https://doi.org/10.7577/formakademisk.4812

Peter Haakonsen
Universitetslektor
Institutt for estetiske fag
Fakultet for teknologi, kunst og design
OsloMet – storbyuniversitetet
peterh@oslomet.no

Programmering i Kunst og håndverk 2018-2022

Utviklingen av pseudokodemodellen

SAMMENDRAG
Helt siden det tidlig i 2018 ble kjent at programmering skulle bli en del av Kunst og håndverk i kommende læreplaner, startet jeg å holde korte kurs i programmering for faglærerstudenter i design, kunst og håndverk. Hensikten var først å gi studentene et innblikk i hva slags muligheter som lå i programmering i Kunst og håndverk, og forhåpentligvis bidra til at enkelte av studentene ville orientere seg videre i temaet på egenhånd. Utviklingen av denne undervisningen har gradvis utviklet seg fra år til år, i en vekselvirkning sammen med andre relevante prosjekter. Gjennom designbasert forskning med mål om å utvikle og forbedre egen praksis, ser denne artikkelen på fem års undervisning i programmering med hovedfokus på Scratch. En erfaring er at det er viktigere at studenter (og elever) får komme fort i gang med å utforske programmeringens muligheter og begrensninger, enn at de skal få veldig mye opplæring i forkant av en økt. Et annet biprodukt eller resultat av disse erfaringene er at strukturen på undervisningen settes opp som en egen kode, som har fått navnet pseudokodemodellen.

Nøkkelord:
Programmering, LK20, Scratch, DBR, Kunst og håndverk.

INNLEDNING

I 2017 var det klart at programmering skulle bli en del av kommende læreplaner i skolen (Kunnskapsdepartementet, 2017a), og i løpet av 2018 røpet ulike høringer i forkant av fagfornyelsen at det kunne bli en del av grunnskolefaget Kunst og håndverk (Utdanningsdirektoratet, 2018). Jeg hadde uavhengig av dette begynt å utforske både blokk- og tekstbasert programmering ut fra faglige interesser, men det var altså i skoleåret 2017-2018 at jeg, sammen med en annen ansvarlig underviser i digitale medier, ble

enige om at det burde være en del av faglærerutdanningen i design, kunst og håndverk ved OsloMet – storbyuniversitetet.

Første undervisningsopplegg i programmering skulle dermed gjennomføres på nyåret 2018. Tanken var først og fremst å gi studentene en liten innføring i hva programmering er og kan brukes til, og at disse framtidige lærerne burde ha vært innom temaet i løpet av sin utdanning før LK20 skulle tre i kraft (dette studentkullet ville være ferdig med sin bachelorutdanning våren 2020). Dette var også for å ruste dem mot en mulig framtid der programmering er sentralt både i undervisning og som del av en forståelse av hva digital kompetanse er, i motsetning til hvordan for eksempel nettbrett med sine sømløse grensesnitt som gjerne skjuler teknologien som ligger bak (Høibo, 2022). Programmering i flere fag i grunnskolen betyr ikke at alle barn skal være potensielle informatikere i framtida, men det kan bidra til å synliggjøre hvordan digitale verktøy er bygget opp og hvorfor de fungerer som de gjør.

Programmering handler også om algoritmisk tenkning, som både rommer selve algoritmen, altså en stegvis oppskrift eller kode en datamaskin kan følge, og en mer bred forståelse som handler om ulike strategier for å løse et problem. Utdanningsdirektoratet (2019) definerer algoritmisk tenkning som en problemløsningsmetode, der en systematisk tilnærming som å bryte et større problem ned i mindre, mer overkommelige deler, er en av flere strategier. Det handler også om å dele en større oppgave inn i stegvise prosesser, noe som også er overførbart til det å skrive en kode. Begrepet er en norsk oversettelse av det engelske *computational thinking* (Utdanningsdirektoratet, 2019). Utdanningsdirektoratet (2019) definerer inn en rekke arbeidsmåter her, der fikle og eksperimentere, evne til å holde ut og samarbeide, er noen av dem. Flere av de samme formuleringene finnes også i den nye læreplanen for Kunst og håndverk 1.-10., under kjerneelementet kunst- og designprosesser:

