MASTEROPPGAVE

Emnekode: NAT5003

Navn: Trine Gundersen

Algoritmisk tenkning gjennom programmeringsundervisning i naturfag – En praktisk studie med programmering.

Computational thinking through educational programming in science - A practical study with programming.

Dato: 15.11.2023

Totalt antall sider: 71

NORD universitet

www.nord.no

Forord

Jeg vil starte med å takke foreldre, søsken, familie, venner, medstudenter og kolleger for all støtte og tålmodighet gjennom det som har vært den mest utfordrende, interessante, stressende og emosjonelle prosessen i mitt liv. Jeg er veldig takknemlig for deres kontinuerlige og motiverende samtaler og hjelp.

Jeg ønsker også å takke min veileder Knut Moksnes for hans gode veiledninger, hjelp og tilbakemeldinger i løpet av denne skriveprosessen.

Videre ønsker jeg å gi en stor takk til skolen, lærerne og de tre klassene som var villige til å gi av sin tid, og til å la meg komme for å utføre forskningen min. Det er noe jeg setter veldig stor pris på.

Til slutt vil jeg takke mine nærmeste medstudenter for god støtte, hjelp og gode samtaler under en utfordrende prosess. Takk til Maria, Alexander, Erlend, Linn, Tomine og Nora. Dere har vært et godt støtteapparat.

Sammendrag

Hensikten med denne studien var å finne ut om elevers aktivitet preges av algoritmisk tenking ved løsning av programmeringsoppgaver. I fagfornyelsen (LK20) ble det et større fokus på innføring av programmering i skolen, og dette førte til at Utdanningsdirektoratet utviklet en modell for algoritmisk tenking, som har fått navnet den algoritmiske tenkeren. Modellen beskriver seks nøkkelbegreper og fem arbeidsmåter for løsning av programmeringsoppgaver. Videre ble det en nasjonal satsing hvor det ble delt ut klassesett med super:bit til alle sjetteklasser i Norge. Studien har en kvalitativ tilnærming, hvor observasjon gjennom video-opptak, var datainnsamlingsmetode. Ved analyse av datamaterialet ble det gjort en tematisk analyse. Resultatet viser at elevene preges både av nøkkelbegreper og arbeidsmåter innenfor den algoritmiske tenkeren.

Abstract

The purpose of this study was to determine whether students' activity is influenced by computational thinking when solving programming tasks. The newly reviewed national curriculum (LK20) requires an increased focus on the implementation of programming in schools, and this led to the Directorate of Education developing a model for the computational thinker, which describes six key terms and five approaches to solving programming tasks. Further, a national initiative began in which all Norwegian sixth graders were given class sets of super:bit. This study utilizes a qualitative approach where observation of video-recordings was the primary mean of data gathering. When analyzing the data material, a thematic analysis was conducted. The results show that the students were influenced by both key terms and approaches within the computational thinker.

Innholdsfortegnelse

Figurer:

Tabeller:

1. Innledning

Algoritmisk tenking og programmering er et interessant tema, som også er relevant i norsk skole i dag. Samfunnet vårt utvikler seg bare mer innenfor teknologi, og derfor er det viktig at skolene klarer å henge med når det kommer til den teknologiske utviklingen rundt oss.

I Meld. St. 28 skrev Kunnskapsdepartementet: «Fremtiden for kunnskapsnasjonen Norge skapes i barnehage og grunnopplæringen» (2015-2016, s. 5). Stortingsmeldingen gikk videre ut på at samfunnet var i en fornyelsesfase, og derfor måtte også skolen fornyes i tråd med samfunnet. Et svar til Meld. St. 28 skrev Trond Giske m.fl. (2016) at elever som går på skolen nå, skal være i arbeidslivet 60 år frem i tid. Med tanke på samfunnets digitale utvikling, vil det stilles andre krav til framtidige arbeidstakere enn det er i dag. Derfor mener Giske m.fl. (2016) at det stiller krav og forventninger til skolen om både hva og hvordan de skal lære, arbeide og forholde seg til kunnskap. Kunnskapsdepartementet (2017-2020) skriver at samfunnet trenger flere som er spesialiserte og har bedre generell IKT-kompetanse for at det framtidige arbeidslivet skal gå rundt. Begge partene mener at for å få utviklet den digitale kompetansen som trengs framover, må kompetanseutviklingen starte i skolen.

Dette endte opp som et forslag om fagfornyelsen, og senere til kunnskapsløftet 2020. I de nye læreplanene som ble produsert legger man godt merke til at problemløsing, generelt over alle trinn, har fått et stort fokusområde. I den sammenheng introduserte Utdanningsdirektoratet en problemløsningsmetode de valgte å kalle «Den algoritmiske tenkeren» (Utdanningsdirektoratet, 2019). Det er en metode som inneholder seks forskjellige nøkkelbegreper, som egentlig beskriver en form for tenkemåte for den algoritmiske tenkeren ved problemløsing. «Den algoritmiske tenkeren» kan derfor brukes som en mal til hvordan elever kan utvikle en bedre digital kompetanse og en forståelse for hvordan teknologi fungerer.

Kunnskapsdepartementet (2017-2020) skriver at elever må lære seg algoritmiske tenkemåter og prosesser i skolen, for å kunne vurdere om et problem kan løses ved hjelp av en datamaskin. Kunnskapsdepartementet ønsker også at elevene skal lære seg programmering i skolen.

Formålet i denne studien er å forske på elevers bruk av algoritmisk tenking ved problemløsning av programmeringsoppgaver. Hensikten er å se på hvordan et undervisningsopplegg kan stimulere til bruk av algoritmiske tenkemåter ved løsning av programmeringsoppgaver.

1.1 Problemstilling

På bakgrunn av det som ble skrevet innledningsvis er følgende problemstilling definert:

På hvilken måte preges elevers aktivitet av algoritmisk tenking ved løsning av programmeringsoppgaver?

1.2 Forskningsspørsmål

For å belyse denne problemstillingen ble det utarbeidet to forskningsspørsmål:

1. Hvilke av de seks nøkkelbegrepene bruker elever under oppgaveløsning i programmering?
2. Hvilke av de fem arbeidsmetodene bruker elever under oppgaveløsning i programmering?

2. Teori

2.1 Fagfornyelsen

«En godt utdannet og omstillingsdyktig arbeidsstyrke, vil bidra til å lette omstillingskostnadene i økonomien. Det stiller krav til fornyelse, også i utdanningssystemet.», skrev Kunnskapsdepartementet (Meld. St. 28, 2015-2016, s. 6). Det var en av de første begrunnelsene som en finner i innledningen, som leder opp mot fagfornyelsen. Roten til det som skulle føre til fagfornyelsen startet med et utvalg, med professor Sten Ludvigsen i spissen, som skulle vurdere grunnopplæringens fag opp mot krav til kompetanse i et framtidig samfunns- og arbeidsliv (Meld. St. 28, 2015-2016). I NOU 2014:7 (2014) kom Ludvigsenutvalget inn på at teknologi kom til å bli en nødvendig kompetanse for det 21. århundre. De skriver:

> «I dag bruker et overveiende flertall av de fleste aldersgrupper internett i løpet av et døgn. Digital teknologi har hatt innvirkning på de fleste sektorer i samfunnet. … Endringshyppigheten på IKT-området understreker at skolen må være i stand til å forandre seg og legge til rette for kontinuerlig læring i deltakende prosesser.» (NOU 2014:7, 2014, s. 113)

I dette sitatet kom de inn på at teknologien var i utvikling, og at den ble utviklet fort, noe den fortsatt gjør i dag. Derfor mener de at det er viktig at skolene klarer å henge med i denne teknologiske utviklingen.

I NOU 2015:8 (2015) presenterer Ludvigsenutvalget fire forskjellige fagområder, hvor et er «matematikk, naturfag og teknologi». De mente at fagfornyelsen burde skje gjennom de fagområdene, og ikke direkte i enkelte fag. Ludvigsenutvalget anbefalte et bredt kompetansebegrep som omfattet både tenkning, praktiske ferdigheter, sosial og emosjonell

læring og utvikling, og at dette ble reflektert i målene i fagene i grunnopplæringen (Meld. St. 28, 2015-2016). Kunnskapsdepartementet (2015-2016, s. 6), skriver i Meld. St. 28 at: «Samfunnet er i endring og endringstakten øker på mange samfunnsområder. Store deler av arbeidslivet preges sterkt av den teknologiske utviklingen.». Dette nevnes flere ganger i forskjellige formuleringer i stortingsmeldingen, men det munner ut i at teknologi er et fagområde som inngår i fagene matematikk og naturfag. De skriver at «Teknologi og design er et hovedområde i fellesfaget naturfag.», og de definerer at teknologisk kompetanse består «… både av kompetanse som alle elever bør ha, og mer spesialiserte ferdigheter og kunnskap som elever kan velge å få opplæring i…» (Meld. St. 28, 2015-2016, s. 54).

Som nevnt over ble det i 2016 sendt inn et representantforslag, som supplerer til Meld. St. 28, fra Trond Giske m.fl. (2016), som inneholdt en begrunnelse om at det å kunne bruke IKT og digitale tjenester godt, blir en stadig viktigere kompetanse i arbeidslivet og i samfunnet. De ville få på plass en nasjonal strategi for digitalisering i skolen, som inneholdt en nasjonal satsing innenfor blant annet utvikling av programvare for undervisning og forslag til en nasjonal innkjøpsordning for nasjonale læringsmidler (Giske, Aasen, Henriksen, Sønsterud, & Bjørnø, 2016). De begrunner at de vil ha denne satsingen innenfor teknologi på grunn av at «Elevene som begynner på skolen i år, skal være i arbeidslivet i 60 år frem i tid.» (Giske, Aasen, Henriksen, Sønsterud, & Bjørnø, 2016, s. 1), og derfor er det viktig at de får god undervisning innenfor teknologi.

2.2 Teknologi i skolen

«Teknologi har gjennom historien preget samfunnet så sterkt at det har gitt navn til historiske epoker som steinalder, industrialder og romalder.», skriver Andreas Sanne m.fl. (2016, s. 18). Teknologi ble først innført i læreplanen i naturfag i Kunnskapsløftet (LK06) i 2006, som et tverrfaglig tema med navnet *Teknologi og design* (Utdanningsdirektoratet, 2013), og som nevnt over har teknologibegrepet blitt mer og mer relevant i skolen, og i fagfornyelsen. «Teknologi bygger på håndverkstradisjonene, men omfatter i dag både digital teknologi, herunder programmering, og de ulike ingeniørdisiplinene.», skriver Anders Sanne m.fl. (2016, s. 11), og beskriver at teknologi omfatter analyse, utvikling og anvendelse av teknologiske produkter. De beskriver teknologi så enkelt som at «En pinne som ligger på bakken, er en del av naturen. Men når et menneske tar den opp og bruker den som et redskap, er det teknologi…» (Sanne, et al., 2016, s. 11). Teknologi kan sees på som moduler eller byggesteiner, og når disse modulene tas fra et område og anvendes på et annet område, skjer det en teknologisk innovasjon (Sanne, et al., 2016). Det at noe er et teknologisk produkt

gjenkjennes av at det er et redskap med en gitt oppgave eller funksjon, altså det er teknologiens mål å løse problemer og skape noe nytt (Sanne, et al., 2016).

Mange kan kanskje tro at programmering er et nytt tema som kom inn i skolen først etter fagfornyelsen, men faktumet er at det stemmer ikke. Seymour Papert (1983) skrev i 1980 ei bok som omhandler hvordan barn kan ha dialog med datamaskinen (oversatt til norsk i 1983). Han bruker samme illusjon om barnet som Jean Piaget, han ser på barnet som en byggmester. Papert levde etter metaforen om å imitere hvordan et barn lærer sitt første språk, gjennom å lære å kommunisere med datamaskiner. Papert (1983) jobber ut fra det han kaller «Piaget-læring», som er den naturlige, spontane læringen hos mennesket i interaksjon med miljøet. Han setter det i kontrast til pensumstyrt læring som er typisk for tradisjonelle skoler. Papert bruker Piaget som begrunnelse i sin undervisning, men han er samtidig kritisk til Piaget.

Papert lagde sitt eget programmeringsspråk, som ble kaldt LOGO ut fra en forskningsgruppe han hadde på MIT. Han mener at barn lærer vellykket læring ved måten barnet lærer å snakke på, som han mener er en prosess som finner sted uten bevisst og organisert undervisning. LOGO er et programmeringsspråk innenfor det Papert (1980) kaller «Turtle Geometry», eller skilpadde geometri på norsk. Der har Papert definert skilpadden som et punkt med retning, og kaller punktet skilpadde slik at det er enklere for barn å identifisere seg med skilpadden. Måten elevene kan snakke med skilpadden på er et språk han kaller «skilpaddesnakk», hvor kommandoene fram x og tilbake x får skilpadden til å bevege seg x antall steg i retningen kommandoen beskriver, mens venstre x og høyre x får skilpadden til å rotere x antall grader (Papert, Dialog med datamaskinen, 1983).

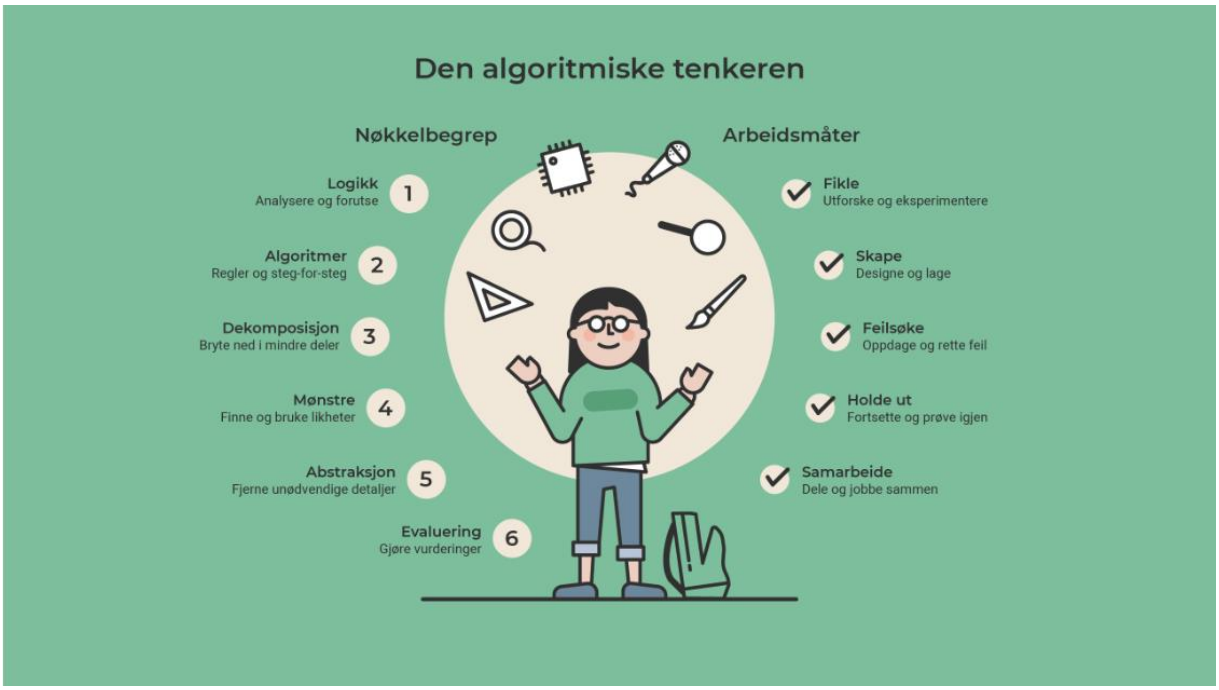
Et programmeringsspråk som er mer relevant for dagens skole er blokkprogrammering. Yuhan Lin og David Weintrop (2021) beskriver at blokk-basert programmering er det samme som å pusle, noen brikker passer sammen, mens andre ikke gjør det. Dette kan være grunnen til at blokk-basert programmering er et programmeringsspråk som kan være egnet for skolen, da det virker relativt enkelt å håndtere. Blokk-basert programmering har også blitt mer populært de siste årene på grunn av programmer som Scratch og micro:bit. Lin og Weintrop (2021) kommer inn på at blokk-basert programmering er lettere for nybegynnere å forstå, fordi det er forskjellige blokker, og alle trenger ikke nødvendigvis å passe i hverandre. De skriver at blokkene i blokk-basert programmering er lagd slik at blokker som ikke kan lage en valid kommando, ikke kan settes sammen, som viser til metaforen om at dette programmeringsspråket kan sees som et puslespill (Lin & Weintrop, 2021).

2.3 Algoritmisk tenkning

Algoritmisk tenkning er et tankesett som det har blitt lagt mer vekt på etter fagfornyelsen. Utdanningsdirektoratet (2019) beskriver algoritmisk tenkning som en problemløsningsmetode, og at det omhandler å tilnærme seg problemer på en systematisk måte. De definerer algoritmisk tenkning slik: «Å tenke algoritmisk er å vurdere hvilke steg som skal til for å løse et problem, og å kunne bruke sin teknologiske kompetanse for å få en datamaskin til å løse (deler av) problemet.» (Utdanningsdirektoratet, 2019, s. 1).

Begrepet algoritmisk tenkning ble først beskrevet av Jeannette Wing (2006), som i 2006 publiserte en artikkel med navnet *Computational Thinking*, som kan oversettes til algoritmisk tenkning. I artikkelen begrunner Wing hvorfor algoritmisk tenkning er viktig for alle å forstå, ikke bare «computer scientists». Wing (2006) definerer algoritmisk tenkning slik «Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science». Forskjellen på definisjonen mellom Utdanningsdirektoratet og Wing er at Utdanningsdirektoratet konsentrerer seg for det meste om hvordan problemløsning med hjelp av algoritmisk tenkning kan brukes mot teknologi og datamaskiner, mens Wing også drar inn hvordan problemløsning med hjelp av algoritmisk tenkning kan brukes til å forstå seg på mennesker og deres oppførsel. Videre kommer Wing inn på at hvis en algoritmisk tenker får et stort problem som skal løses, klarer den tenkeren å dele opp det store problemet inn i mindre og enklere problemer, for å løse oppgaven (Wing, 2006).

