computer Science Education,

Dybå, T., Arisholm, E., Sjoberg, D. I., Hannay, J. E. & Shull, F. (2007). Are two heads better than one? On the effectiveness of pair programming. *IEEE software*, *24*(6), 12-15.

Eriksen, H. & Svanes, I. K. (2021). Kategorisering og koding i intervju- og observasjonsforkning IE. Andersson-Bakken & C. P. Dalland (Red.), *Metoder i klasseromsforskning* (s. 287-304). Universitetsforlaget.

Fangen, K. (2019). *Deltagende observasjon*. Fagbokforlaget.

Flowers, T. R. & Gossett, K. A. (2002). Teaching problem solving, computing, and information technology with robots. *Journal of Computing Sciences in Colleges, 17*(6), 45-55.

Flø, E. E. (2021). Programmering i LK20. *Tangenten: tidsskrift for matematikkundervisning, 32 (1)*, 3-9.

Gjevjon, E. R. (2019). Tema, problemstilling, hensikt, forskningsspørsmål, hypotese og mål – hva er hva? *Sykepleien forskning (Oslo)*, (79024), e-79024. https://doi.org/10.4220/Sykepleienf.2019.79024

Hazzan, O., Lapidot, T. & Ragonis, N. (2015). *Guide to Teaching Computer Science: An Activity-Based Approach*. Springer.

Hazzan, O., Lapidot, T. & Ragonis, N. (2011). Guide to teaching computer science: An activity-based approach. I. Springer Publishing Company, Incorporated.

Hertz-Lazarowitz, R. (2008). Beyond the Classroom and into the Community: The Role of the Teacher in Expanding the Pedagogy of Cooperation. I *The teacher's role in implementing cooperative learning in the classroom* (s. 38-55). Springer.

Hodges, S., Scott, J., Sentance, S., Miller, C., Villar, N., Schwiderski-Grosche, S., Hammil, K. & Johnston, S. (2013). . NET Gadgeteer: a new platform for K-12 computer science education. Proceeding of the 44th ACM technical symposium on Computer science education,

Hodges, S., Sentance, S., Finney, J. & Ball, T. (2020). Physical computing: A key element of modern computer science education. *Computer*, *53*(4), 20-30.

Hsieh, H.-F. & Shannon, S. E. (2005). Three Approaches to Qualitative Content Analysis. *Qual Health Res, 15*(9), 1277-1288. https://doi.org/10.1177/1049732305276687

Johnson, R. B. & Christensen, L. (2019). *Educational research: Quantitative, qualitative, and mixed approaches*. Sage publications.

Klette, K. (2003). Forskningstilnærming og datainnhentingsstrategier [Research approaches and data acquisition strategies]. *Klasserommets praksisformer etter Reform 97*, 21-36.

Kvale, S., Brinkmann, S., Anderssen, T. M. & Rygge, J. (2015). *Det kvalitative forskningsintervju* (3. utg. utg.). Gyldendal akademisk.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. & Werner, L. (2011). Computational thinking for youth in practice. *Acm Inroads*, *2*(1), 32-37.

Lincoln, Y. S. & Guba, E. G. (1986). But is it rigorous? Trustworthiness and authenticity in naturalistic evaluation. *New directions for program evaluation*, *1986*(30), 73-84.

Ludvigsen, S., Gundersen, E., Indregard, S., Bushra, I., Kleven, K., Korpås, T. & Skjørberg, S. (2015). Fremtidens skole: fornyelse av fag og kompetanser. *Norges offentlige utredninger (tidsskrift: online), Vol. NOU, 8.*

Maloney, J., Resnick, M., Rusk, N., Silverman, B. & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 1-15.

McDowell, C., Hanks, B. & Werner, L. (2003). Experimenting with pair programming in the classroom. Proceedings of the 8th annual conference on Innovation and technology in computer science education,

Medeiros, R. P., Ramalho, G. L. & Falcão, T. P. (2018). A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, *62*(2), 77-90. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8447543

Mulhall, A. (2003). In the field: notes on observation in qualitative research. *Journal of advanced nursing*, *41*(3), 306-313.

Muller, M. M. & Padberg, F. (2004). An empirical study about the feelgood factor in pair programming. 10th International Symposium on Software Metrics, 2004. Proceedings.,

Olsson, H., Sörensen, S. & Bureid, G. (2003). *Forskningsprosessen : kvalitative og kvantitative perspektiver*. Gyldendal akademisk.

Postholm, M. B. (2005). Observasjon som redskap i kvalitativ forskning på praksis. *Norsk pedagogisk tidskrift*, *89*(2), 146-158.

Price, T. W. & Barnes, T. (2015). Comparing textual and block interfaces in a novice programming environment. Proceedings of the eleventh annual international conference on international computing education research,

Przybylla, M. & Romeike, R. (2015). Key competences with physical computing. *KEYCIT 2014: key competencies in informatics and ICT*, *7*, 351.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J. & Silverman, B. (2009). *Communications of the ACM*, *52*(11), 60-67.

Sanne, A., Berge, O., Bungum, B., Jørgensen, E. C., Kluge, A., Kristensen, T. E., Mørken, K. M., Svorkmo, A.-G. & Voll, L. O. (2016). Teknologi og programmering for alle-En faggjennomgang med forslag til endringer i grunnopplæringen-august 2016.