> Kjerneelementet kunst- og designprosesser innebærer at elevene skal utvikle nysgjerrighet, kreativitet, mot, skaperglede, utholdenhet og evne til å løse problemer. Kjerneelementet vektlegger både åpne og utforskende prosesser, og stegvise prosesser med utvikling og innovasjon som mål.
> (Kunnskapsdepartementet, 2019)

Et ideal som kommer fram her, er det å jobbe mot et mål som ikke nødvendigvis har en klar fasit. Hvordan skal elevene (og lærerstudentene) bryte et stort og kanskje uoversiktlig problem ned i mer håndterbare deler? Og hvordan kan de jobbe mot et resultat som ikke er forhåndsdefinert, gjennom både stegvise og mer åpne og utforskende prosesser? Programmering kan være en måte å synliggjøre slike (algoritmiske) måter å tenke på. Det kan bidra til å bruke algoritmisk tenkning til å løse problemer i ulike fag. I tillegg kan det bidra til å synliggjøre ens egen tankeprosess gjennom dokumentasjon, deling og samarbeid, og dermed legger til rette for å reflektere over denne (Resnick et. al., 2009, s. 62). Med dette som utgangspunkt søker jeg med denne artikkelen å belyse hvilke perspektiver som kan være relevante å trekke inn når en skal introdusere programmering for faglærerstudenter i Kunst og håndverk.

Fikle / tinker

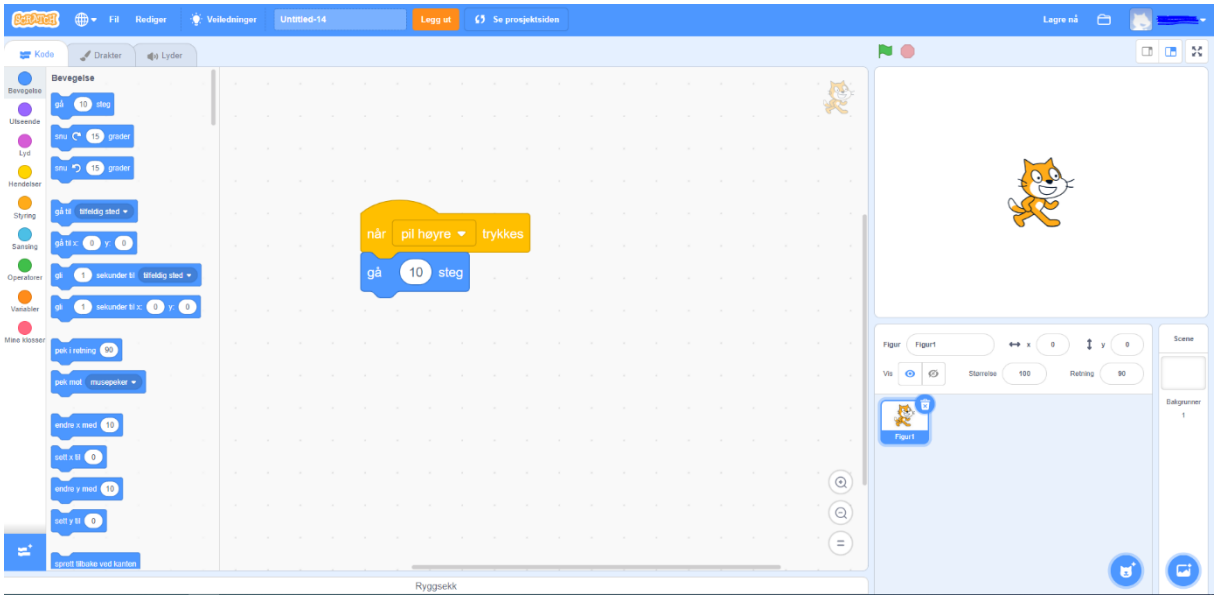
Tinkering er en arbeidsmåte som kan beskrives som en åpen, leken, utforskende og eksperimenterende måte å gå fram på når en skal løse et problem (Martinez & Stager, 2019; Resnick & Rosenbaum, 2013). Utdanningsdirektoratet (2019) bruker *fikle* i sin oversettelse av ordet, der det er innlemmet som en av arbeidsmåtene som hører under algoritmisk tenkning. Martinez og Stager (2019, s. 72, 79) skriver om hvordan overdreven bruk av instruksjoner i forkant av en oppgave kan ødelegge skapergleden og motivasjonen i et klasserom, mens det å la elevene prøve seg fram tidlig i et materiale eller verktøy kan gi bedre grobunn for engasjement og motivasjon. Ved å sette i gang med aktiviteter, i stedet for å måtte høre om dem først, får eleven muligheten til å bli kjent med hva det dreier seg om, og de kan fikle og leke seg fram. Her får de et grunnlag å jobbe videre med, og de vil kanskje være mer mottakelig for supplerende instruksjoner fra læreren senere og underveis. Samtidig lærer de å jobbe selvstendig i