Utdanningsdirektoratet har laget en modell med det de mener er de viktigste nøkkelbegrepene og arbeidsmåtene som inngår i algoritmisk tenkning:



Figur 1 – Den algoritmiske tenkeren (Utdanningsdirektoratet, 2019, s. 2)

Her definerer Utdanningsdirektoratet seks forskjellige nøkkelbegrep, som er logikk, algoritmer, dekomposisjon, mønstre, abstraksjon og evaluering, og fem forskjellige arbeidsmåter, som er å fikle, skape, feilsøke, holde ut og samarbeide (Utdanningsdirektoratet, 2019). Det som inngår i nøkkelbegrepene, er hvilke tankeverktøy elevene bruker innen algoritmisk tenkning. Måten Utdanningsdirektoratet (2019) skriver om arbeidsmetodene, kan tolkes som at de mener at ved bruk at arbeidsmetodene i den algoritmiske tenkeren blir elevene mer robuste i oppgaveløsing i form av at de kan skaffe seg en «kognitiv kondis» og gjør at de må samarbeide for å bli bedre i oppgaveløsingen. Utdanningsdirektoratet (2019, s. 2) skriver at «Gode løsninger oppstår ikke i et vakuum, og samarbeid og deling er derfor sentrale arbeidsmetoder for den algoritmiske tenkeren.»

Utdanningsdirektoratets modell av den algoritmiske tenkeren er ikke veldig beskrivende når det kommer til forklaringen rundt de forskjellige punktene i modellen. De bruker for det meste bare to til tre ord i beskrivelsene, noe som gjør at den virker nesten litt uferdig. Derfor brukes andre kilder for forklaringen av hvert punkt.

2.3.1 Nøkkelbegrep

Det som inngår i nøkkelbegreper er konsepter som barn og elever bruker under oppgaveløsning i programmering (Barefoot Computing, u.å.c). Det er altså hvordan elevene bruker hodet under oppgaveløsing.

2.3.1.1 Logikk

Logisk tenking hjelper mennesker å forklare hvorfor noe skjer eller hva som kommer til å skje, med bakgrunn i forutsigbarhet (Barefoot Computing, u.å.h). Barefoot Computing (u.å.h) beskriver at hvis du gir to datamaskiner samme instrukser i programmet, og samme utgangspunkt, kan man nesten med sikkerhet vite at resultatet blir likt. Det er fordi datamaskiner ikke er selvstendige eller jobber annerledes på grunn av humør, de er forutsigbare (Barefoot Computing, u.å.h). Derfor kan man bruke logiske resonnement for å finne ut akkurat hvordan et program vil utspille seg (Barefoot Computing, u.å.h).

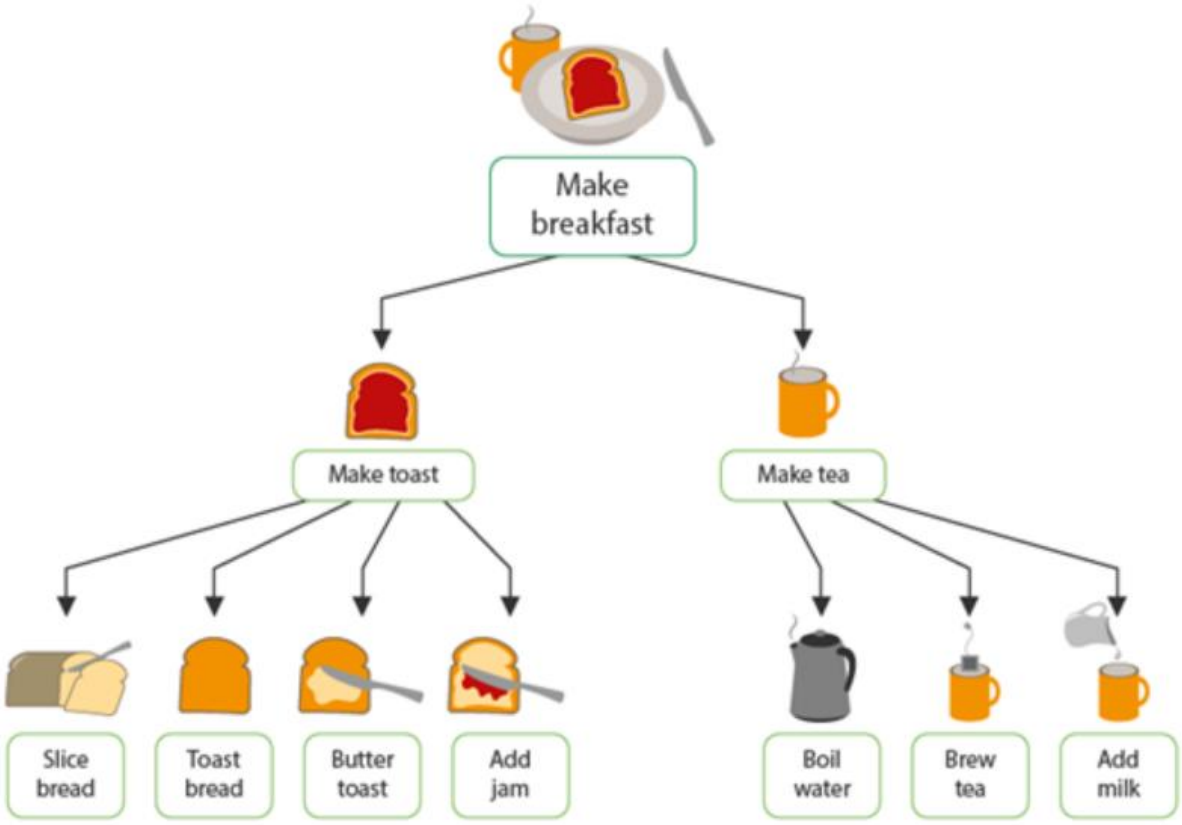
Barefoot Computing (u.å.h) skriver at barn er gode på å gjenkjenne og forstå logikk. Ved å se på andre og eksperimentere selv, klarer barn å tilnærme seg en mental modell av hvordan teknologi fungerer (Barefoot Computing, u.å.h). Et eksempel kan være hvis et barn spiller et dataspill, og trykker på knappen med pil til høyre, så beveger karakteren seg også til høyre (Barefoot Computing, u.å.h). Videre i spillet kan barn møte andre utfordringer som gjør at de enten må prøve seg på nye kommandoer, eller sette sammen det de allerede har lært for å klare utfordringen. På samme måte mener Barefoot Computing (u.å.h) at barn lærer gjennom logisk tenking i modellen for algoritmisk tenking.

2.3.1.2 Algoritmer

«Algoritmer er en sekvens av instruksjoner eller et sett med regler for å få noe gjort.», skriver Barefoot Computing (u.å.b). Sanne m.fl. (2016) beskriver at algoritmer er et programmeringsspråk som en datamaskin kan tolke, mens Utdanningsdirektoratet (2019) beskriver det som framgangsmåter. Barefoot Computing (u.å.b) beskriver et eksempel på algoritmer for barn, som hvilket veivalg barnet velger når det skal gå hjem fra skolen. Man kan ha mange forskjellige veivalg å velge mellom, med hver sin sti for hvordan barnet skal komme seg hjem. På denne måten kan man nesten se på algoritmer som mer viktige for mennesker å forstå, da algoritmene kan variere i forskjellige programmeringsprogram (Barefoot Computing, u.å.b). Wing (2006, s. 35) skriver at datamaskiner er kjedelige, det er mennesker som gjør datamaskiner interessante, og at det er på grunn av menneskets intelligens som gjør at vi klarer å bygge system (algoritmer) som er funksjonelle.

2.3.1.3 Dekomposisjon

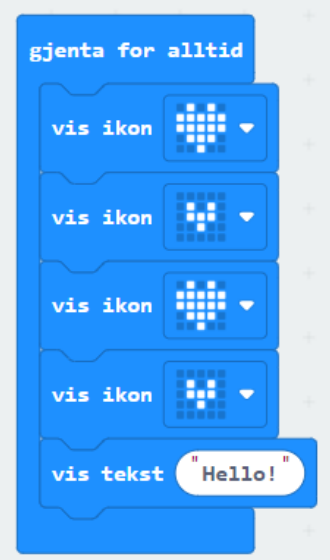
Kunnskapen til å bryte ned komplekse problem til mindre og enklere problemer er en egenskap som har flere fordeler (Barefoot Computing, u.å.f; Utdanningsdirektoratet, 2019). Det gjør det enklere å håndtere store prosjekter og gjør prosessen ved å løse et komplekst problem, mindre skremmende, og enklere å hive seg på (Barefoot Computing, u.å.f). Ved bruk av dekomposisjon i samarbeidsoppgaver, stiller hvert gruppemedlem med kunnskap og ferdigheter særegen for dem, og kan bidra på forskjellige aspekter i prosjektet (Barefoot Computing, u.å.f). I et gruppearbeid kan man også dele hovedoppgaven i mindre problemer, og fordele disse mellom gruppemedlemmene. Et enkelt eksempel på dekomposisjon kan være å lage frokost, som vist på figur 2. Her ser man at det er mulig for to personer å hjelpe til på samme oppgave.



Figur 2 Dekomposisjon (Barefoot Computing, u.å.f)

2.3.1.4 Mønstre

Mønster kan man finne over alt. Ved å gjenkjenne mønster kan man lage regler og løse flere generelle problemer (Barefoot Computing, u.å.i). I algoritmisk tenkning er en egenskap som å søke seg gjennom flere sekvenser av data for så å klare og gjenkjenne mønster, sett på som en



Figur 4 Eksempel før mønstergjenkjenning og optimalisering

Figur 3 Eksempel etter mønstergjenkjenning og optimalisering

god egenskap (Wing, 2006). Grunnen til at gjenkjenning av mønster er en viktig egenskap er fordi innen algoritmisk tenking hjelper det med å optimalisere hastigheten på å løse et problem (Barefoot Computing, u.å.i). Et mønster som kan være i programmering kan være for eksempel at et program starter som figur 3 og ender opp som figur 4 etter man har lagt merke til mønsteret.

2.3.1.5 Abstraksjon

Abstraksjon er en viktig egenskap når det kommer til algoritmisk tekning, da det omhandler å forenkle ting for å gjøre utføringen av problemet enklere (Barefoot Computing, u.å.a; Wing, 2006). Det omhandler å identifisere hva som er viktig i et problem, uten å gå for mye inn i detaljer (Barefoot Computing, u.å.a). Det vil si at abstraksjon representerer noe som er forenklet til de detaljene som er nødvendig å vite for å løse oppgaven. En abstraksjon som brukes i hverdagen kan være noe så enkelt som en ukeplan elevene har å forholde seg til. Den viser hvilke fag elevene skal ha til enhver tid, men trenger ikke vise detaljert hvilket innhold som skal skje for hver time.

2.3.1.6 Evaluering

Å kunne evaluere omhandler å gjøre vurderinger der det er mulig, på en objektiv og systematisk måte (Barefoot Computing, u.å.g). Evaluering går også ut på å vurdere kvalitet, effektivitet og prestasjon av produkter, løsninger og prosesser (Barefoot Computing, u.å.g). Elever som da skal bruke den algoritmiske tenkeren i programmeringsoppgaver, skal forstå hva de har programmert, hvordan de har programmert og hvorfor de har programmert på den måten de har gjort (Sevik, 2016). Et eksempel kan være at elevene får en oppgave hvor de må vurdere spesifikke kriterier, som lengde og fart, for å oppfylle det oppgaven trenger for å bli sett på som suksessfull (Barefoot Computing, u.å.g).

2.3.2. Arbeidsmåter

Det som inngår i arbeidsmåter er måter barn og elever går fram og jobber med oppgaveløsing i programmering (Barefoot Computing, u.å.c). Altså hvordan elevene jobber med hendene og kroppen, sett sammen med hvordan de bruker hodet.

2.3.2.1 Fikle

Yihuan Dong m.fl. (2019) skrev i en artikkel at fikling har vist seg å ha positiv påvirkning på elever når det kommer til å løse mer åpne oppgaver. Barefoot Computing (u.å.k) beskriver fikling som at vi ofte utforsker nye ting for å oppdage hva det gjør og hvordan det virker. Videre beskrives det som å være en stor del av individuell læring med tanke på å erfare for eksempel at «hvis man trykker på denne knappen, så skjer det, men hvis man trykker på den andre skjer det noe annet.» Etter hvert lærer barn hvordan ting fungerer basert på erfaringer som bygges mens man fikler. Ved å gi barn og elever mulighet til å eksperimentere risikofritt på denne måten, gjør det ofte at de blir tryggere i seg selv, men det ufarliggjør også det faktum at man kan, og kanskje må, fikle med ting for å lære mer om det (Barefoot Computing, u.å.k).

2.3.2.2 Skape

Å skape kan man gjøre både i den fysiske verden, men også den digitale. Programmering er i seg selv en kreativ prosess, hvor man enten skaper noe nytt eller endrer og forbedrer et program som allerede eksisterer. Å skape gjennom programmering og digitale medier, gir barn mulighet til å uttrykke seg selv på en kreativ måte (Barefoot Computing, u.å.d). Gjennom programvare og digitale medier får barn rom til å mestre diverse programvareverktøy, og opparbeider seg kompetanse, trygghet og uavhengighet som videre

kan brukes på en leken måte ved å komme fram til gode løsninger med et målrettet uttrykk av deres ideer (Barefoot Computing, u.å.d).

2.3.2.3 Feilsøke

Det er sjelden at et komplekst program virker som det var ment ved første forsøk. Derfor er det viktig at man kan å lete etter feilkoder. Feilkoder er feil man finner i algoritmer og koder, derfor blir feilsøking en prosess som omhandler å finne feil og fikse dem (Barefoot Computing, u.å.e). Å lete etter feil i et program, kan være mye mer tidkrevende enn selve programmeringsprosessen. Barefoot Computing (u.å.e) skiller mellom to typer feil som kan forekomme, feil i logikken i programmet og feil i syntaksen i programmet. De beskriver det som at når det er feil i logikken i programmet, er det som at plottet i en historie ikke gir mening (Barefoot Computing, u.å.e). Hvis det er feil i syntaksen i programmet, er det for dårlig formulert hva programmet skal gjøre, mer som dårlig staving (Barefoot Computing, u.å.e). Barefoot Computing (u.å.e) mener man kan feilsøke gjennom fire trinn ved å:

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

Elever som har lært å forvente at algoritmer og koder kommer til å inneholde feilkoder, har ofte egenskapen til å finne feilen og rette opp i den raskere, uten at det blir sett på som en stor utfordring for utførelsen av oppgaven (Barefoot Computing, u.å.e). Det er derfor viktig at læreren gir elevene mulighet til å måtte løse sine egne feil, uten å hjelpe dem med en gang de møter på et problem, slik at elevene får øve seg på å fikse egne feil.

2.3.2.4 Holde ut

Programmering kan ofte være frustrerende og krevende. Derfor er det viktig at elever lærer seg å holde ut i frustrasjonen. «Å holde ut inneholder å være målbevisst, fleksibel og utholdende – altså, å aldri gi seg.», skriver Barefoot Computing (u.å.j). Programmering krever mer enn å forstå seg på programmeringsspråket, algoritmer og koder. Det krever at viljen til å ville holde ut for å løse oppgaven eleven har foran seg. Å lære seg å holde ut omhandler å øve og trene for å utvikle en «kognitiv kondis» til å holde ut i problemer som oppstår (Barefoot Computing, u.å.j; Utdanningsdirektoratet, 2019).

Hvis elever har lært seg gode metoder ved bruk av dekomposisjon og feilsøking, fører det ofte til en bedre kognitiv kondis, da de har verktøyene for å gjøre operasjonen enklere å løse, istedenfor å bare ende opp i frustrasjon (Barefoot Computing, u.å.j). For å oppfordre elever til å utvikle egenskapen med å holde ut, bør elevene bli vist hvordan de eventuelt kan dele opp problemet, slik at det blir enklere å løse. En ting som kan gjøres for å gjøre det enklere for elevene, er å gjøre dem kjent med de forskjellige trinnene i feilsøking, slik at de ikke direkte går fra å finne ut at det er et problem og direkte går til å skal fikse det, uten å vite hvor problemet i programmet egentlig ligger, da punkt tre i Barefoot Computings fremgangsmåte blir oversett (Barefoot Computing, u.å.j).

2.3.2.5 Samarbeide

Samarbeid omhandler å arbeide sammen med andre ved å dele oppgaver og diskutere sammen, slik at man oppnår et felles mål med best mulig resultat (Barefoot Comuting, u.å.l; Utdanningsdirektoratet, 2019). «Samarbeid er sentralt i arbeidet med programmering og kan ses i sammenheng med samarbeidslæring slik vi kjenner det i skolen.», skriver Sevik m.fl. (2016, s. 15). Samarbeid legger til rette for samtaler og diskusjoner om utfordringene elevene kan møte på under en oppgaveløsing, og gir grunnlag for refleksjon og kritisk tenking om hvordan problemet kan løses (Sevik, 2016). Slike samtaler mellom elevene skal bidra til at de kommer frem til en løsning på problemet. Samarbeid kan også være en motivasjonsfaktor til at elevene holder ut når det kommer til oppgaveløsning, da de har en partner å støtte frustrasjonen på (Barefoot Comuting, u.å.l).

Ludvigsenutvalget (2015) ser på samarbeid som en god mulighet til å utvikle nysgjerrighet, kreativitet, utholdenhet og fantasifull problemløsing. De kommer også inn på at egenskapen om å kunne samarbeide er en særdeles viktig egenskap i dagens arbeidsliv, med tanke på at det stiller høye krav til omstillingsevne og kompetanse (NOU 2015:8, 2015). Elever som da lærer seg godt samarbeid blir ofte flinkere til å kommunisere og samhandle med andre (NOU 2015:8, 2015).

2.3.3 Computational thinking vs den algoritmiske tenkeren

Wing (2006) kommer inn på at det er flere områder som inngår i algoritmisk tenkning. Hvis noen skal løse et problem, er det ikke selve programmeringen som løser problemet. Det er flere aspekt av abstrakt tenkning om problemet, av personen som programmerer. Det vil si at personen som skal løse problemet, klarer å se bort ifra unødvendige detaljer som ligger i problemet, da en ikke trenger å ta hensyn til disse for å få problemet løst (Wing, 2006).