Schulz, S. & Pinkwart, N. (2016). Towards Supporting Scientific Inquiry in Computer Science Education. Proceedings of the 11th Workshop in Primary and Secondary Computing Education,

Sentance, S. & Csizmadia, A. (2017). Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Education and Information Technologies*, *22*(2), 469-495.

Sentance, S., Waite, J., Yeomans, L. & MacLeod, E. (2017). Teaching with physical computing devices: the BBC micro: bit initiative. Proceedings of the 12th Workshop on Primary and Secondary Computing Education,

Sevik, K. (2016). Programmering i skolen. https://www.udir.no/globalassets/filer/programmering_i_skolen.pdf

Shute, V. J., Sun, C. & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142-158.

Skilbrei, M.-L. (2019). *Kvalitative metoder : planlegging, gjennomføring og etisk refleksjon* (1. utgave. utg.). Fagbokforlaget.

Sphero Inc. (2022). *Sphero BOLT Koding Robot*. Hentet 04.04.22 fra https://sphero.com/collections/all/products/sphero-bolt

Stenseth, B., Kaufmann, O. T. & Forsström, S. (2019). Programmering og matematikk. *Tangenten–tidsskrift for matematikkundervisning*, *30*(2), 7-12.

Säljö, R. (2001). Læring i praksis. *Et sosiokulturelt perspektiv. Oslo: JW Cappelens forlag*.

Sørby, S. A. & Angell, C. (2012). Undergraduate students' challenges with computational modelling in physics. *Nordic Studies in Science Education*, *8*(3), 283-296.

Taraldsen, L. H. & Myhra, K. S. (2019). Programmering med Spheroballer. *Tangenten: tidsskrift for matematikkundervisning*, *4*. http://www.caspar.no/tangenten/2019/Tangenten%204%202019%20Taraldsen%20Myhra.pdf

Tellefsen, C. W. (2021a). Realfaglig programmering I. Det utdanningsvitenskapelige fakultetet https://www.uv.uio.no/forskning/satsinger/fiks/kunnskapsbase/realfaglig-programmering/realfaglig-programmering---en-innledning/

Tellefsen, C. W. (2021b). Undervisning i realfaglig programmering I. Det utdanningsvitenskapelige fakultetet https://www.uv.uio.no/forskning/satsinger/fiks/kunnskapsbase/realfaglig-programmering/undervisning-i-realfaglig-programmering/

Tsai, C.-Y., Chang, C.-K. & Yang, Y.-F. (2015). *Cognitive load comparison of traditional and distributed pair programming on visual programming language* International Conference of Educational Innovation through Technology, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7446165

Tyrén, M., Carlborg, N., Heath, C. & Eriksson, E. (2018). Considerations and technical pitfalls for teaching computational thinking with BBC micro: Bit. Proceedings of the Conference on Creativity and Making in Education,

UiB. (2022). 8 viktige yrker for fremtiden Hentet 26.03.2022, fra https://www.uib.no/realfag/140960/8-viktige-yrker-fremtiden

Utanningsdiraktoratet. (2019). Hva er nytt i naturfag og naturfag samisk? Hentet 25.04.22, fra https://www.udir.no/laring-og-trivsel/lareplanverket/fagspesifikk-stotte/nytt-i-fagene/hva-er-nytt-i-naturfag-og-naturfag-samisk/?fbclid=IwAR3jL7BAygTm_ahtRxQygL886GJ-kzP0WBbDmGUTMx2TxHEdjmwFB_dE1P0

Utdanningsdirektoratet. (2013). *Lærerplan i naturfag* https://www.udir.no/kl06/NAT1-03/Hele/Grunnleggende_ferdigheter

Utdanningsdirektoratet. (2017). *Overordnet del - formålet med opplæringen* https://www.udir.no/lk20/overordnet-del/formalet-med-opplaringen/?lang=nob

Utdanningsdirektoratet. (2019). *Algoritmisk tenkning* https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Utdanningsdirektoratet. (2020a). *Naturfag (NAT01-04) Kjerneelementer* https://www.udir.no/lk20/nat01-04/om-faget/kjerneelementer

Utdanningsdirektoratet. (2020b). *Naturfag (NAT01-04) Kompetansemål og vurdering*. https://www.udir.no/lk20/nat01-04/kompetansemaal-og-vurdering/kv79

Utdanningsdirektoratet. (2020c). *Naturfag (NAT01-04) Kompetansemål og vurdering* https://www.udir.no/lk20/nat01-04/kompetansemaal-og-vurdering/kv78

Voogt, J., Fisser, P., Good, J., Mishra, P. & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, *20*(4), 715-728.

Wassermann, S. & Haukeland, E. (2001). *Lekende læring : lek på alvor i klasserommet : styrking av barns mestringsevne gjennom praktiske læringsaktiviteter*. Høyskoleforl.

Weintrop, D. & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 1-25.

Wing, J. (2017). Computational thinking's influence on research and education for all. *Italian Journal of Educational Technology*, *25*(2), 7-14.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717-3725.

Yin, R. K. (2018). *Case study research and applications : design and methods* (Sixth Edition. utg.). SAGE.

Aakervik, A. O., Haugaløkken, O. K., Johnson, R. T. & Johnson, D. W. (2006). *Samarbeid i skolen : pedagogisk utvikling - samspill mellom mennesker* (4. rev. utg. utg.). Pedagogisk psykologisk forl.