stedet for at det å vente på at læreren har tid til å hjelpe dem er normen i klasserommet (Martinez & Stager, 2019, s. 73). Resnick og Rosenbaum (2013, s. 164) trekker tinkering fram som en motsetning til det å gjenta hva læreren viser eller demonstrerer, og at alle elevene lager forskjellige ting i stedet for å ende opp med å ha laget det samme.

Scratch

Hvordan introdusere programmering for ferske faglærerstudenter, og hva fungerer best når en må anta at de færreste kjenner til programmering fra før? Den første piloten kunne ikke ta mer enn en halv dag med mindre vi skulle ofre andre, mer etablerte deler av undervisningsinnholdet i digitale medier. Rammer for undervisningen var én tretimers undervisningsøkt per studentgruppe. Hvilket programmeringsspråk ville passe best under disse betingelsene? Mulige alternativer kunne være å utvikle nettsider med HTML og CSS, prøve seg med tekstbaserte programmer som Python eller Processing.org, eller programmering kombinert med fysiske mikrokontrollere som Arduino (tekstbasert programmering) eller micro:bit (blokkbasert programmering). Samtidig eksisterer det en rekke muligheter innen blokkbasert programmering på skjerm, der Scratch kanskje er det mest kjente. Blokkbasert programmering krever mindre tid til opplæring, og er dermed et passende utgangspunkt for en workshop av kort varighet. Scratch ble lansert i 2007, og er åpent og fleksibelt med mange muligheter for hva du kan utvikle. For å ramse opp noen eksempler, kan en lage interaktive historier, spill, animasjoner og simuleringer (Resnick et al., 2009). Scratch kan passe for alle som ikke har prøvd programmering før, da det er enkelt å komme i gang. Alle kan mestre på et visst nivå, samtidig som det er mulig å utvikle mer avanserte ting. Selv om grensesnittet er et lekent og fargerikt, vil ikke det si at en ikke kan jobbe med avanserte prosjekter. Bruk av Scratch i undervisning var ikke nytt i 2018, da det blant annet har blitt brukt i valgfaget programmering (Steenbuch, 2016).