Digitale ferdigheter er fundamentale ferdigheter for å kunne fungere i samfunnet. Det betyr at det er et behov for en viss kunnskap om digitale ferdigheter for å kunne fungere både i arbeidsliv og privatliv, da det er ekstremt mye teknologi rundt oss som vi må kunne bruke. Den riktige måten å se algoritmisk tenkning på er «måten mennesker tenker på, ikke datamaskinen» (Wing, 2006, s. 35). Grunnen til at det er viktig å se det på denne måten er fordi det er måten mennesket løser problemer på, med datamaskinen som verktøy for problemløsning som er interessant for oss, ikke hvordan datamaskinen løser problemet alene. For å få det maksimale ut av algoritmisk tenkning og problemløsning er det viktig å kombinere matematikk og innovativ tenkning (Wing, 2006). Det gjør det mulig å bygge utenfor den fysiske verden. Det som kan menes med å bygge utenfor den fysiske verden, kan være å bruke programmering på ei datamaskin som et verktøy til å bygge i en virtuell verden, som ikke gir de samme fysiske utfordringene som å bygge i virkeligheten. Med tanke på gjenstandene som er i den fysiske verden, så er det ikke disse som egentlig er viktige for oss, det er ideene og all problemløsningen som lå bak prosessen med å lage gjenstanden, som er viktig (Wing, 2006).

Videre er det kanskje mange som ikke ser viktigheten i å lære seg algoritmisk tenkning, spesielt kanskje for foreldre i skolen. Mange ser ikke viktigheten i at barna lærer seg algoritmisk tenkning, da de tror at algoritmisk tenkning kun inngår i et smalt felt i arbeidslivet (Wing, 2006). De tror at det fundamentale innenfor feltet er ferdig forsket, og at det ikke er stort for deres barn å innhente ved å lære seg algoritmisk tenkning, men det stemmer ikke. Du kan ha en karriere innenfor mange yrker hvis du kan å bruke algoritmisk tenkning, som innenfor medisin, rettssystemet, politikk etc.

Men kan det være en forskjell mellom Wings definisjon og hvordan algoritmisk tenkning blir brukt i grunnskolen? Valerie Barr og Chris Stephenson (2011) kommer inn på at algoritmisk tenkning i grunnskolen har blitt definert av Computer Science Teachers Association (CSTA) og International Society for Technology in Education (ISTE), hvor de har definert begrepet ved å dele det inn i konsepter og muligheter (Barr & Stephenson, 2011). De lagde en tabell som inneholder konseptene, som virker som å være definert innen fag, og mulighetene, som definerer hvilke muligheter algoritmisk tenkning har i faget. De har definert hvordan algoritmisk tenkning kan brukes i naturfag, og mulighetene som er mulig for algoritmisk tenkning i faget er datainnsamling, dataanalyse, representasjon av data, bryte ned problemer, gjøre ting abstrakt, algoritmer og prosedyrer, automatisering, parallellisering og simulering (Barr & Stephenson, 2011).

Eksempler på dette kan da være at under datainnsamling, kan elevene samle inn data fra et eksperiment, videre under dataanalyse kan elevene analysere dataen de har samlet inn fra eksperimentet. Deretter kan elevene, under representasjon av data, sette opp og summere dataen fra eksperimentet de utførte. Under å bryte ned problemer kan elever se på for eksempel bli gitt ei utfordring innen teknologi og programmering, hvor de blir nødt til å dele hovedproblemet opp i mindre deler, slik at problemet blir enklere å løse. Å gjøre ting abstrakt for elever kan være å bygge en modell av en fysisk ting i for eksempel minecraft eller plage en modell med et annet verktøy elevene behersker. Under algoritmer og prosedyrer kan elevene for eksempel utføre forskjellige prosedyrer når det kommer til eksperimenter eller oppgaver. En slik prosedyre kan være så enkelt som å kle på seg om morgenen eller å smøre matpakken om morgenen før skolen. Automatisering vil si at det blir enklere for elever å gjøre noe. I naturfag kan automatisering sees på som eksempel at elever forbereder seg til å gjøre et forsøk, så har de automatisert at de skal ha på sikkerhetsutstyr før de gjør noe videre i forsøket. Parallellisering i naturfag kan være å kjøre to forsøk av det samme eksperimentet samtidig, bare med noen komponenter som er forskjellig, og se på hvordan de komponentene kan påvirke resultatet til forsøket. At elevene skal simulere noe i naturfag kan være så enkelt som å simulere solsystemet, enten ved hjelp av programmering eller ved å bygge en modell som simulerer bevegelsene i solsystemet.

2.4 Micro:bit/Super:bit

Super:bit kan bli sett på som et resultat av Giske m.fl. håp om en nasjonal satsing på innkjøp av nasjonale læremidler. Super:bit skole ble utviklet av de regionale vitensentrene i Norge og er en del av regjeringens teknologiske skolesekk som er en nasjonal satsing i regi av Utdanningsdirektoratet (Vitensenterene, u.å.a). Super:bit er et klassesett som alle sjetteklasser i Norge fikk utdelt mellom 2019-2021 (Vitensenterene, u.å.a).

Christina Frausig Binau og Olga Trolle (2021) har forsket på hvordan ultra:bit, som var et prosjekt som gikk ut på å forsterke læreres syn på teknologiens relevans i undervisning. De brukte micro:bit som verktøy i programmeringsundervisningen, men hva er egentlig en micro:bit? «BBC micro:bit er en datamaskin i lommeformat som du kan programmere, tilpasse og kontrollere for å sette dine digitale ideer, spill og programmer ut i livet.», skriver n00b.no (u.å.). Det er bevist at micro:bit fører til at elevenes interesse for koding og teknologi styrkes ved å bruke micro:bit i undervisningen (Binau, F, & Trolle, 2021). Det at elevenes interesse øker, spesielt for teknologi, hjelper til at de ønsker å fortsette å lære seg koding. Det blir på en måte en evig sirkel hvor elevene utvikler teknologisk kunnskap, frem til sin egen

kunnskap om programmering, og deretter må utvikle både programmeringskunnskapen og teknologikunnskapen for å fortsette henge med i teknologiens utvikling.

I den norske skolen har som tidligere nevnt, har alle sjetteklasser fått super:bitsett, som inneholder blant annet 20 microbiter, roboter, dioder og annet utstyr som kan kobles til micro:biten (Vitensenterene, u.å.b). En micro:bit mottar enten blokk-basert programmeringsspråk eller tekstbasert programmeringsspråk. For de fleste på grunnskolen er det enklest å lære seg programmering gjennom blokk-basert programmering i starten, og eventuelt gå over til tekstbasert programmering.

3. Metode

Problemstillinga i studien er, som tidligere nevnt:
På hvilken måte preges elevers aktivitet av algoritmisk tenking ved løsning av programmeringsoppgaver?

Målet med denne studien var derfor å finne og gjenkjenne algoritmisk tenkning gjennom elevers utføring av programmeringsoppgaver. For utføre det på best mulig måte, ble metoden video-observasjon benyttet.

I denne delen av studien, vil det kort presenteres oppgavens filosofiske vinkling og forskningsmetode, med begrunnelser som bygger under valgene som har blitt tatt. Oppgaven vil videre beskrive begrunnelser av valg for utvalg og gjennomførelse av forskningen. Deretter beskrives analysemetode og transkriberingsprosess. Til slutt vil det diskuteres etiske valg som ble gjort før, under og etter forskningen var gjennomført, samt diskutere validitet og reliabilitet innenfor studien.

3.1 Fenomenologisk ståsted

I denne studien var det relevant med en fenomenologisk vinkling, og det kan begrunnes med et sitat fra Nyeng (2012, s. 33): «… studier som tar for seg hvordan mennesker opplever ulike sider ved sin tilværelse, hvilket innhold som fylle bevisstheten, og hvordan deres tanker og refleksjoner avslører hvilke forhold som fremstår som virkelige for dem.» Måten dette beskrives på reflekterer hvordan datainnsamlingen ønskes i innhentes på, med tanke på at målet med studien er å se på hvordan elever kan bruke algoritmiske nøkkelbegreper underveis i arbeidet med programmeringsoppgaver. Dermed, med et fenomenologisk syn på forskningen, kan en se på elevens løsningsmetoder ved programmeringsoppgaver, og se om elevene faktisk bruker algoritmiske nøkkelbegreper underveis i oppgaveløsning.

Et annet kriteria innenfor fenomenologien som gjør det enkelt å begrunne hvorfor studien er fenomenologisk og ikke hermeneutisk er at i hermeneutikken er det et krav om å ha forkunnskap om fenomenet som forskes på (Gilje & Grimen, 1993). Det er ikke et krav om forkunnskap i fenomenologien, fordi det går ut på at hvert individ erfarer verdens virkelighet forskjellig (Nyeng, 2012). Med tanke på problemstillingen, kan det være vanskelig å ha en forkunnskap om hvordan resultatet kan bli, da resultatet fra hvert individ kan variere.

3.2 Kvalitativ forskningsmetode

Målet for kvalitativ forskning er som oftest å tolke og forstå et fenomen eller en case, altså det har ikke et mål om å generalisere resultatet i forskningen (Nyeng, 2012). Denne oppgaven har som mål om å finne ut hvordan forskjellige nøkkelbegreper innenfor algoritmisk tenkning kan gjenkjennes gjennom programmeringsoppgaver. Derfor ble det anvendt en kvalitativ forskningsmetode. Dette er i hovedsak fordi ønsket er å forske på om et spesielt tankesett kan oppnås gjennom en gitt oppgavekontekst, noe som best kan observeres. Det ble sendt inn en søknad til Sikt som ble godkjent (vedlegg 1).

3.3 Utvalg

Kvalitativ forskning har ikke mål om generalisering av resultater, og derfor er det ofte at utvalget velges på et ikke-sannsynlig-utvalg, og man kan av bekvemmelighet begynne å studere det som er lett tilgjengelig (Nyeng, 2012). Kvalitativ forskning har et utvalgsproblem der målet er å forstå kontekst og beskrive sosiale prosesser og relasjoner mest mulig inngående, og derfor kan man snakke mer om grader av overførbarhet mellom lignende fenomener, istedenfor å kunne generalisere forskningsfunnene, slik det gjøres i kvalitativ forskning (Nyeng, 2012).

Kvalitative studier krever derfor ofte mindre, mer spesifikke utvalg når det kommer til å finne kandidater til studien. Derfor består utvalget for denne studien av 51 mellomtrinnselever i en alder av 9-14 år, og utvalget ble strategisk utvalgt av bekvemmelighetsgrunner.

3.3.1 Observasjonsguide

For denne studien ble det utviklet et informasjonsskjema (vedlegg 2). Informasjonsskjemaet ble utformet med hovedintensjon om å forske ved hjelp av video-observasjon.

Informasjonsskjemaet ble utformet fra en mal på Sikt og inneholdt informasjon om hva studiens formål var, informasjon om hvordan kandidatenes personvern ville bli tatt vare på, samt informasjon om kandidatens rettigheter angående innsyn av opplysninger, krav til å trekke samtykke og lignende.

3.4 Forskningsdesign

For å svare på problemstillingen på best mulig måte, blir observasjon som forskningsdesign mest relevant. Observasjon handler om å bruke synet for å få med seg hva som skjer i en setting man ønsker å observere. Det som er viktig under observasjon er at forskeren er til stede i situasjonen som er relevant for studien, og observerer og registrerer det som skjer (Larsen, 2007). Derfor blir det noe vi har sett, som vi noterer oss enten under selve observasjonen, eller etter observasjonen har skjedd (Larsen, 2007).

Høgheim (2020) kommer inn på at observasjon kan gjøres på flere forskjellige måter, men det mest sentrale elementer i metoden er konteksten og forskeren. Når det kommer til kontekster er det enten naturlige kontekster, hvor forskeren observerer situasjoner uten å gripe inn, mens i simulerte kontekster har forskeren «satt sammen» kontekster for å forske på resultatet av den sammensetningen (Høgheim, 2020). Forskeren kan enten være passiv eller aktiv med hvor involvert den er i studien, og forskeren kan være åpen eller lukket når det kommer til informasjon som skal deles med elevene om studien (Høgheim, 2020).

3.4.1 Gjennomføring

Måten forskningen av studien ble utført på, var ved hjelp av GoPro-kamera. Grunnen til at dette valget ble tatt, var at ved å bruke denne metoden kunne flere kandidater observeres samtidig, samt at datainnsamlingen ble mer effektiv. Ved bruk av kamera istedenfor fysisk observasjon, ble mer datamateriale innsamlet. Datamaterialet ble også mer nøyaktig enn eventuelle egne observasjoner. Før innsamlingen av datamaterialet ble gjennomført, fikk kandidatene informasjon om hva studien omhandlet. Under gjennomføringen fikk kandidatene jobbe uforstyrret og selvstendig, som vil si at denne studien har en åpen, men passiv forskningsform.

Elevene hadde til sammen fire undervisningstimer som inneholdt programmering, hvor de første to timene gikk til introduksjon til micro:bit, og de to siste timene gikk til mer utforskende oppgaver med hjelp av micro:bit. Oppgavene som elevene fikk den første timen var direkte undervisningsopplegg fra superbit.no, hvor det første var en presentasjon (vedlegg 3), hvor elevene skulle være roboter og læreren programmereren. Deretter fikk elevene utdelt micro:bit og startet med oppkobling og fikling. Da alle var oppkoblet fikk elevene utdelt et oppgaveark (vedlegg 4), der de skulle prøve seg frem litt selv.

I andre time hadde læreren laget en egen presentasjon (vedlegg 5) hvor elevene individuelt skulle programmere micro:biten til å bli en terning. Deretter skulle alle elevene reise seg for å spille terning med utslagsrunder mot hverandre. Den siste som sto igjen, var vinner. Det ble spilt to runder, hvor første var at du vant hvis du fikk størst tall, og på runde nummer to var det minste tallet som vant. Etter dette gikk timen videre til at elevene skulle lære seg å bruke variabler, ved å lage skritteller. Det ble en felles gjennomgang om hvordan man skulle lage programmet, og hva betydningen av de forskjellige blokkene var. Da alle hadde programmet klart, festet de micro:bitene til en fot og gikk i to minutter, og leste deretter av resultatet. De to siste oppgavene som elevene skulle gjøre hang litt sammen, hvor den første oppgaven gikk ut på at elevene skulle lage et program som gjorde en ting når de trykket på knapp A, og gjorde en annen ting når de trykket på knapp B. Neste oppgave gikk ut på at de skulle programmere videre på det eksisterende programmet de hadde, og få micro:biten til å gjøre forskjellige ting hvis de trykket på knapp A og deretter knapp B, ble ristet og programmene måtte inneholde minst en løkke og en blokk med «*gjenta for alltid*.»

I tredje time ble elevene introdusert til Bit:Bot, hvor læreren først viste elevene en video fra superbit.no (vedlegg 6). Videoen ble satt på pause etter hvert store steg som personen i videoen gjorde, slik at elevene kunne følge instruksene steg for steg med videoen. Da videoen var ferdig fikk elevene oppgaven som introduksjonsvideoen ledet opp til. Oppgaven gikk ut på at elevene skulle få Bit:Boten til å kjøre en meter, snu ved metersmerket og kjøre tilbake. For å få til dette på en nøyaktig måte fikk elevene utdelt to svarte teipbiter og et måleband, slik at de kunne markere en meter på gulvet.

I den fjerde og siste timen skulle elevene fortsette å forbedre programmet de hadde fått timen før, til de hadde løst oppgaven. Da oppgaven var løst av de fleste gikk timen videre til den siste oppgaven (vedlegg 7). Denne oppgaven gikk ut på at elevene skulle lage en robotgressklipper-/støvsuger, altså programmere Bit:Boten til å kjøre på gulvet og svinge unna eventuelle hindringer som kommer i veien for den. Elevene fikk se et uferdig program på tavla og måtte prøve seg frem for å få Bit:Boten til å unngå hindringene.

3.5 Analysemetode

3.5.1 Tematisk analyse

Analysemetoden som skal brukes i denne oppgaven er Virginia Braun og Victoria Clarkes (2006) tematiske analyse, som deles inn i seks faser. De beskriver denne analysemetoden som en mer nybegynnervennlig analysemetode innenfor kvalitativ forskning (Braun & Clarke,

Thematic Analysis, A Practical Guide , 2022, s. 4). Tematisk analyse går ut på å transkribere alt datamateriale, deretter analysere og kode rådata, hvor kodene deretter skal plasseres inn under forskjellige tema, og til slutt lage en slags rapport med de endelige beskrivelsene av tema og siste del av analysen. (Braun & Clarke, 2006). Braun og Clarke påpeker at tema er de ultimate analytiske mål, som vil si at det er temaene som skal analyseres og reflekteres rundt (Braun & Clarke, Thematic Analysis, A Practical Guide , 2022, s. 4).

De forskjellige fasene i beskrivelsen til Braun og Clarke (2006, s. 87) er:

1. Bli kjent med datamaterialet ditt. Det vil si transkriber, og bli kjent med datamaterialet.
2. Generalisere innledende koder. Kode interessante trekk i datasettet, gjennom alt datamaterialet.
3. Se etter tema. Samle koder til potensielle tema, og samle alle data som er relevante for hvert potensielle tema.
4. Gjennomgå tema. Sjekke om tema fungerer i forhold til de kodede utdragene og deretter hele datasettet, og lag et kart av analysen.
5. Definering og navngiving av tema. Avgrense detaljene for hvert tema, og lag klare definisjoner og navn for hvert tema.
6. Lage en rapport. Bruke utvalg av overbevisende, eksakte eksempler, endelig analyse av utvalgte utdrag, relater analysen til forskningsspørsmål og litteratur, for å produsere en vitenskapelig rapport av analysen.

3.5.2 Transkribering, koding og analyse

Den første fasen til Braun og Clarke (2022) går ut på å bli godt kjent med rådataen. Det vil si å transkribere datamaterialet, hvis nødvendig, og lese gjennom det flere ganger mens man noterer seg diverse ideer man kommer over (Braun & Clarke, 2006).

For å kunne analysere videoklippene, måtte datamaterialet transkriberes, altså omformulere muntlig datamateriale til tekst (Høgheim, 2020).