Scratch er utviklet på grunnlag av tre idealer: Det skal være *tinkerable*, meningsfullt og sosialt (Resnick et al., 2009). *Tinkerabiliteten*, eller fiklebarheten, er altså programmets muligheter for enkelt å sette sammen digitale brikker for å lage koder, og teste, utforske og endre på disse kodene. Det kan sammenlignes med Lego, men en digital variant (Resnick et al., 2009, s. 63). Brikkene kobles sammen ved hjelp av visuelle hint, og smetter på plass når de passer sammen. Klossenes former hjelper brukeren til å sette dem sammen på måter som øker sjansene for at koden vil fungere. C-formede brikker indikerer for eksempel at andre brikkes skal legges inne i disse (Resnick et al., 2009, s. 63), og på en enkel måte har en laget en løkke, som det heter på fagspråket (Nygård, 2018). En kan med en gang teste hvordan koden fungerer, og det er raskt å veksle mellom det å bygge, teste og justere en kode. Kodeblokkene er sortert etter kategorier med egne fargekoder. Under blå kategori "bevegelse" finner du for eksempel alt som har med bevegelse og retning å gjøre. Ved å hente kodeblokker fra venstremenyen, og sette de sammen midt på skjermen, er det fort gjort å teste ut hvordan to kodeblokker trigger noe til å skje på skjermen. Du kan for eksempel lage en kode som gjør at en figur skal bevege seg til høyre langs x-aksen, og dette kan trigges ved å legge til en betingelse, som at du må trykke på en bestemt tast for at figuren skal bevege seg (se figur 1). I en testrute til høyre på skjermen kan du prøve ut koden din umiddelbart, og en typisk arbeidsprosess i Scratch er en kontinuerlig veksling mellom utvikling, testing og justering. Her inngår også prøving, feiling og endring av kode før den testes på nytt. Testruta til høyre i skjermbildet består av et visst antall piksler i bredde og høyde, henholdsvis x- og y-aksen. Endring av tallene i en kodeblokk hører også med i en slik prosess. Figur 1 viser en enkel kode med en guloransje hendelse: "Når pil høyre trykkes" over en bevegelsesblokk: "gå 10 steg". Tallet 10 indikerer antall steg langs x-aksen, men hva 10 steg egentlig innebærer i praksis bør man heller teste for å erfare i stedet for å høre at en lærer forklarer, slik Martinez og Stager (2019, s. 54) trekker fram hvordan en bør unngå TMI, altså unngå at læreren bruker tid på *Too much instructions / interruptions / interventions* i en undervisningsøkt.

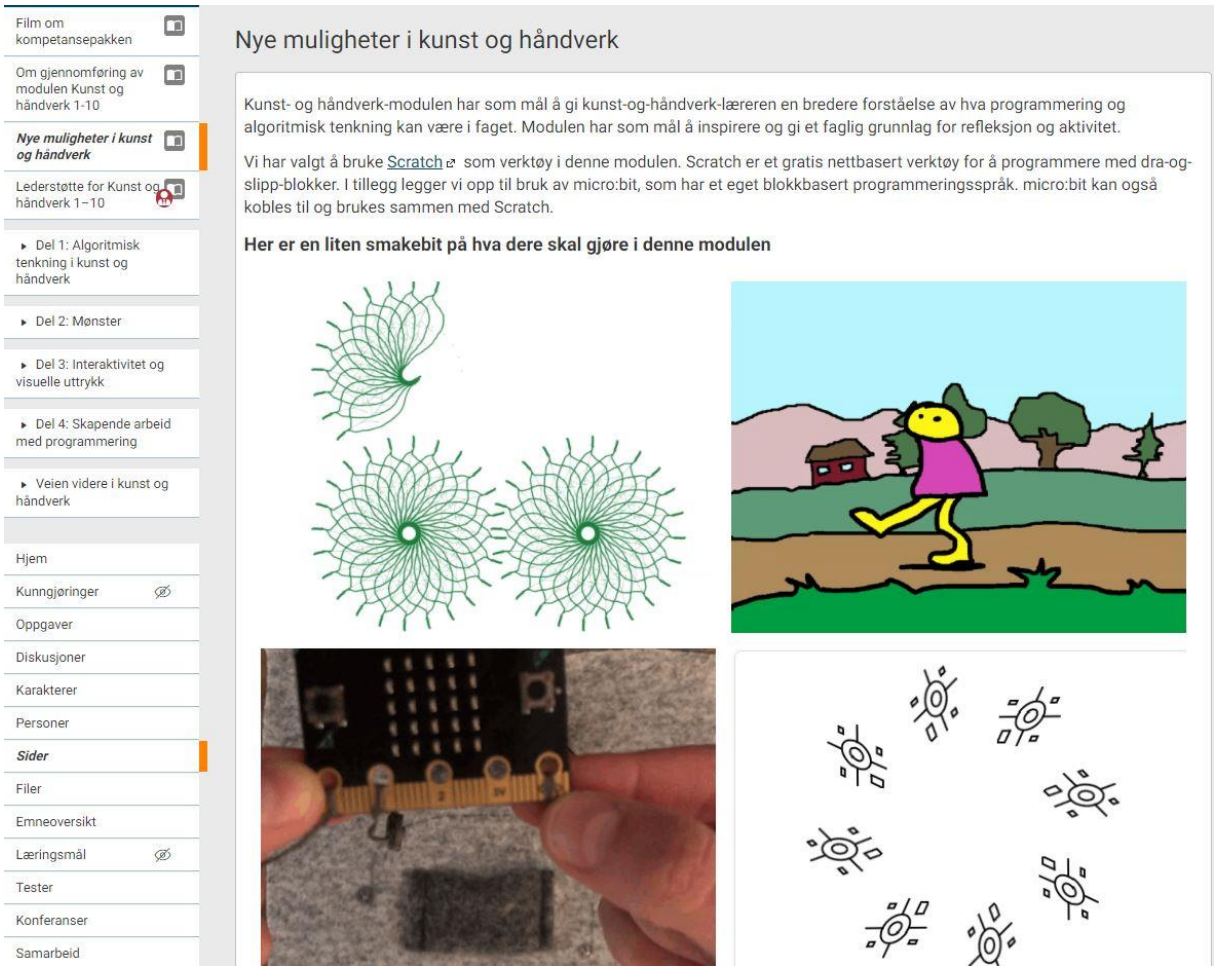


FIGUR 1. Skjermbilde fra Scratch der du ser blå kodekategori "bevegelse" til venstre, selve koden i midten, og testrute med en figur helt til høyre.

Overlappende prosjekter

Parallelt med undervisningsøktene i Scratch har jeg i samarbeid med en kollega utviklet lignende opplegg i andre utdanninger og prosjekter, noe som både blir påvirket av og påvirker undervisningen som er tema for denne artikkelen. Det har totalt vært tre parallelle prosjekter: Det første og mest sentrale prosjektet, er utvikling av kompetansepakker i programmering og algoritmisk tenkning på oppdrag for Utdanningsdirektoratet i 2019-2020 som en støtte til arbeid med læreplanverket (Utdanningsdirektoratet, 2022). Her utviklet vi en modul i Kunst og håndverk, sammen med andre samarbeidspartnere som utviklet tilsvarende moduler i musikk, matematikk og naturfag i tillegg til en fellesmodul som tar for seg det mer overordnede og fagovergripende innen programmering og algoritmisk tenkning. Disse kompetansepakkene er designet slik at lærere skal kunne gå gjennom innholdet i team- eller fellestiden, og jobbe praktisk, diskutere og reflektere i fellesskap. Pakkene er utviklet for å fungere uten en ekstern kursholder, og er utformet som læringsstier som bygger på hverandre i en kombinasjon av teori og praksis, sistnevnte i form av digitale oppgaver og diskusjons- og refleksjonsoppgaver. Figur 2 viser et skjermbilde fra Kunst- og håndverks-modulen, med fire visuelle eksempler fra oppgavene her. Tre av disse bildene viser eksempler utviklet med Scratch, og som er representative for oppleggene med faglærerstudentene.