Før transkriberingen kunne skje, ble hver kandidat oppført i et deltakerskjema (vedlegg 8) og gitt et tall, som ble deres kandidatnummer når videoklippene skulle transkriberes og anonymiseres. Kandidatene ble tilfeldig innskrevet i deltakerskjemaet, slik at det minimerer gjenkjennelse av deltakerne. Deretter ble hvert videoklipp sett igjennom, og deretter ble det skrevet et detaljert manus med rådata, om hva som både ble gjort og sagt. Med tanke på at det totalt innsamlede datamaterialet var på til sammen 72 timer med videoklipp, ble noen av de

første timene transkribert, men etter hvert ble det mer hensiktsmessig å bare transkribere de to siste timene (altså 3., og 4. time) klassene hadde, da det var her det var mest selvstendige oppgaver som ikke var steg for steg følging, og det er som oftest slike oppgaver algoritmisk tankemåte best stimuleres i. De seks kameraene ble fordelt på tre grupper i hver klasse, og blir derfor videre beskrevet i oppgaven som gruppe 1 – 9. Gruppenumrene er tilfeldig gitt spredt over klassene, for sikrere anonymisering.

Den første transkriberingen av alle videoklippene, ble transkribert på dialekt, for å få med alt meningsverdig innhold og at ingenting skulle bli mistet i oversetting til bokmål. Små ord som «ehm» «oi» og «hmm» ble også tatt med i transkriberingen, for å bedre vise at kandidaten tenkte og vurderte. Når det kommer til sitering av transkriberingene i denne oppgaven, vil sitatene bli oversatt til bokmål for å beholde mest mulig anonymitet for kandidatene.

Utdrag fra transkribering av rådata, tabell 1:

L = lærer **nr.** = kandidat **[---]** = beskriver det elevene gjør

vanlig skrift = det de sier **…** = stopp i prat/tenkepause

Kamera 1 – Time 2, gruppe 8	
Personer	Tekstutdrag
Lærer	Så nå skal alle sammen lage den der [viser en kode på tavla], så laster dere den ned til micro:biten.
Elev 1	Ah, okei. Når ristes.. [plasserer den i koderommet, finner frem gjenta fire ganger og plasserer den i gjenta for alltid] nei [flytter den til når ristes]. Gjenta fire ganger, vis tall mellom tilfeldig [finner vis tall på basis og deretter finner velg tilfeldig x til x på matematikk, og gir tallene 1 og 6]. Sånn, og så må jeg…
Elev 2	Du må kanskje slette alle de andre greiene…
Elev 1	[prøver å slette det som ikke brukes i selve kodingen] nei, åh herregud.
Elev 2	Oi, fem, fem, fire, fem, fem, seks…
Elev 1	[holder inn knappene og laster ned]
Lærer	Hva er det egentlig vi holder på å lage da tror dere?
Elev 3	Åja, en terning
Lærer	En terning, hvor mange ganger kaster vi denne terningen?
Elev 3	Fire ganger?
Lærer	Yes, så her har vi egentlig laget.. Vi egentlig en terning som, jeg sier kaste, men vi rister da, for å få den til å lage et tilfeldig tall mellom 1 og 6, sånn som en terning gjør. Dere vil sikkert få fire tall etter hverandre, for dere skal… Den kaster på en måte fire ganger.

Tabell 1 Eksempel på transkribering av rådata

I fase nummer to beskriver Braun og Clarke (2006) at man må finne kjennetegn til forskjellige koder, og generalisere dem. Det vil si samle inn data relevant til hver kode som bestemmes (Braun & Clarke, 2006).

Derfor ble det naturlig å fortsette transkriberingsprosessen med å gi en kvalitativ beskrivelse av forskjellige interessante trekk i datasettet. Et eksempel på hvordan dette ble gjort vises i tabell 2, hvor forslag til tema blir skrevet, som inngår i fase tre, hvor man skal se etter tema. Det ble fargekodet i forskjellige farger ut ifra hvilken kode som ble funnet, opp mot hvilket tema som koden kan knytes opp mot. Utdrag fra videre transkribering:

Kamera 1 – Time 3, gruppe 2			
Personer	Tekstutdrag	Kvalitativ beskrivelse	Eventuelt tema
Elev 1	[Leser på skjermen] Sensors… Lys… Andre lys! Andre lys! [Observerer at noen andre har regnbuelys på sin bil] Oi! Hvordan får man rainbow? [Begynner å lete i fargepalettet til koden «Sett alle LED til farge», som allerede ligger i programmet deres] Hvordan får man det da…	Det kan se ut som at elevene brukte logikk til å forutse hvor de skulle lete etter den gitte kommandoen, som i dette tilfellet var rainbow-funksjonen Elev 1 bruker algoritmer til å bestemme hvor kommandoen til regnbuelyset må være for at det skulle lyse gjennom hele kjøringen, og setter inn regnbuekoden øverst i programmet. --- Elevene forbedrer kodene ved å legge inn farger på Bit:Boten. Elevene diskuterer hvilke løsninger de har å velge i når det kommer til lys.	Nøkkelbegreper: Logikk Algoritmer Arbeidsmåter: Skape Samarbeid
Elev 2	Det er ikke her… Det er en annen greie.. Sett.. [Leter i lyskategorien]		
Elev 1	[Spør andre elever] Hvordan får man sånn…		
Elev 2	Vis.. Her tror jeg! Konverter fra… Nei… Her! Sett LED til regnbue. [Drar den inn i programmet]		
Elev 1	[Fjerner «Sett alle LED til turkis» og setter inn «Sett alle LED til regnbue» istedenfor, først i programmet]		
Elev 2	Nei! Vi skal jo ikke ha…		
Elev 1	Jo, vi skal ha regnbue.		
Elev 2	Men vi skulle ikke ta den (mener «set alle LED til turkis») bort! Vi må jo ta den regnbuegreien på slutten.		
Elev 1	Nei, men da vises det ikke når den kjører…		

Tabell 2 Transkribering med kvalitativ beskrivelse og tema

Videre i fase tre blir det beskrevet at man skal samle kodene for de potensielle temaene og deretter samle all data som kan være relevant for hvert potensielle tema (2006). Denne studien har allerede bestemte tema som skal gjenkjennes basert på nøkkelbegrepene og arbeidsmetodene i Utdanningsdirektoratets modell på den algoritmiske tenkeren (figur 1). Utdanningsdirektoratet har allerede gitt nøkkelbegrepene og arbeidsmetodene en beskrivelse på hva de inneholder, men som tidligere nevnt, var de ikke veldig beskrivende. Derfor ble det

valgt å bruke andre kilder for å forklare mer hva hvert punkt i den algoritmiske tenkeren egentlig inneholder og betyr. For å gjøre analyseprosessen enklere ble det laget en tabell med hvert punkt og en kort forklaring for punktet, inspirert av Dragland og Ebert (2022, ss. 33-34):

Kategori	Operasjonalisering
Logikk	Forklare hvorfor noe skjer eller kommer til å skje på bakgrunn av en forutsigbar respons.
Algoritmer	En sekvens av instruksjoner eller et ett regler for å få noe gjort. Disse sekvensene skal danne en fremgangsmåte for å løse et problem.
Dekomposisjon	Dele opp problemet i mindre deler, løse delproblemene sammen for å gi en løsning på det overordnede problemet.
Mønstre	Se etter likheter og ulikheter for å videre danne regler og løse generelle problemer.
Abstraksjon	Forenkle gjennom å hente ut den informasjonen som er relevant, og samtidig kvitte seg med irrelevant informasjon.
Evaluering	Vurdere hva, hvordan og hvorfor en har programmert, og vurdere kvaliteten og effektiviteten på løsningen opp mot formålet.
Fikle	Løse problemer gjennom direkte utforskning og eksperimentering. Prøve seg frem og teste løsninger.
Skape	Forberede allerede eksisterende produkter eller skape noe nytt.
Feilsøke	Identifisere og deretter rette opp feil gjennom systematisk testing og modifisering.
Holde ut	Fortsette og prøve å finne en løsning, selv når en møter på utfordringer.
Samarbeide	Vurdere ulike løsninger og diskutere seg frem til en hensiktsmessig løsning.

Tabell 3 Beskrivelser for analyse

Ved hjelp av disse beskrivelsene gikk fase tre ut på finne kjennetegn på disse temaene, i form av sitater og handlinger og situasjoner fra elevene, og deretter samle disse under et tema. Det var ikke alt som ble observert i direkte sitater, men i situasjoner. Derfor ble det valgt å beskrive disse situasjonene istedenfor å skulle prøve å finne sitater om det.

Et eksempel for hvordan noen av kodene ble samlet under et potensielt tema:

Nøkkelbegrep	
Evaluering: Vurdere hva, hvordan og hvorfor en har programmert, vurdere kvaliteten og effektiviteten på løsningen opp mot formålet.	Gruppe 5 Elev 1: Oi, det var alt for langt. [Elev 2 henter bilen og prøver på nytt] Elev 2: Men den snur seg jo ikke. Elev 1: Det var alt for lite grader, den snudde seg, jeg så det. Det var alt for lite grader. Elev 2: Ja, men det er jo ingen grader da (i programmet). Elev 1: Kanskje vi må ta 2000 da, men den kjørte jo litt for langt også.
	Gruppe 3 Elev 1: Det var litt for mye. Elev 2: Det var alt for mye, men den hadde bra kjøretid da. [Elev 2 endrer i programmet] Elev 1: Men 3000 var jo bra da (i kjørelengde) Elev 2: Nei, vi må ha 3050. [Endrer til 3050] Elev 1: Nei
	Gruppe 2 Elev 1: Litt mindre snurring og litt lengre? Elev 2: Litt mye snurring ja. Elev 1: Og litt lengre kjøring.
	Gruppe 7 Elev 1: Men vi må sette på.. Vi må sette inn sånn der.. Eh.. At den må.. Justere til venstre, fordi den kjørte nesten bare til høyre. Elev 2: Hmm? Elev 1: Den kjørte jo bare [lager en armbevegelse mot høyre]

Tabell 4 Eksempel på kategorisering.

Videre i den neste fasen av analysen ble det gjennomgått og definert hovedtema. Hovedtemaene ble som følger: Nøkkelbegreper og arbeidsmetoder, med flere underkategorier.

3.6 Feilkilder

Når det kommer til validiteten i studien, sier Larsen (2007) at validitet handler om samsvar mellom den teoretiske og den operasjonelle definisjonen, altså at man samler inn data som er relevant til problemstillingen.

Observatørens roller kan gi noen avvik og feilkilder når det kommer til resultatene i datasettet som innsamles. Hvor involvert observatøren var under selve datainnsamlingen kan ha stor påvirkning med tanke på hvor reel og troverdig dataene ble. Hvis observatøren hadde blandet seg for mye og hjulpet elevene, kunne det ført til store feilkilder i datainnsamlingen, og ikke ført et realistisk resultat i forskningen.

Andre faktorer som kan føre til feilkilder kan være hvis forskeren deler for mye informasjon om studien til kandidatene, som fører til at det påvirker resultatet av datainnsamlingen. I dette tilfellet fikk kandidatene kun informasjon om at det skulle forskes på oppgavene lærerne gir,

og hvordan det får dem til å tenke. Kandidatene fikk bekreftet at det ikke var dem som individ som ble forsket på, noe som førte til at flere meldte seg på studien.

Noe annet som kan ha ført til feilkilder i datasettet, var at det virket som at eleven ikke var klar over at GoPro-kameraene gjorde lydopptak samt at de filmet. Når elevene enten spurte eller ble klar over det selv, ble de stillere og mer bevisste hva de sa. Det kunne ført til feilkilder, med tanke på at det kunne begrense datasettene, og gjøre at elevene holdt mer tilbake av hva de tenkte.

Batterikapasiteten på GoPro-kameraene kunne også være en faktor til feilkilder. På grunn av at kameraene ble brukt i fire skoletimer på samme dag, var det viktig at de ble ladet i friminuttene og når de ikke var i bruk. Selv om dette ble gjort, var det ikke alle kameraene som hadde batterikapasitet til å vare gjennom hele de to siste timene. Det endte opp i at noen av videoopptakene ble kuttet i midten av timen, noe som kan gi feilkilder i datasettet.

3.7 Forskningsetikk

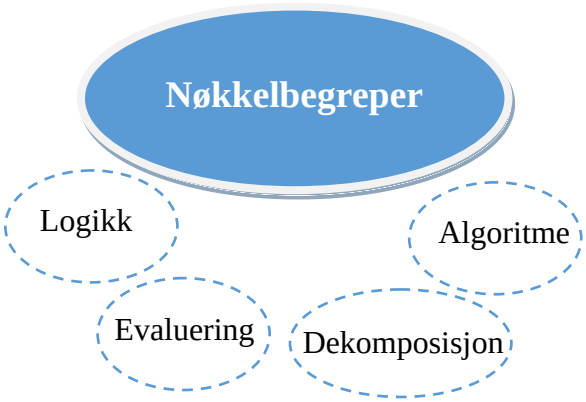
Alle som jobber innen forskning kommer innom etiske dilemmaer, og alle som gjennomfører et forskningsprosjekt må ta stilling til etiske prinsipper (Larsen, 2007). Med tanke på problemstillingen er det ikke spesielle etiske dilemmaer man trenger å ta hensyn til. Der hvor det oppstår etiske prinsipper er under datainnsamlingen og bruk og formidling av datamaterialet (Larsen, 2007). Nyeng (2012) beskriver at det etiske saksområdet i forskning grovt sett kan deles i to: Forskningsinterne regler og normer, og forskningseksterne vurderinger. Det som inngår i forskningsinterne regler og normer er at man skal være saklig, åpen og redelig internt i et forskersamfunn, noe som vil si hvordan forskning skal gjennomføres og rapporteres (Nyeng, 2012). Hvis man skal se på dette i lys av denne studien, kan måten det gjennomføres på, altså datainnsamlingen, skape etiske dilemmaer med tanke på at datainnsamlingen skjer gjennom videoopptak av elever som løser oppgaver. Derfor, for å få det på den etisk riktige siden, ble det sendt inn et meldeskjema til Sikt, om studien og forskningsmetoden.

4. Resultater

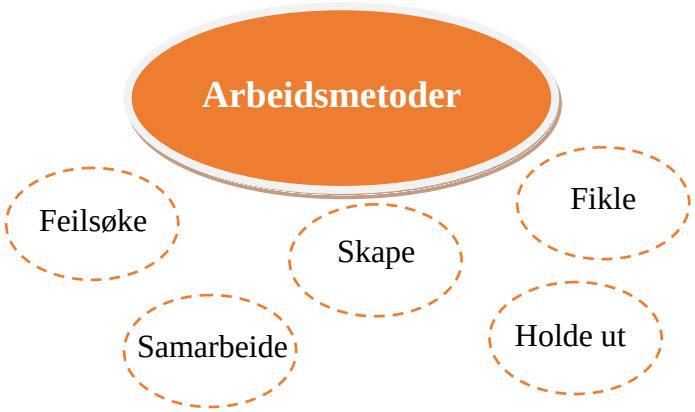
I denne studien gikk undersøkelsen ut på å finne ut om elever preges av algoritmisk tenking ved løsning av programmeringsoppgaver. For å gjøre det enklere ble det konkretisert forskningsspørsmål som:

1. Hvilke uttrykk og kjennetegn av de seks nøkkelbegrepene blir uttrykt hos elever under oppgaveløsning i programmering?
2. Hvilke uttrykk og kjennetegn av de fem arbeidsmetodene blir uttrykt hos elever under oppgaveløsning i programmering?

For å få belyst problemstillingen: «På hvilken måte preges elevers aktivitet av algoritmisk tenking ved løsning av programmeringsoppgaver?» ble det, som nevnt tidligere, utarbeidet to hovedtema i analysen, nøkkelbegreper og arbeidsmetoder. Grunnen til at disse ble valgt, var fordi de ble som paraplyer over de forskjellige undertemaene som ble funnet. Ikke alle temaene innenfor den algoritmiske tenkeren ble gjenkjent i datamaterialet, men i figur 5 og 6 ser man hvilke kategorier som ble definert.



Figur 5 Nøkkelbegreper med undertema



Figur 6 Arbeidsmåter med undertema

4.1 Hvilke uttrykk og kjennetegn av de seks nøkkelbegrepene blir uttrykt hos elever under oppgaveløsning i programmering?

Hovedtemaet nøkkelbegreper omhandler, som tidligere nevnt, hvordan elever bruker hodet i oppgaveløsing. Ikke alle temaene innenfor den algoritmiske tenkeren ble gjenkjent i datamaterialet, men de som det ble funnet eksempler på var i hovedsak logikk, algoritme, dekomposisjon og evaluering når det kommer til nøkkelbegrepene, som vist i figur 5 .

4.1.1 Logikk

Under punktet logikk ble det, ut ifra tabell 3, satt en forklaring for å gjøre det enklere i analyseprosessen å sette ring rundt sitater og handlinger og kunne begrunne at det var tegn til bruk av logikk i den situasjonen.

Logisk tenkning ble brukt jevnlig gjennom hele datamaterialet mens elevene prøvde å få Bit:Boten til å kjøre en meter, for så å snu og kjøre tilbake igjen. Når gruppe 8 først begynte å løse oppgaven, begynte de å diskutere hvordan oppgaven skulle løses. Selv om ingen av dem hadde brukt Bit:Bot før ble det, gjennom observasjon fra videoklippene, åpenbart for elevene hvor de skulle lete etter de forskjellige funksjonene for Bit:Boten. I diskusjoner og handlinger i gruppa, satt de to og så på samme skjerm og sa:

> *Elev 2:«Ok, vi skal få bilen til å kjøre en meter, snu 180 grader, og kjøre tilbake til mål.»*
> *Elev 1 telte til fem mens Bit:Boten ble skyvet langs målebåndet.*
> *Elev 1: «Ja, det blir fem sekunder, skriv fem sekunder.»*
> *Elev 2: «Nei, jeg tror ikke det er fem sekunder, for du telte for sakte.»*

Videre mens de diskuterte, brukte de Bit:Boten til å vise det de trodde kom til å skje. I denne samhandlingen viste elevene tegn til logiske resonnement, fordi de forklarer hva som kommer til å skje, til hverandre. Elev 2 forklarte det som at hvis de bruker fem sekunder i programmet, vil Bit:Boten kjøre for langt, fordi elev 1 telte for sakte.