To andre parallelle prosjekter som har pågått i den samme perioden, ligger under etter- og videreutdanning. *Programmering og algoritmisk tenkning i Kunst og håndverk*, en nyopprettet videreutdanning på 5 studiepoeng med lærere i Kunst og håndverk som primær målgruppe, ble gjennomført for første gang høsten 2021. Innholdet her bygget på erfaringer med tidligere undervisning i programmering, samt innhold vi utviklet til kompetansepakken. Det andre prosjektet, Desentralisert kompetanseutvikling i skolen (DeKomp), handler om skoleutvikling med skapermetodikk som overordnet tema. Også her er programmering aktuelt for skolene som velger å arbeide med dette, og enkelte workshops holdt for lærerne i 2021, bygger på erfaringer fra undervisningsøktene beskrevet i denne artikkelen samtidig som de har bidratt med impulser tatt tilbake inn i undervisningen igjen.



FIGUR 2. Skjermbilde fra Kunst- og håndverks-modulen i kompetansepakka i programmering og algoritmisk tenkning, viser eksempler på hvordan vi har brukt Scratch også i de aktuelle undervisningsøktene.

Pseudokode

Gjennom arbeidet med kompetansepakka fikk jeg erfaring med hva en pseudokode er via de som var fagansvarlige for fellesmodulen i programmering og algoritmisk tenkning. En pseudokode er kort fortalt en kort oppskrift du skriver selv der du planlegger hva koden din skal gjøre (Nygård, 2018). Den fungerer som et slags manus eller en liste over hva som skal skje når, gjerne skrevet ned punktvis. Det er en kode du selv skal forstå, og som senere omgjøres til en kode en datamaskin kan operasjonalisere. Dette har vi trukket fram i undervisningen, altså at studentene bør skrive en kort liste over hva deres koder skal gjøre. Men vel så viktig er at selve undervisningen gradvis er blitt strukturert som en pseudokode. Dette skjedde først i 2021, og kan se omtrent slik ut for en økt på 3 timer, der studentene skal gjøre 3 ulike oppgaver:

Gjenta 3 ganger:

1. Læreren viser (5-10 min)
2. Jobb med oppgave (20-30 min)
3. Del et skjermbilde og en kort beskrivelse av det du har gjort (Padlet) før vi tar en kort plenumsgjennomgang av erfaringene (10-15 min)
4. Pause (10-15 min)

Denne modellen har jeg valgt å kalle pseudokodemodellen, og er et metagrep som både forklarer hva en pseudokode er, og som gir studentene en forutsigbarhet over dagens økt slik at de vet når de kan få seg en pause og hva som forventes av dem i løpet av dagen. Oppgaven de skal jobbe med under punkt to blir spesifisert i plenum. En balanse mellom få, men relevante instruksjoner i forkant, egenutprøving, og mulighet for mer individuell avansering etter hvert, har vært sentral i denne undervisningen, selv om det her skjedde en utvikling fra de første øktene. I dag gis det ikke mer opplæring enn nødvendig helt i starten av en programmeringsøkt, men akkurat nok til at studentene kan prøve seg fram med oppgaven. Fordi det gjennomføres i tre runder er ideen at læring fra første runde skal bygges videre i de påfølgende to rundene. Tiden brukt på hvert punkt varierer ut fra oppgavens kompleksitet.