Etter at de hadde kommet frem til hvor lenge Bit:Boten måtte kjøre for å kjøre en meter, kom sitater som:

> *«Ok, nå kjører den rett frem. Da må vi få den til å snu 180 grader, for å kjøre tilbake, men den kan ikke snu like lenge som bilen kjørte. Det blir for lenge. Kanskje 400 (millisekund)?»*

Her kan det virke som at de bruker erfaringen fra hvor lenge Bit:Boten måtte kjøre for å nå en meter, til å bestemme hvor lenge Bit:Boten måtte ha mye mindre tid for å snu 180 grader.

4.1.2 Algoritme

I tabell 3 ble algoritme definert som «En sekvens av instruksjoner eller et sett regler for å få noe gjort. Disse sekvensene skal danne en framgangsmåte for å løse et problem. I oppgavene elevene fikk i time tre og fire, skulle de lage programmer for å få en bil til å kjøre en meter og tilbake, samt lage robotstøvsuger/-gressklipper. I begge oppgavene måtte elevene lage framgangsmåter for å finne ut hvordan man skulle lage programmet. Ingen av gruppene lagde framgangsmåten på ark eller lignende, men de lagde det etter hvert mens de lagde programmet. For det meste diskuterte gruppene hva som måtte skje først og sist. Gruppe 2 valgte å leke litt med lysene på Bit:Boten, og diskuterte hvor den koden måtte stå i programmet for å bli riktig.

> *Elev 1: «Jeg tror koden må stå først i programmet.»*
> *Elev 2: «Nei, den skal jo lyse, da må den stå her.»*
> *Elev 2 peker nederst på koden i programmeringsfeltet.*
> *Elev 1: «Nei, den skal jo lyse, men hvis den skal lyse så må koden være her.»*
> *Elev 1 peker øverst i programmet, over kodingen for kjøring*.

Etter hvert med litt testing kom de fram til at koden måtte stå først hvis bilen skulle kjøre med lysene, men fant også ut at de kunne ha enda en kode, som gjorde at bilen endret farge når den var ferdig å kjøre.

4.1.3 Dekomposisjon

Dekomposisjon ble definert som «Dele opp problemet i mindre deler, løse delproblemene sammen for å gi en løsning på det overordnede problemet.» i tabell 3. Når det kommer til dekomposisjon i datamaterialet var det ingen av gruppene som definerte eller forklarte at å dele opp hovedproblemet i oppgaven, ville gjøre det enklere å løse. Hvis man ser bort i fra forståelsen for å forklare om dekomposisjon, benyttet alle gruppene seg av det steget likevel.

Gruppe 1 hadde en diskusjon som gikk slik:

> *Elev 1: «Sånn, men ok, hva skal den egentlig gjøre?»*
> *Elev 2: «Den skal kjøre framover og snu unna slik at den ikke krasjer i noe.»*
> *Elev 1: «Hvordan kan vi gjøre det da? Den må hvert fall ha kjøring.»*
> *Elev 2: «Ja, og så må den ha snuing, slik at den ikke krasjer.»*

Her ser man at elevene deler inn hovedoppgaven, som er å få Bit:Boten til å kjøre som en robotstøvsuger/-gressklipper, i mindre deloppgaver uten at de kanskje er klar over det selv.

Elev 1 kommer fram til at de må ha med at den skal kjøre, hvis ikke det er med, beveger ikke Bit:Boten seg. Elev 2 kommer fram til at de må også ha med delen som svinger unna objekter og hindringer, slik at Bit:Boten ikke ender opp med å sitte fast i for eksempel et hjørne.

4.1.4 Evaluering

Evaluering gjenkjennes ut ifra tabell 3 som å «Vurdere hva, hvordan og hvorfor en har programmert, og vurdere kvaliteten og effekten på løsningen opp mot formålet» Et mønster som ble funnet var at ofte etter elevene hadde brukt logisk sans, og testet ut det de hadde programmert, måtte de evaluere hvordan programmet endte. Ved første test av programmet ble det sjelden suksess, da det var første gang de prøvde seg med Bit:Bot, men samtalene rundt resultatene var som oftest like mellom gruppene. Da gruppe 9 testet programmet sitt for første gang endte det opp i denne samtalen:

> *Elev 1: «Oi, den kjørte alt for kort! Kanskje vi må justere farten slik at den kjører lenger?»*
> *Elev 2 henter bilen og setter den ved startlinjen igjen.*
> *Elev 2: «Ja, kanskje 4000 (millisekunder)?»*
> *Elev 1 endret farten til 4000 millisekunder, og elev 2 lastet over programmet og prøvde på nytt*
> *Elev 2: «Nå har vi for lang tid (i programmet). Bilen kjører for langt. Da må vi endre farten til mindre.»*

Det som kan observeres fra denne samtalen mellom elevene er at de forstår hva programmet gjør, siden de forstår at de må justere opp farten for å få Bit:Boten til å kjøre lenger. På grunn av at elevene forstår at det er farten som må justeres på, kan det være et tegn til at de forstår hvordan programmet skal fungere.

Gruppe 5 prøvde å programmere de to første stegene i programmet samtidig, det vil si kjørelengde og snu, for å slå to fluer i en smekk på testingen. De justerte først inn kjørelengden slik at den ble riktig, og deretter begynte de å justere på å få Bit:Boten til å snu 180 grader. Bit:Boten opp med å svinge alt for lenge:

> *Elev 2: «Det (tiden på svingen) var alt for lenge, men den hadde bra kjøretid nå da.»*
> *Elev 1 :«Ja, den snurret jo bare. Den skal jo bare snu 180 grader, men det går ikke an å bestemme grader i programmet.»*
> *Elev 2: «Da må vi prøve å ha på mindre tid. Det vi har nå er hvert fall alt for mye.»*

Her evaluerer begge elevene godt om hvordan kjøretiden og svingen skal gjøres, og elev 1 viser forståelse for hva programmet skal gjøre, men viser frustrasjon over hvordan de skal få Bit:Boten til å snu akkurat 180 grader.

4.2 Hvilke uttrykk og kjennetegn av de fem arbeidsmetodene blir uttrykt hos elever under oppgaveløsning i programmering?

Hovedtemaet arbeidsmetoder, går som tidligere nevnt ut på hvordan elever bruker kroppen sammen med hode for å løse oppgaver. Når det kommer til hvilke arbeidsmetoder som ble funnet i datamaterialet, var det fikle, skape, feilsøke, holde ut, samarbeide, som vist i figur 6. Det vises derfor at elever bruker for det meste alle arbeidsmetodene i oppgaver som omhandler programmering.

4.2.1 Fikle

Å fikle går ut på å løse problemer gjennom direkte utforsking og eksperimentering, og å prøve seg frem og teste forskjellige løsninger, tabell 3. Når elevene først fikk utlevert Bit:Botene, begynte de å sette de sammen og koble micro:bitene opp mot iPadene sine. Når de fikk oppgaven om å få bilen til å kjøre en meter, snu og kjøre tilbake, var det mange av gruppene som ikke klarte det på første forsøk. Gruppe 4 fikk bilen til å kjøre, men den hadde en helling mot venstre, slik at den ikke kjørte rett mot det punktet den skulle snu på.

> *Elev 1: «Den kjører skeivt.»*
> *Elev 2: «Ja, den gjør det. Hvorfor det?»*
> *Elev 1: «Jeg vet ikke, kanskje dekket ikke sitter riktig?»*
> *Elev 2 tar av begge dekkene og bytter plass på dem.*
> *Elev 2: «Ok, prøv nå.»*
> *Elev 1 prøver programmet igjen.*
> *Elev 2: «Nei, det virka ikke. Hva annet kan det være da?»*

Her ser man derfor tegn til at elevene i gruppe 4 fikler med Bit:Boten, da de utforsker om det er hjulet som er problemet for at den kjører skeivt.

Gruppe 6 utforsket og eksperimenterte på hvordan de skulle få bilen til å svinge riktig.

Elev 1: «Ok, den skal svinge 180 grader, men vi kan bare velge fart i prosent.»
Elev 2: «Ja, men da prøver vi snu til venstre i fart 60 prosent.»
Elev 2 tester programmet.
Elev 3: «Oi! Det ble ikke riktig. Den kjørte en meter, men snurrer bare.»
Elev 1: «Ja, men hvordan kan vi stoppe den? Stoppe snurringen liksom?»
Elev 3: «Kanskje legge inn en pause?»

Her utforsker gruppa hvordan svingfunksjonen på Bit:Boten fungerer, og fortsetter å prøve seg frem ved å finpusse programmet.

4.2.2 Skape

Egenskapen med å skape går ut på å kunne forbedre allerede eksisterende produkter eller skape noe nytt. I hovedsak var alle oppgavene gitt med et allerede påstartet program. Derfor fortsatte alle gruppene med å forbedre et allerede eksisterende produkt som var halvferdig.

Gruppene hadde veldig varierende framgangsmåte på hvordan de skulle gjøre ferdig programmet for å få Bit:Boten til å fungere som en robotstøvsuger/-gressklipper. Etter gruppe 2 klarte å gjøre seg ferdig med grunnprogrammet for Bit:Boten, fikk de spørsmålet om de klarte å få programmet til å kjøre jevnere, slik at Bit:Boten ikke krasjet i objekter når den skulle snu seg, altså forbedre programmet de allerede hadde laget.

Elev 2: «Vi kan sette inn denne (stopp med bråstopp). La oss prøve.»
Elev 2 setter i gang programmet.
Elev 1: «Den stopper og snur, men den krasjer enda hvis det står noe i veien rundt.»
Elev 2: «Det ser ut som at den kjører for fort. Prøv å senk farten.»

Etter hvert klarte de å forbedre programmet slik at Bit:Boten kjørte sakte og rolig framover, og stoppet med god margin og snudde seg sakte. De diskuterte sammen at for å få den til å kjøre best mulig, måtte det gå saktere, selv om det var mer gøy at det gikk fort. Dette gjorde at Bit:Boten ikke krasjet i noe, da den snudde sakte nok til å registrere andre hindringer i veien foran.

4.2.3 Feilsøke

Egenskapen å kunne feilsøke blir beskrevet i tabell 3 som å identifisere og deretter rette opp i feil gjennom systematisk testing og modifisering. For elevene var feilsøking en naturlig del av deres programmeringsprosess, og det var noe alle gruppene benyttet seg av. Gruppe 6, fortsettelse av eksemplet over, slet med å få Bit:Boten til å svinge riktig og få den til å stoppe. Hvor det andre eksemplet stoppet, tenkte elevene å legge inn en pause slik at snurringen stoppet etter bilen hadde snudd 180 grader. Etterpå skjedde dette:

> *Elev 3: «Kan jeg se skjermen litt? Herregud kan vi ikke bare bruke denne? Blir ikke det lettere?»*
> *Elev 3 peker på «snu til venstre med fart __ prosent i __ millisekunder».*
> *Elev 1: «Oi, ja prøv den.»*
> *Elev 3 tester programmet.*
> *Elev 2: «Nå stopper den hvert fall. Nå må vi bare finne hvor lenge den skal snurre.»*

I dette eksemplet viser elevene at de kjenner programmet og kodene godt nok til å finne feil i kodesettet og korrigerer feilen med å finne en bedre kode til erstatning. Videre fortsatt gruppe 6 slik:

> *Elev 1: «Prøv farten på 500 (millisekund), den skal jo ikke snu så lenge.»*
> *Elev 3 tester programmet*
> *Elev 2: «Den snur jo ikke.»*
> *Elev 1: «Da må tiden være for kort. Prøv mer da ta 600.»*
> *Elev 3 tester programmet.*
> *Elev 1: «Det ble fortsatt for lite.»*
> *Elev 2: «Prøv 1000 da.»*
> *Elev 3 tester programmet.*

Her viser gruppa at de systematisk tester forskjellige tider for å finne ut den ideelle tiden for å få Bit:Boten til å snu 180 grader. De viser også forståelse for at de må modifisere koden for å få til det resultatet oppgaven ber om. Måten man ser det på er at elevene aktivt går inn og endrer i koden slik at det blir mer og mer spesifikt for hver test som kjøres.

4.2.4 Holde ut

Egenskapen om å holde ut handler om å fortsette og prøve å finne en løsning, selv om man møter utfordringer i oppgaven. Derfor er det viktig at elevene velger å ikke gi opp selv om man møter litt motstand i oppgaven. Alle gruppene viste en form for utholdenhet. Et eksempel fra gruppe 1, og hvordan de viste utholdenhet, var da de skulle få sensoren til Bit:Boten til å registrere hindringer foran seg:

> *Elev 1: «Vi må prøve noe annet. Dette virker jo ikke.»*
> *Elev 2: «Ja, kanskje, men hva må vi endre da?»*
> *Elev 1: «Hva står programmet på nå? Hva gjør at den ikke stopper?»*
> *Elev 2: «Det står at den skal snu når den merker en hindring innen 10 cm.»*

Det elevene ikke la merke til var at koden fortalte Bit:Boten at hvis den leste av en hindring på under 10 cm, skulle den kjøre framover, hvis ikke skulle den snu konstant til venstre.

> *Elev 1: «Hva skjer hvis vi endrer < til =, kanskje det virker da?»*
> *Elev 2 endret programmet, og de testet.*
> *Elev 2: «Nei, det virket ikke. Jeg skjønner ikke hva som er galt.»*
> *Elev 1: «Ikke jeg heller. Skal vi prøve å starte på nytt?»*
> *Elev 2: «Men da mister vi alt vi allerede har laget.»*
> *Elev 1: «Ja, men det vi allerede har laget virker jo ikke, vi må prøve noe annet.»*

Her viser begge elevene frustrasjon over at programmet deres ikke fungerer, selv om de prøvde å justere det programmet de allerede hadde laget. Det virket som at elev 1 prøvde å holde motet oppe, og ville prøve å løse oppgaven. Ved å foreslå å starte hele kodeprosessen på nytt, virker det som at det gjorde at elev 2 ble mindre motivert til å fortsette med oppgaven, da de måtte gjøre alt arbeidet en gang til. Etter hvert fikk gruppa litt veiledning og programmet ble lest høyt for dem. Da fant de ut at de hadde plassert de to kodene feil, og endret plasseringene. Ved neste test ble det derfor suksess.

4.2.5 Samarbeide

Samarbeid mellom elevene går ut på at de klarer å vurdere ulike løsninger og diskutere seg frem til en hensiktsmessig løsning. De bør klare å bruke hverandres kunnskaper og erfaringer for å gjøre prosessen med oppgaveløsning enklere og mer effektiv. Resultatet av godt samarbeid er forhåpentligvis at elevene lærer av hverandres kunnskap og gruppas erfaringer. For å ha et godt samarbeid er det viktig at elevene klarer å vente på tur, lytte til hverandres

meninger og inngå kompromisser om nødvendig. Det handler om å ha en «oss» følelse, og ville løse oppgaven sammen.

Det var varierte preg over gruppene når det kom til samarbeid. Noen av gruppene hadde godt samarbeid mellom seg, mens andre hadde en dynamikk hvor enten det var en elev som tok kontroll, eller en elev meldte seg ut. For det meste var det et visst samarbeid i alle gruppene i starten, men hos noen av dem dabbet det av på grunn av en av de to grunnene nevnt over.

Et eksempel på ei gruppe som hadde godt samarbeid jevnt over begge timene var gruppe 3. De hadde gode diskusjoner sammen som inneholdt å lytte til hverandre og inngå kompromiss, som dette:

> *Elev 2: «Vi prøver det vi har laget nå, og så ser vi hvordan det går.»*
> *Elev 1 tester programmet.*
> *Elev 1: «Ok, den snur ikke, men den kjører. Har du noe forslag til hvordan vi kan få den til å snu?»*
> *Elev 2 ser på skjermen.*
> *Elev 2: «Jeg syns det er vanskelig å se hva som mangler eller kanskje virker. Kan du komme og se sammen med meg?»*

Her vises det at det er en trygghet i gruppa, med tanke på at elev 2 kan være åpen om at det er utfordrende å gjøre kodingen alene, og foreslår at det blir enklere å løse det sammen. De jobbet videre for å finne ut løsningen av problemet, og klarte dette gjennom gode diskusjoner og forslag.

I gruppe 7 ble det observert en diskusjon som gikk på kompromiss i gruppa.

> *Elev 1: «Kan du ikke bare se om bilen kjører 1 meter med tiden 4000 (millisekunder)?»*
> *Elev 2: «Nei, det må bli for mye. Det er ikke noe vits.»*
> *Elev 1: «Men hvorfor kan vi ikke prøve? Vi vet jo ikke før vi har sett det.»*
> *Elev 2: «Fordi jeg tror ikke det er noe vits.»*
> *Elev 1: «Dette er en gruppeoppgave, vi skal jo prøve det begge mener.»*
> *Elev 2: «Ok da, men jeg vet det kommer til å bli for mye.»*
> *Elev 2 tester programmet*

Det viste seg i dette eksemplet at elev 1 hadde riktig i forhold til at 4000 millisekunder ble for lang tid for Bit:Boten å kjøre, men det var viktig for elev 2 at det ble inngått et kompromiss og bli hørt i gruppearbeidet.

5. Diskusjon

Hensikten i denne studien var å undersøke på hvilken måte elevers aktivitet preges av algoritmisk tenkning ved løsning av programmeringsoppgaver. I dette kapittelet skal resultatene fra analysen diskuteres og drøftes opp mot teori som tidligere er presentert i studien. Diskusjonen blir regulert av disse forskningsspørsmålene:

1. Hvilke av de seks nøkkelbegrepene bruker elever under oppgaveløsning i programmering?
2. Hvilke av de fem arbeidsmetodene bruker elever under oppgaveløsning i programmering?

5.1 Kjennetegn på nøkkelbegreper i den algoritmiske tenkeren

Resultatene av analysen viser at elevene benyttet seg av logikk, algoritme dekomposisjon og evaluering. Disse vil i hovedsak bli diskutert hver for seg i underkapitler, men noen strategier vil bli nevnt i drøftingen av andre strategier der det faller naturlig.

5.1.1 Logikk – Evnen til å forutse og forklare hva som vil skje

Da gruppe 8 diskuterte hvordan de skulle løse oppgaven med å få Bit:Boten til å kjøre en meter og tilbake, brukte elev 2 et matematisk begrep som ikke var nevnt da de fikk oppgaven, for å forklare hvordan oppgaven skulle løses. Eleven sa at Bit:Boten måtte snu 180 grader for å kjøre tilbake. Her kan det derfor virke som at elev 2 viser bruk av god forståelse for logisk tenkning, med tanke på at eleven prøver å forutse hva oppgaven må inneholde for å få løst oppgaven. I følge Barefoot Computing (u.å.h) er dette en god egenskap innenfor deres definisjon for begrepet logikk. Måten eleven bruker dekomposisjon for å forutse hvordan oppgaven må gjøres, gir forutsigbarhet til å analysere hva som må til for å produsere en kode for oppgaven. Dette kan stemme over ens med måten Barefoot Computing (u.å.h) definerer logikkbegrepet.

Utdanningsdirektoratet (2019) kommer inn på at de viktige faktorene for logikk innenfor algoritmisk tenking er analyse og forutsigbarhet. Derfor virker det som at evnen til å analysere og evnen til å forutse hva som vil skje, er konkret det Utdanningsdirektoratet ser etter innenfor logikkbegrepet. Gruppe 8 prøvde å analysere det som kom til å skje i oppgaven

gjennom framgangsmåten elev 2 valgte, for å prøve å bestemme antall sekunder programmet måtte ha for å få Bit:Boten til å kjøre en meter. Som nevnt i resultatet dyttet elev 1 Bit:Boten langs målebåndet samtidig som eleven telte til fem. Ved å gjøre dette kan det virke som at elev 1 prøver å eksperimentere seg fram for å få forutse hvor lang tid Bit:Boten trenger for å kjøre en meter. Egenskapen om å eksperimentere seg fram for å finne en løsning til hvordan koden i et program kan fungere, er en viktig del av logisk tenkning (Barefoot Computing, u.å.h).

Videre bruker gruppa erfaringen av hvor lang tid Bit:Boten brukte for å kjøre en meter, til å prøve å bestemme hvor lenge snu-kommandoen måtte vare for at Bit:Boten skulle snu 180 grader. Her brukte elevene logikk til å analysere tallene i den første deloppgaven (kjør en meter), og prøvde å videreføre de erfaringene de hadde gjort seg tidsmessig, til det neste delproblemet (snu 180 grader), slik at den kanskje blir løst like effektivt, om ikke mer effektivt. I resultatet fra analysen hadde alle gruppene på generell basis god forståelse for at hvis Bit:Boten skulle snu, måtte den snu 180 grader for å bli riktig.

Resultatet viste at det kunne være utfordrende for elevene å forklare hvorfor koden i et program gir det resultatet som det gir. I slike undervisningstimer som datainnsamlingen ble gjort i, er det naturlig at gruppene vandrer litt mellom hverandre og diskuterer det de har klart å produsere. Det ble observert at hvis to grupper hadde løst oppgaven ved forskjellige koder, men oppnår likt mål, klarte ikke gruppene å forklare konkret hvorfor deres program ga riktig resultat. Med tanke på at elevene ikke var veldig drevne innen programmering kan det være grunnen, ellers kan det rett og slett være på grunn av at de ikke forstår programmet.

5.1.2 Algoritme – Finne en rekkefølge steg-for-steg

I resultatene angående algoritmer var det ikke veldig mange eksempler på hvordan elevene var preget av bruken av algoritmer. Barefoot Computing (u.å.b) mener at algoritmer er et sett med regler for å få noe gjort. Med tanke på eksemplet i 4.1.2 hvor elevene diskuterer hvor en kode-blokk skal stå i programmet for å få riktig utfall, følger de ingen regler for å finne fram til dette. Elevene prøver seg fram for å få det resultatet de ønsker å oppnå. Derfor følger ikke elevene direkte det Barefoot Computing (u.å.b) mener er algoritmer, men hvis man ser på at elevene må følge regler for å benytte seg av det.

Man kan også diskutere om algoritme i det hele tatt ble brukt av elevene, med tanke på at elevene var relativt ferske når det kom til bruk av micro:bit og blokkprogrammering. Det kommer helt an på hvordan man tolker resultatene. Med tanke på at den algoritmiske tenkeren

viser forståelse for underpunktene som beskrives, kan det være vanskelig å vise om elevene faktisk forstår og bruker algoritmer bevisst når de jobber med programmering, eller om de bare «plotter» inn kommandoer der de passer inn uten at elevene trenger å tenke over det.

Det kan være vanskelig å begrunne om algoritmer ble bevisst brukt av elevene, men under observasjonen av videoklippene var det tydelig tegn for at elevene hadde bestemt seg for en rekkefølge som kodene skulle settes i, det virket ikke som at de ble tilfeldig plassert. Hvis en kode blir sluppet og havner på det elevene mente var feil plass i rekkefølgen, var det lett å se at de hadde en formening om hvor koden opprinnelig var tenkt. På denne måten kan det begrunnes at resultatene fra analysen viser at de fleste av gruppene benyttet seg av algoritmer i en viss grad under oppgaveløsningen.

I eksemplet med gruppe 2, er det åpenbart at de forstår at rekkefølge har noe å si for hvordan koden utspiller seg i Bit:Boten. Med tanke på Wings (2006) mening om algoritmisk tenkning, som går ut på å bygge et system som er funksjonelt slik at en datamaskin klarer å lese programmet, blir det en mer relevant forklaring for hvordan gruppe 2 gikk fram for å få riktig program. Elevene diskuterte hvor koden måtte være plassert for å få det resultatet de ønsket, men de fulgte ikke en regel som Barefoot Computing (u.å.b) mener de måtte gjøre. Videre i diskusjonen viste det seg at elevene kom fram til at for å få bilen til å ha farge gjennom kjøringen, så måtte koden som bestemte dette, være før kjørekodene.

Ut ifra disse resultatene kan det være et tegn på at elever som er mer bevisst på hvor kodene må være for å få riktig rekkefølge, har lettere for å forstå hvordan oppgavene skal løses. Elevene som viste store tegn til å forstå hvorfor kodene måtte plasseres i en bestemt rekkefølge hadde mindre feil som måtte rettes eller slettes fra sine programmer. På denne måten kan man se nødvendigheten i at elever lærer seg om viktigheten i rekkefølge og være klar over hvorfor koder blir plassert der de blir, i et program.

5.1.3 Dekomposisjon – Dele opp problemet i mindre delproblemer

Alle gruppene brukte en form for dekomposisjon i sine oppgaveløsninger, men som i eksemplet vist i resultatet, viste gruppe 1 at de brukte dekomposisjon som del av fremgangsmåten for å komme i gang med oppgaven. Elev 2 viste at det å bryte ned oppgaven til mindre deloppgaver ikke var en utfordring. Elev 1 supplerte med å spesifisere hvilken deloppgave som måtte gjøres først. Måten gruppe 2 benyttet seg av dekomposisjon på, stemmer med hvordan både Barefoot Computing (u.å.f) og Utdanningsforbundet (2019) definerer begrepet dekomposisjon. Begge beskriver dekomposisjon, som at elever forstår

hvordan man skal bryte ned komplekse problemer til mindre og enklere problemer å løse (Barefoot Computing, u.å.f; Utdanningsdirektoratet, 2019).

Ved dekomposisjon kan man også stille seg det samme spørsmålet som ble gjort under algoritme, altså om elevene faktisk er klar over at de bruker dekomposisjon, eller om det bare er en «følgefeil» av informasjonen elevene får for å løse oppgavene. Ut ifra resultatet virker det som at dekomposisjon ikke er tilfeldig, og at de fleste gruppene benyttet seg av å dele opp alle de større oppgavene de ble tildelt. Elevene delte ikke opp arbeidet på måten Barefoot Computing (u.å.f) beskriver det, som at en elev 1 løser første problem av oppgaven og elev 2 løser det andre problemet. Det kan det være flere grunner til at de ikke gjorde. For det første var det et begrenset antall Bit:Boter tilgjengelig, som gjorde det mer utfordrende for elevene å teste programmet sitt for hvert delproblem. En annen grunn til at elevene kanskje valgte å løse delproblemene sammen var at dette var første gang elevene hadde jobbet med Bit:Boter, som kanskje gjorde det tryggere for elevene å løse hvert delproblem som gruppe.

En annen gruppe som benyttet seg av dekomposisjon var gruppe 8, selv om eksemplet ligger under logikk, vises det godt at gruppa benyttet seg av å dele opp oppgaven i mindre deloppgaver. Ut ifra sitatene ble det beskrevet at de hadde fått Bit:Boten til å kjøre en meter, og deretter diskuterte hvordan de skulle få den til å snu, for så å kjøre tilbake.

Med tanke på resultatene ser det ut til at de elevene som benyttet seg av dekomposisjon, løste oppgavene enklere og mer effektivt enn elevene som ikke gjorde det. Ut ifra det Barefoot Computing (u.å.f) mener om dekomposisjon, stemmer det, da de mener at dekomposisjon gjør det enklere for elever å håndtere store prosjekter. Dette ble bekreftet da elevene som i delvis benyttet seg av å dele inn i deloppgaver ofte trengte mer hjelp fra andre medelever og lærere. Elevene som hadde delt opp oppgaven i så mange deloppgaver som mulig viste tegn til mer selvstendighet og trygghet i oppgaveløsingen, selv om de ikke delte opp arbeidet med å fordele deloppgavene.

5.1.4 Evaluering – Vurdere eget arbeid

Evaluering var noe elevene viste et høyt kunnskapsnivå om, hvert fall kunne det tolkes slik ut ifra resultatene. Alle gruppene benyttet seg av evaluering, enten de var klar over det eller ikke. Under observasjonen kunne det virke som at evalueringsevnen falt naturlig for elevene når de utforsket hvordan kodene skulle være i programmet, som vist i eksempelet med gruppe 9. De brukte evaluering gjennom hele testprosessen, og viser preg av gode vurderingsevner i diskusjonen, som videre førte til gode forbedringer i programmet. Ut ifra

Utdanningsdirektoratets (2019) beskrivelse av begrepet evaluering gjør elevene det som er forventet for den algoritmiske tenkeren, da de mener at elever som kan gjøre vurderinger, evaluerer arbeidet.

Det er ikke noe i resultatet som tilsier at elevene trengte veiledning for å huske og bruke evaluering. Da viser resultatet heller at noen grupper trenger hjelp med tankeprosessen videre etter evalueringen. Ut ifra måten Barefoot Computing (u.å.g) beskriver evalueringsprosessen på, altså gjøre vurderinger der det er mulig, på en objektiv og systematisk måte, klarte noen grupper dette helt uten problem, mens andre kunne ha noen utfordringer med å gjøre det systematisk, som gruppe 5. Gruppe 5 viste at de hadde gode evalueringer etter hver test som ble gjort på programmet, men de valgte og prøve å kode to steg samtidig. Det kan sees på som å ikke være systematisk nok med tanke på det Barefoot Computing (u.å.g) definerer. Hvis man sammenligner dette med Utdanningsdirektoratets meninger når det kommer til evaluering er elevene helt innenfor kravene, da deres mål med evaluering er bare at elevene skal gjøre vurderinger, noe gruppe 5 viste gode preg av.

Det er viktig at elevene lærer å gjøre effektive og gode evalueringer, da dette er en ferdighet de mest sannsynlig kommer til å ha god bruk for i et framtidig arbeidsliv, som er i stor teknologisk utvikling (Meld. St. 28, 2015-2016). Derfor, ved å gi elevene oppgaver hvor de blir nødt til å benytte seg av evaluering, og utvikle ferdigheten for å bli mer effektiv og se løsninger til problemer raskere, er det nyttig å utfordre elevene til å prøve seg på nye og mer utfordrende programmeringsoppgaver, ettersom de utvikler seg.

5.2 Kjennetegn på arbeidsmåter i den algoritmiske tenkeren

I analysen ble arbeidsmåtene i den algoritmiske tenkeren sett på hver for seg som et tema. Under diskusjonen av resultatene vil arbeidsmåtene diskuteres hver for seg opp mot relevant teori, og noen nøkkelbegreper og arbeidsmåter kan bli nevnt i drøftingen hvis det er hensiktsmessig.

5.2.1 Fikling – Bli kjent med objekt og kode

Fikling går ut på å utforske og eksperimentere nye ting for å oppdage hva det gjør eller hvordan det virker (Barefoot Computing, u.å.k; Utdanningsdirektoratet, 2019). Fikling kan virke som et begrep som kun kan brukes om objekter som kan være i hendene og plukkes fra hverandre for å se hva som er inni. Det trenger ikke alltid å være tilfellet, ja det kan virke fristende å bare se på det som objekt, men innenfor programmering kan det være både et objekt, men i hovedsak vil fiklingen foregå i et program, som inneholder koder. I resultatene er det eksempler på fikling både med objekt (Bit:Bot), og i program.

Det første eksemplet med gruppe 4, som hadde en bil som kjørte skeivt, valgte elevene å eksperimentere og utforske om det var selve bilen som var grunnen til problemet. Med tanke på at elevene ikke hadde brukt Bit:Boter før, kan det være en naturlig problemstilling for dem å utforske, om det er dekket eller motoren som er årsaken til skeivkjøringen på Bit:Boten. Etter de hadde funnet ut at ingen av disse var grunnen, og hadde diskutert problemet med en lærer, fikk de bekreftet at det kan være et problem med motoren som gjør det, men at man kan tilrettelegge for slike feil med forskjellige koder i programmet.

Oppgavene elevene fikk fra læreren i 3. og 4. time var åpne oppgaver der elevene fikk et mål som programmet skulle inneholde, og et program som var delvis programmert. Når det da kommer til å fikle i koder og programmer var det mange grupper som utforsket og eksperimenterte med hvilke koder som måtte brukes for å få programmet til å nå sluttmålet. Gruppe 6, som vist i resultatdelen, eksperimenterte med diverse fartsprosenter for å finne ut hvordan de kunne få Bit:Boten til å snu 180 grader. Etter mer utforsking og eksperimentering kom gruppa fram til at de hadde lagt inn feil kode, som gjorde at Bit:Boten konstant snurret etter at den hadde kjørt en meter.

Fikling kan derfor sees på som å ha god påvirkning på hvordan elevene løser oppgavene, noe som blir bygget under av forskningen gjort av Yihuan Dong m.fl. (2019). Elevene er positive og viser at de har lyst å få programmet til å virke, og kommer seg framover ved hjelp av fikling. Det ble også observert i noen av gruppene, at det å utforske og prøve nye forslag av programmene ble ufarlig, og å teste det ikke var noe problem. På denne måten blir oppgaveløsing i programmering mer ufarliggjort for elevene, og gjør dem tryggere i sin problemløsningsprosess.

5.2.2 Skape – Finjustering og kreativitet

Å skape i programmeringssammenheng, vil være å skape et program som gjør noe i den digitale eller fysiske verden. I begge oppgavene elevene fikk i 3. og 4. time var det gitt et halvferdig program, som viste elevene grunnkodene som måtte være med for å kunne lage programmet ferdig suksessfullt. Allerede her startet skaperprosessen for elevene, med tanke på at å skape var ikke bare å lage et nytt program, men også å forbedre et allerede eksisterende program.

I starten av denne prosessen var det mange av gruppene som klarte å forutse hvordan rekkefølgen på kodene for programmet skulle være, for å få Bit:Boten til å kjøre en meter, snu og kjøre tilbake. Etter hvert ble oppgaven mer krevende, da de måtte finpusse på programmet for å få bilen kunne kjøre 100 prosent rett til en metersmerket, og snu perfekt 180 grader for så å kjøre 100 prosent rett tilbake. Dette krevde at elevene var åpne til å bruke kreativitet for å designe en kode som gjorde programmet mer finjusterende for få løst oppgaven (Barefoot Computing, u.å.d). Det var ingen av gruppene som rakk å bli ferdig med denne finjusteringen, fordi de skulle starte med robotstøvsuger/-gressklipper oppgaven, men de gruppene som hadde begynt å finjustere, viste smått frustrasjon til finjusteringen på programmet, da det tok mye lengre tid å få en kommando riktig, enn det hadde gjort med grunnprogrammet.

Finjustering er et begrep som kan tolkes som at elevene må designe og lage et program som er mer effektivt, som kan gjennom Utdanningsforbundets (2019) beskrivelse av begrepet skape, stemme med tolkningen. Skaperprosessen hos elevene ble ofte fremprovosert for det meste av læreren, som i oppgaven med robotstøvsuger/-gressklipperen var det mye mer utfordrende å gjenskape det halvferdige programmet som de ble gitt ved oppgavestart. Som vist i resultatene var gruppe 2 en av de gruppene som først ble ferdig med grunnprogrammet og videre ble utfordret av lærer til å finjustere. På en generell basis virker det som at en slik ordlegging fra læreren får elevene til å virke mer negativ til programmeringen videre. En mulig grunn til det kan være at elevene viser stolt frem at de har løst oppgaven, og blir belønnet med mer arbeid for strevet. Det var ikke tilfellet for gruppe 2, da de tok denne utfordringen på strak arm, og viste eierskap til sitt arbeid.

5.2.3 Feilsøke – Gjenkjenne feil og korrigere dem

Ferdigheten med å se på en kode i et program og finne hvor feilen ligger, er noe som er viktig under oppgaveløsning i programmering. Som tidligere nevnt er det sjelden at et komplekst program virker som det var ment ved første forsøk. Dette bekreftes ved å se på de fleste eksemplene dratt frem i resultatene fra analysen.

Hvis man skal se resultatene gjennom Utdanningsdirektoratets (2019) beskrivelse av begrepet feilsøking, er det beskrevet som å oppdage og rette feil. Alle gruppene jobbet godt med å prøve å finne og gjenkjenne hvor og hva eventuelt feilen i deres program var, gjennom hele programmeringsprosessen. I elevenes programmer var det ofte feil i syntaksen som var problemet, ikke feil i logikken. Det vil si at historien for programmet er riktig formulert, men den er stavet feil (Barefoot Computing, u.å.e). Barefoot Computing (u.å.e) nevnte også at å lete etter feil i programmer er mye mer tidkrevende enn selve programmeringsprosessen. Det ble tydelig observert da elevene mente de fort ble ferdig med selve programmeringen, men når det kom til å finne frem til akkurat hvilke tall som skulle inn for hastighet og for tid, brukte elevene mye lengre tid på dette.

Eksempelvis som gruppe 6 som prøvde seg frem med både forskjellige koder og forskjellige tider for å få Bit:Boten til å snu 180 grader. Ut ifra datamaterialet kan man se at disse elevene gjennomgikk alle punktene til Barefoot Computings fire trinn for feilsøking (Barefoot Computing, u.å.e). Elevene forutså hva som skulle skje gjennom å bruke logisk tenking, de fant ut hva som skjedde, ved bruk av testing og utforskning, de fant ut hvor noe var galt ved å evaluere resultatet av utforskingen, og de fikset det etter flere gjennomganger av disse stegene. Dette var en loop som elevene havnet i, noen av seg selv, mens andre fikk små påminnelser fra læreren under oppgaveløsingen.

Feilsøking ble noe elevene etter hvert brukte bevisst, da de visste at det var det som måtte til for å komme seg videre i oppgaven. Ut ifra resultatet vises det at feilsøking faller elevene naturlig i programmeringsoppgaver, og det er et godt tegn at de viser at de kan få det til uten hjelp av lærere. Grunnen til det, er at hvis elever trenes godt i hvordan man skal tenke i den algoritmiske tenkeren, kan man etter hvert implementere samme tankesett i andre oppgaver enn programmering (Wing, 2006). Evnen til å finne feil i uansett oppsett kan være en veldig god ferdighet for elevene å tilegne seg, slik at å løse feil i oppgaveløsing i framtiden blir enklere.

5.2.4 Holde ut – Utfordrende? Fortsett å prøv

Gruppe 1 møtte på en utfordring ved at Bit:Boten bare snurret, når den egentlig skulle kjøre framover, og når den møtte en hindring skulle den snu seg unna. Det viste seg at de hadde plassert kodene feil i programmet, men elevene var ikke klar over det. De endret både koder og enheter i programmet for å prøve å få det til å virke, men de kom ingen vei.

Elevene i gruppe 1 viste at de hadde evnen til å holde ut i en utfordring, og de fokuserte på å feilsøke og prøve å løse problemet. Ifølge Utdanningsdirektoratets (2019) beskrivelse av å holde ut, stemmer det med hva elevene gjør, da deres definisjon er at elevene fortsetter å holde ut i oppgaveløsing. Som nevnt fra resultatkapittelet, endte gruppa opp i frustrasjon og viste tegn til å ville gi opp. Da fikk de veiledning og hjelp til å lese programmet av læreren. I følge Barefoot Computing (u.å.e) er det viktig at elevene får hjelp til å utvikle den kognitive kondisen, som for denne gruppa var hjelpen så enkel som å lese programmet for elevene, slik at de kunne høre hvor feilen var. Med tanke på at elevene ikke hadde jobbet spesielt mye med programmering, kan det å lese hva programmet sier være en stor utfordring og fallgruve for dem, da de ikke har lært hvordan det gjøres enda.

Å holde ut er en arbeidsmåte som blir enklere hvis elevene har gode metoder for bruk av feilsøking og dekomposisjon (Barefoot Computing, u.å.j). Hvis du er god på dekomposisjon, gjør det feilsøkingen enklere, og det ender opp med at oppgavene blir løst lettere, som deretter fører til at elevene holder ut lenger i problemløsingen.

5.2.5 Samarbeid – Bruke hverandre som en ressurs

Hvis man skal se på gruppesammensetningene, var det ikke lagt noe tanke bak hvilke elever som var på gruppe med hverandre. Gruppene ble bestemt ut ifra hvordan elevene satt i klasserommet, og plassene ble tilfeldig trukket hver uke. Det kan ha stor påvirkning til hvordan resultatet for gruppesamarbeid utspilte seg ved oppgaveløsingen.

Egenskapen om å kunne samarbeide med andre blir sett på som en forventet ferdighet i arbeidslivet (NOU 2015:8, 2015). Samarbeid er som oftest en god bruk av ressurser, hvis partene som skal samarbeide klarer å se hverandres ferdigheter, og bruke dem hensiktsmessig, men det er ikke alltid gruppesammensetningen er like heldig, med tanke på å få elever til å samarbeide godt sammen. I datamaterialet ble det observert for det meste godt samarbeid mellom elevene i gruppene, men det var naturligvis også noen grupper som ikke var like gode til å samarbeide. Dette er en viktig ferdighet for barn å øve seg på, da det er noe de mest sannsynlig kommer til å møte i en eller annen form gjennom livet.

Det er forskjellige måter å samarbeide på, og det ble også observert. I gruppe 3 jobbet elevene sammen i hele prosessen, de programmerte sammen, testet sammen og evaluerte sammen. I gruppe 5 var oppgavene delt mellom elevene, altså en elev hadde ansvar for programmeringen, og den andre eleven hadde ansvar for å sette på plass Bit:Boten, gjøre alt klart til neste test og ordne slik at programmet ble overført til micro:biten.

I gruppe 7 var det en del utfordringer knyttet til samarbeidsmetodene. Kommunikasjonen og oppgavedelingen mellom elevene var ikke optimale. Det viste de fleste av diskusjonene som gruppa hadde, og gav uttrykk av at de ikke hadde den samme gleden ut av programmeringen som de fleste andre gruppene hadde. Selv om samarbeidet ikke fungerte optimalt i gruppe 7, kunne det gitt dem en mulighet til å lære fra situasjonen. Det er ikke alltid slik at man skal samarbeide med noen man går over ens med, og derfor må man bli bedre på å kommunisere profesjonelt og snakke til hverandre med respekt.

Selv om gruppe 5 hadde fordelt oppgavene, var det i hovedsak en elev som programmerte og en elev som gjorde det praktiske. Hvis man skal se på læringen i dette blir det bare en elev som lærer selve programmeringen for oppgaven. Grunnen til at oppgavene kanskje ble delt slik mellom elevene kan være fordi at elevene ser på iPadene til hverandre som veldig personlige, som kan gjøre det litt vanskeligere å dele på den. Derfor kan det videre føre til at det er en som ender opp med å programmere, mens den andre tar seg av Bit:Boten, som kan sees på som ufarlig da det er skolen sitt utstyr.

Hvis man skal velge ut samarbeidsmetoden som kunne vært mest gunstig ut ifra situasjonene, kan man konkludere med at måten gruppe 3 jobbet, var den beste for at begge skulle lære fra øvelsen. Grunnen til at dette kanskje er den mest effektive samarbeidsmetoden for begge elevene er at de jobber sammen på hvert steg gjennom programmeringsprosessen. Utdanningsforbundet (2019) kommer inn på at å samarbeide handler om å dele og jobbe sammen, som er noe gruppe 3 viste gode tendenser til å gjøre. De samarbeidet godt for å oppnå best mulig resultat, og hadde gode diskusjoner gjennom hele prosessen. Gruppa viste at de jobbet mot et felles mål, og prøvde å jobbe sammen for å få til dette. Derfor blir dette, følge Barefoot Computing (u.å.l), den beste samarbeidsmåten som er observert i resultatene.

6. Konklusjon

Hensikten med denne studien var å undersøke på hvilken måte elevers aktivitet preges av algoritmisk tenking ved løsning av programmeringsoppgaver. Forskningsspørsmålene for oppgaven var:

1. Hvilke av de seks nøkkelbegrepene bruker elever under oppgaveløsning i programmering?
2. Hvilke av de fem arbeidsmetodene bruker elever under oppgaveløsning i programmering?

Ut ifra funnene i denne studien kan man konkludere med at elever blir preget av den algoritmiske tenkeren ved løsning av programmeringsoppgaver. Elevene bruker de fleste av nøkkelbegrepene og arbeidsmåtene til å reflektere og gjøre oppgavene enklere å løse sammen i grupper. Selv om resultatene viste at elevene brukte de fleste av strategiene i den algoritmiske tenkeren, var det to nøkkelbegreper som ikke ble brukt av elevene, nemlig mønstre og abstraksjon. Det kan være flere grunner til at disse ikke ble brukt, men hovedårsaken kan være at programmering var nytt for elevene, og at disse nøkkelbegrepene kanskje er strategier som må innarbeides. Med tanke på at dette var elevenes første fire timer med programmering, kunne det vært interessant å sammenligne resultatene fra denne studien, hvor elevene var relativt ferske ved bruk av programmering, med elever som har mer erfaring med programmeringsoppgaver og løsninger, og se om det er forskjeller i resultatene.

Basert på funn i forskningen, kan man se for seg at modellen for den algoritmiske tenkeren kan være formålstjenlig i andre oppgaver enn kun programmeringsoppgaver. Muligheten til å bruke modellen for å løse generelle problemløsningsoppgaver, og å se om elever bruker strategier fra den algoritmiske tenkeren til å løse andre oppgaver, hadde vært interessant å sett resultater fra i en annen forskningsoppgave.

7. Litteraturliste

Barefoot Computing. (u.å.a). *Abstraction*. Barefoot - Computing at School. https://www.barefootcomputing.org/concepts-and-approaches/abstraction

Barefoot Computing. (u.å.b). *Algorithms*. Barefoot - Computing at School. https://www.barefootcomputing.org/concepts-and-approaches/algorithms

Barefoot Computing. (u.å.c). *Computational Thinking Concepts and Approaches*. Barefoot - Computing at School. https://www.barefootcomputing.org/concept-approaches/computational-thinking-concepts-and-approaches

Barefoot Computing. (u.å.d). *Creating*. Barefoot - Computing at School. https://www.barefootcomputing.org/concepts-and-approaches/creating

Barefoot Computing. (u.å.e). *Debugging*. Barefoot - Computing at School. https://www.barefootcomputing.org/concepts-and-approaches/debugging

Barefoot Computing. (u.å.f). *Decomposition*. Barefoot - Computing at School. https://www.barefootcomputing.org/concepts-and-approaches/decomposition

Barefoot Computing. (u.å.g). *Evaluation*. Barefoot - Computing at School. https://www.barefootcomputing.org/concepts-and-approaches/evaluation

Barefoot Computing. (u.å.h). *Logic*. Barefoot - Computing at School. https://www.barefootcomputing.org/concepts-and-approaches/logic

Barefoot Computing. (u.å.i). *Patterns*. Barefoot - Computing at School. https://www.barefootcomputing.org/concepts-and-approaches/patterns

Barefoot Computing. (u.å.j). *Preserving*. Barefoot - Computing at School. https://www.barefootcomputing.org/concepts-and-approaches/persevering

Barefoot Computing. (u.å.k). *Tinkering*. Barefoot - Computing at School. https://www.barefootcomputing.org/concepts-and-approaches/tinkering

Barefoot Comuting. (u.å.l). *Collaborating*. Barefoot - Computing at School. https://www.barefootcomputing.org/concepts-and-approaches/collaborating

Barr, V., & Stephenson, C. (2011). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community? *ACM Inroads*, 2 (1), 48-54. https://doi.org/10.1145/1929887.1929905

Binau, F, C., & Trolle, O. (2021). Delevaluering af DR ultra:bit. *Naturfagenes evaluerings- og udviklingscenter*, 1 - 46.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3 (2), 77-101. https://doi.org/10.1191/1478088706qp063oa

Braun, V., & Clarke, V. (2022). *Thematic Analysis, A Practical Guide* . Sage.

Dong, Y., Marwan, S., Cateté, 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*, 19 (1), 1204-1210. https://doi-org.ezproxy.nord.no/10.1145/3287324.3287437

Dragland, H., & Ebert, C. M. (2022). *Programmering i naturfag: den algoritmiske tenkeren og utfordringer på veien* [Masteroppgave, OsloMet]. ODA Open Digital Archive. https://oda.oslomet.no/oda-xmlui/handle/11250/3027460

Gilje, N., & Grimen, H. (1993). *Samfunnsvitenskapens forutsetninger: innføring i samfunnsvitenskapeks vitenskapsfilosofi* (3. utgave). Universitetsforlaget AS.

Giske, T., Aasen, M., Henriksen, M., Sønsterud, T. M., & Bjørnø, C. T. (2016). *Representantforslag om en nasjonal strategi for digitalisering og oversikt over den digitale tilstanden i norsk skole.* Stortinget. https://www.stortinget.no/nn/Saker-og-publikasjonar/publikasjonar/Representantframlegg/2016-2017/dok8-201617-004s/

Høgheim, S. (2020). *Masteroppgaven i GLU.* Fagbokforlaget.

Kunnskapsdepartementet. (2017-2020). *Framtid, fornyelse og digitalisering.* https://www.regjeringen.no/contentassets/dc02a65c18a7464db394766247e5f5fc/kd_framtid_fornyelse_digitalisering_nett.pdf

Larsen, A. K. (2007). *En enklere metode.* Fagbokforlaget.

Lin, Y., & Weintrop, D. (2021). The landscape of Block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming. . *Journal of Computer Languages*, 67 (1), 1-18. https://doi.org/10.1016/j.cola.2021.101075

Meld. St. 28. (2015-2016). *Fag - Fordypning - Forståelse.* Kunnskapsdepartementet. https://www.regjeringen.no/contentassets/e8e1f41732ca4a64b003fca213ae663b/no/pdfs/stm201520160028000dddpdfs.pdf

n00b.no. (u.å.). *BBC micro:bit i Norge.* n00b. https://n00b.no/collections/bbc-micro-bit?gclid=Cj0KCQiAr8eqBhD3ARIsAIe-buPeMGk2X0Ebb7UZRAVzWWM5br2gicioMBRWRmRNEP_MpdC4QJx0RGMaAt-aEALw_wcB

NOU 2014:7. (2014). *Elevens læring i framtidens skole.* Kunnskapsdepartementet. https://www.regjeringen.no/contentassets/e22a715fa374474581a8c58288edc161/no/pdfs/nou201420140007000dddpdfs.pdf

NOU 2015:8. (2015). *Framtidens skole.* Kunnskapsdepartementet. https://www.regjeringen.no/contentassets/da148fec8c4a4ab88daa8b677a700292/no/pdfs/nou201520150008000dddpdfs.pdf

Nyeng, F. (2012). *Nøkkelbegreper i forskningsmetode og vitenskapsteori.* Fagbokforlaget.

Papert, S. (1980). *Midstorms: children, computers, and powerfull ideas.* Basic Books.

Papert, S. (1983). *Dialog med datamaskinen.* Cappelens forlag AS.

Sanne, A., Berge, O., Bungum, B., Jørgensen, E. C., Kluge, A., Kristensen, T. E., . . . Voll, L. O. (2016). *Teknologi og programmering for alle.* Utdanningsdirektoratet.

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

Super:bit. (u.å.a). *Kom i gang med micro:bit.* superbit.no: https://www.superbit.no/media/1730/kom-i-gang-med-microbit-bm.pdf

Super:bit. (u.å.b). *Oppvarming - Programmering uten datamaskin.* superbit.no: https://www.superbit.no/media/1283/superbit-oppdraget_intro-presentasjon.pdf

Utdanningsdirektoratet. (2013). *Læreplan i naturfag (NAT1-03)*. Ikke lenger som forskrift. Læreplanverket for Kunnskapsløftet 2006. https://www.udir.no/kl06/nat1-03/#

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

Vitensenterene. (u.å.a). super:bit skole. Hentet 06.01.2023. https://www.vitensenter.no/superbit/skole/

Vitensenterene. (u.å.b). Utstyr til super:bit. Hentet 06.01.2023. https://www.vitensenter.no/superbit/utstyr-til-superbit/

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49 (3), 33-35. https://doi-org.ezproxy.nord.no/10.1145/1118178.1118215

Vedlegg 1: Godkjenning fra Sikt

Sikt

Meldeskjema / Algoritmisk tenkning gjennom programmeringsundervisning i naturfag... / Vurdering

Vurdering av behandling av personopplysninger

Referansenummer	Vurderingstype	Dato
380615	Standard	10.02.2023

Tittel
Algoritmisk tenkning gjennom programmeringsundervisning i naturfag – En praktisk studie med programmering.

Behandlingsansvarlig institusjon
Nord Universitet / Fakultet for lærerutdanning og kunst- og kulturfag / Grunnskole

Prosjektansvarlig
Knut Moksnes

Student
Trine Gundersen

Prosjektperiode
01.02.2023 - 31.12.2023

Kategorier personopplysninger
Alminnelige

Lovlig grunnlag
Samtykke (Personvernforordningen art. 6 nr. 1 bokstav a)

Behandlingen av personopplysningene er lovlig så fremt den gjennomføres som oppgitt i meldeskjemaet. Det lovlige grunnlaget gjelder til 31.12.2023.

Meldeskjema

Kommentar
OM VURDERINGEN
Sikt har en avtale med institusjonen du forsker eller studerer ved. Denne avtalen innebærer at vi skal gi deg råd slik at behandlingen av personopplysninger i prosjektet ditt er lovlig etter personvernregelverket.

UTDYPENDE OM LOVLIG GRUNNLAG
Prosjektet vil innhente samtykke fra foresatte til behandlingen av personopplysninger om barna. 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/foresatte kan trekke tilbake.

FØLG DIN INSTITUSJONS RETNINGSLINJER
Vi har vurdert at du har lovlig grunnlag til å behandle personopplysningene, men husk at det er institusjonen du er ansatt/student ved som avgjør hvilke databehandlere du kan bruke og hvordan du må lagre og sikre data i ditt prosjekt. Husk å bruke leverandører som din institusjon har avtale med (f.eks. ved skylagring, nettspørreskjema, videosamtale el.

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).

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. Se våre nettsider om hvilke endringer du må melde: https://sikt.no/melde-endringar-i-meldeskjema

OPPFØLGING AV PROSJEKTET
Vi vil følge opp ved planlagt avslutning for å avklare om behandlingen av personopplysningene er avsluttet.

Lykke til med prosjektet!

1/1

Vedlegg 2: Informasjonsskriv

Vil du delta i forskningsprosjektet: *Algoritmisk tenkning gjennom programmeringsundervisning i naturfag – En praktisk studie med programmering*?

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å forske på om nøkkelideene innenfor algoritmisk tenkning kan oppnås gjennom programmeringsoppgaver i naturfag. I dette skrivet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål
Formålet med denne masterstudien er å se om programmeringsoppgaver i norsk skole stimulerer til at nøkkelideene innenfor algoritmisk tenkning kan oppnås. Algoritmisk tenkning og programmering er begreper som har fått veldig stort fokus etter fagfornyelsen i 2020. Dette er begreper det derfor bør forskes på i norsk skole, for å se hvor utfordringene i oppgaveoppsettene kan ligge.

Problemstillingen som skal analyseres er:
Hvordan kan forskjellige nøkkelideer innenfor algoritmisk tenkning oppnås gjennom programmeringsoppgaver?

Hvem er ansvarlig for forskningsprosjektet?
Nord Universitet er ansvarlig for forskningsprosjektet.

Hvorfor får du spørsmål om å delta?
Utvalget for denne studien er gjort strategisk, det vil si at valg av klassetrinn og alder påvirker relevansen for resultatet i undersøkelsen. Derfor er hovedfokuset på mellomtrinn, altså 5. - 7. trinn. Utvalget for denne studien er også gjort av bekvemmelighetsgrunner, som betyr at valget av forskningsskole skjer gjennom bekjente.

Hva innebærer det for deg å delta?
Hvis du velger å delta i prosjektet, innebærer det at du deltar i undervisningstimer med oppgaveløsning, hvor det vil bli gjort videopptak for å samle inn datamateriale. Videoopptakene blir lagret på en fysisk isolert enhet med minnekort under innspilling, og deretter overført til digital lagring.

Hvis datamateriale etter observasjon ikke inneholder nok informasjon, kan et gruppeintervju bli nødvendig for å få nok forskningsmateriale. Gruppeintervjuet vil ta ca. 45-60 minutter. Intervjuet vil inneholde spørsmål om hvordan tankene rundt en oppgaveløsning er. Intervjuet vil bli tatt opp med bruk av Nettskjema Diktafon-appen, hvor opptakene blir lagret elektronisk.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykket tilbake uten å oppgi noen grunn. Alle dine personopplysninger vil da bli slettet. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg. Hvis du velger å ikke delta, vil undervisningen bli tilrettelagt i sammenheng med når forskningen skjer på skolen.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger
Vi vil bare bruke opplysningene om deg til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket. De som har tilgang til datamateriale som innsamles er student, Trine Gundersen, og veileder, Knut Moksnes.

Personopplysningene dine vil bli erstattet med en kode som lagres på egen navneliste adskilt fra øvrige data. Datamaterialet vil bli lagret elektronisk på en forskningsserver. Deltakerne vil ikke kunne gjenkjennes i publikasjon, da informasjonen som skal brukes fra prosjektet kun er videoopptak og lydopptak, som vil bli kryptert slik at gjenkjenning ikke vil skje.

Hva skjer med personopplysningene dine når forskningsprosjektet avsluttes?
Prosjektet vil etter planen avsluttes når oppgaven er godkjent, som vil si ca. 31.12.2023. Etter prosjektslutt vil datamaterialet med dine personopplysninger anonymiseres, med å anonymisere navn, og eventuelt andre opplysninger som kan gjenkjennes. Anonymiserte opplysninger vil ikke slettes, men kunne gjenbrukes til for eksempel forskning.

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

På oppdrag fra Nord Universitet har Sikt – Kunnskapssektorens tjenesteleverandør vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

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

- innsyn i hvilke opplysninger vi behandler om deg, og å få utlevert en kopi av opplysningene
- å få rettet opplysninger om deg som er feil eller misvisende
- å få slettet personopplysninger om deg
- å sende klage til Datatilsynet om behandlingen av dine personopplysninger

Hvis du har spørsmål til studien, eller ønsker å vite mer om eller benytte deg av dine rettigheter, ta kontakt med:

- Nord Universitet ved:
  Student Trine Gundersen, tlf.: 959 31 701, e-post: trine.gundersen@hotmail.com
  eller
  Veileder Knut Moksnes, tlf.: 951 29 443, e-post: knut.moksnes@nord.no
- Vårt personvernombud:
  Toril Irene Kringen, tlf.: 74 02 27 50, e-post: personvernombud@nord.no

Hvis du har spørsmål knyttet til vurderingen som er gjort av personverntjenestene fra Sikt, kan du ta kontakt via:

- Epost: personverntjenester@sikt.no eller telefon: 73 98 40 40.

Med vennlig hilsen

Knut Moksnes Trine Gundersen

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet *Algoritmisk tenkning gjennom programmeringsundervisning i naturfag – En praktisk studie med programmering*, og har fått anledning til å stille spørsmål. Jeg samtykker til at mitt barn:

- ☐ ønsker å delta i videoopptak
- ☐ ønsker ikke å delta i prosjektet

Jeg samtykker til at mitt barns opplysninger behandles frem til prosjektet er avsluttet

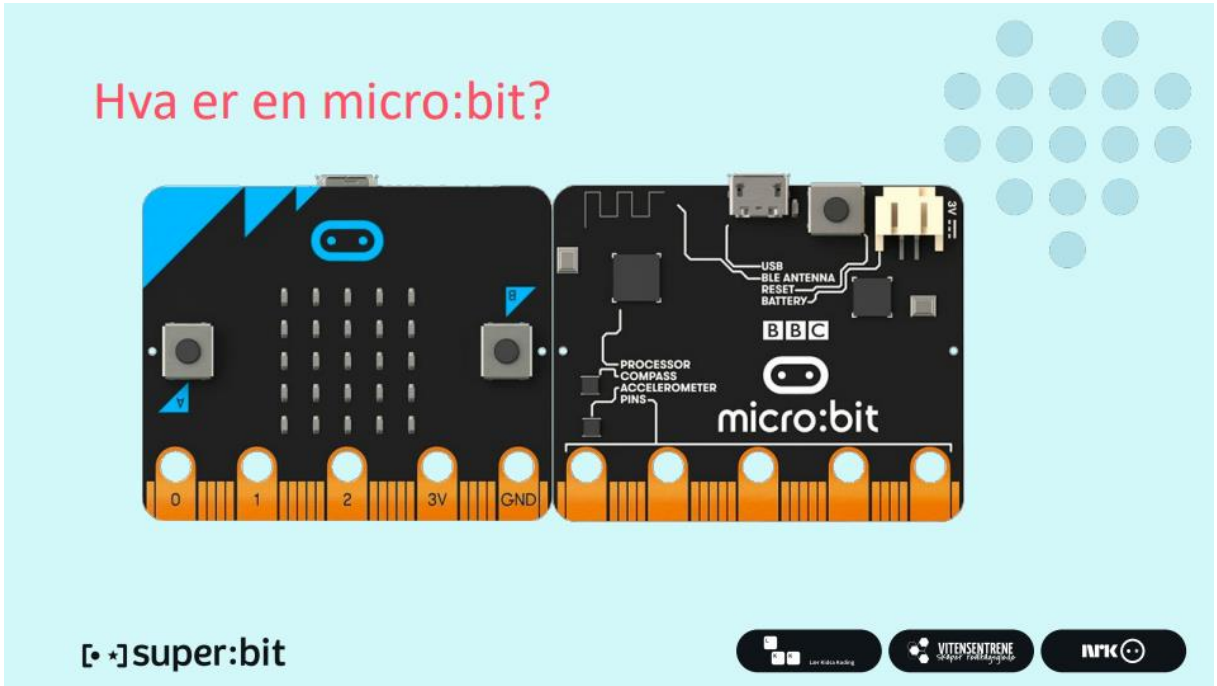
_____

(Barns navn, prosjektdeltaker)

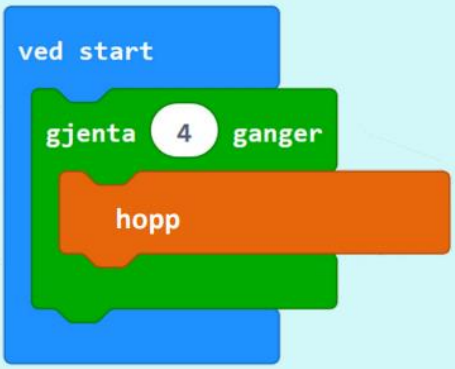
_____

(Signert av foresatt til prosjektdeltaker, dato)

Vedlegg 3: Oppvarming – Programmering uten datamaskin
(Super:bit, u.å.b)



Løkker

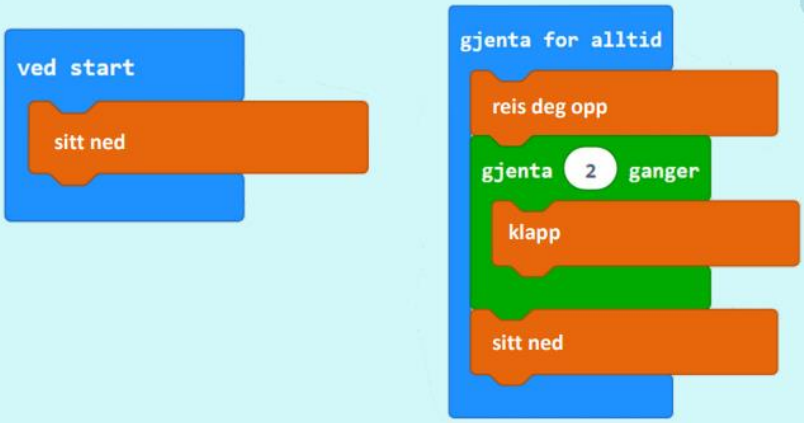


super:bit

Pause

super:bit

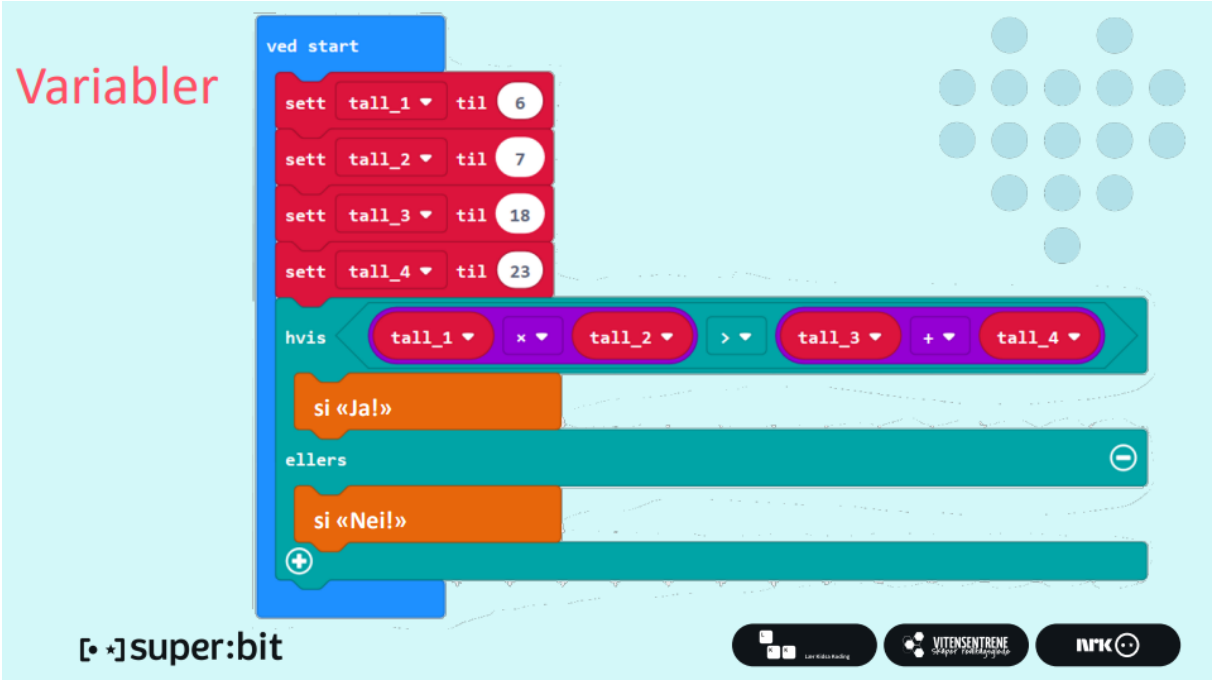
Gjenta for alltid



super:bit

Løkke i løkke

super:bit


Variabler
ved start
sett tall_1 til 6
sett tall_2 til 7
sett tall_3 til 18
sett tall_4 til 23
hvis tall_1 × tall_2 > tall_3 + tall_4
si «Ja!»
ellers
si «Nei!»
super:bit

Vedlegg 4: Komme i gang med micro:bit
(Super:bit, u.å.a)

super:bit

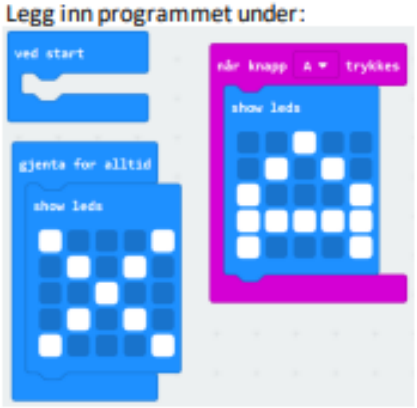
Kom i gang med

micro:bit

Klar for å overføre ditt aller første program til micro:bit? Her er en liten oppgave for å komme i gang. Har du tid, er det ei ekstra utfordring på baksiden av dette arket.

Vis dine egne figurer på skjermen

Legg inn programmet under:



Det kan hende du må lete litt i menyene for å finne alle blokkene. Tips: det er lurt å se på fargene.

Læreren vil vise hvordan du går fram for å laste programmet inn på micro:biten.

Legg gjerne inn egne figurer, og kanskje noe skal skje når du trykker på B?
Du kan også leke deg med de andre blokkene i Basis
Får du den til å skrive navnet ditt?

Side 1 av 2

Blinkende hjerte

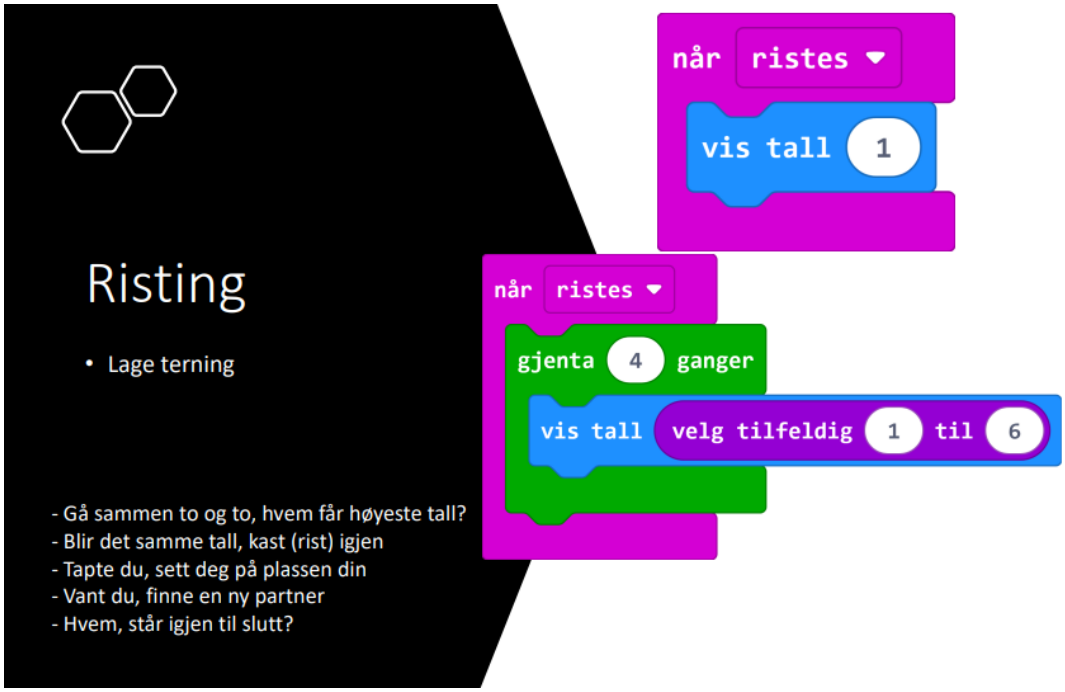
Lag et blinkende hjerte med micro:bit!

Ikke rist meg!

Programmer micro:biten til å protestere dersom den blir ristet:

Legg inn programmet, last ned, rist micro:biten og se hva som skjer.

Vedlegg 5: Presentasjon time 2



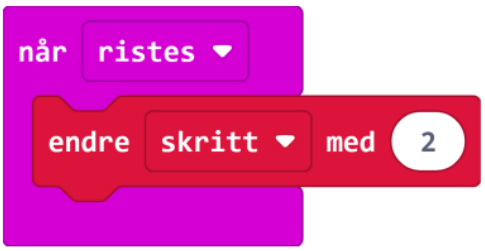
Variabel, skritteller

- Vi skal lage en skritteller
- Vi gjør det sammen

Hvor mang skritt går du på 2 minutt?

Hvor langt går du på et skritt?

Hvor langt har du gått til sammen?



Kan du lage noe der du bruker flere funskjoner?

- Knapp A
- Knapp B
- Ristes
- Løkker (gjenta et bestemt antall ganger)
- Gjenta for alltid

Vedlegg 6: Super:bit – kom i gang med Bit:Bot

URL: https://www.superbit.no/undervisningsopplegg/kjoer-en-meter-med-bitbot/

Introduksjonsvideo til hvordan man setter sammen selve Bit:Boten, hva de forskjellige delene gjør, og introduksjon hvordan man kan programmere den til å gjøre forskjellige ting.

Oppgave som forteller elevene hva som skal til for at oppgaven skal løses. Oppgaven inneholder også en veiledning i hvordan roboten fungerer, og hvilke faktorer elevene bør tenke på når de løser oppgaven.

Vedlegg 7: Lag en robotgressklipper som unngår hindringer

URL: https://www.superbit.no/undervisningsopplegg/lag-en-robotgressklipper-som-unngaar-hindringer/

Oppgavebeskrivelse om hvordan man kan lage en robotgressklipper ved hjelp av en Bit:Bot og en ultralydsensor. Den viser også hint om hvordan programmet kan løses.

Vedlegg 8: Deltakerskjema

Nr.	Deltakere	Deltar/Deltar ikke
1	Ola Norman	Deltar
2	Kari Norman	Deltar ikke
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		