DESIGNBASERT FORSKNING

Med undervisning for faglærerstudenter i perioden 2018 til 2022 som en rød tråd, vil denne artikkelen belyse denne undervisningens utvikling som en designprosess som testes og endres fra år til år. Det er kun snakk om en kort 3 timers økt per klasse (2-3 klasser per kull) per år, gjennomført på vårsemesteret. Gjennom erfaring og evaluering av undervisningen kan en se på endringer fra år til år som iterasjoner i en designprosess. Hver økt kan anses som en prototype som testes, og som raffineres før neste gjennomføring (Bakker, 2018, s. 4), i dette tilfellet året etter. I mellomtiden foregår det andre aktiviteter parallelt, som både blir påvirket av, og påvirker disse undervisningsøktene. Sentralt i designbasert forskning, er å starte med å definere et problem, eller å definere noe som skal forbedres (McKenney & Reeves, 2018, s. 92). I dette tilfellet handler det om et behov for å introdusere programmering for kommende lærere i Kunst og håndverk, basert på kunnskap i 2017 om at dette skal bli med i kommende læreplaner. I stedet for å vente til planene skulle bli klare, ble det avgjort å å gi studentene en liten smakebit på hva programmering er og kan være i 2018. En hensikt var at alle til en viss grad skulle kjenne til dette, og at denne kjennskapen kanskje ville inspirere enkelte studenter til å finne ut mer på egenhånd og ut fra egeninteresse. Med designbasert forskning som en overordnet metode, er det i tillegg gjennomført anonym nettbasert spørreundersøkelse for 2021-kullet i tillegg til en uhøytidelig *Mentimeter*-undersøkelse underveis i øktene gjennomført i 2021 og 22.

I litteraturen brukt om designbasert forskning er det overlappende begreper som design-based research, design research og educational design research (McKenney & Reeves, 2018, s. 18; Bakker, 2018, s. 4). De ulike begrepene tilegnes litt ulike betydninger, men overordnet handler det om "developing theoretical understanding through the design and implementation of interventions in practice" (McKenney & Reeves, 2018, s. 18). I dette tilfellet handler det om å gradvis utvikle et undervisningsopplegg med et mål om å utvikle ny kunnskap som på sikt kan generere ny og forbedret praksis (Sevaldson, 2010). Ut fra dette vil jeg forholde meg til den norske oversettelsen *designbasert forskning*, og forståelsen av at det handler om å se på egen undervisning som iterasjoner i en designprosess. McKenney og Reeves (2018, s. 15) trekker for eksempel fram at i *educational design research* er iterative prosesser med innsikt, utvikling, utprøving og raffinering sentralt. Et annet aspekt som trekkes fram i designbasert forskning, er at det ofte kan dreie seg om å se potensialet i ny teknologi brukt i undervisning og læring (Bakker, 2018, s. 3). For denne artikkelens del starter det med et definert behov i utdanningen av framtidige lærere. Martinez og Stager (2019, s. 78) trekker også fram selve undervisningen som en iterativ prosess når de ser den i et metaperspektiv, i et avsnitt som egentlig handler om elevenes iterasjoner i kreative arbeidsprosesser.

Undervisning som prototype

Ideen om å forske på undervisningen i Scratch, oppsto i forkant av undervisningsøkta som ble gjennomført våren 2020, da vi også jobbet med å ferdigstille og teste kompetansepakka. Det vil si at også ideen

til denne artikkelen så vidt var moden på dette tidspunktet, og ideen om å sortere øktene ut fra design-basert forskning oppsto også rundt her. Selv om metoden som beskrives her ikke var i tankene mellom første undervisningsøkt i 2018 og neste gjennomføring i 2019, var det allikevel et fokus om å prøve noe nytt og være åpen for å endre og justere det til neste år. Ergo kunne disse øktene anses som iterasjoner i en designprosess også før jeg hadde satt ord på det.

Hver økt ble gjennomført for førsteårsstudenter i faglærerutdanningen i design, kunst og håndverk, og alltid som en 3 timers økt per klasse gjennomført i vårsemesteret. Hva som skjedde mellom disse øktene, vil også synliggjøres her. Mitt fokus er fra faglærers perspektiv, ikke studentens, og kvalitet på undervisning blir vurdert gjennom en helhetlig forståelse basert på det som studentene gjennomfører, og et inntrykk av deres motivasjon underveis i øktene. Med en liten 3 timers økt gange to eller tre grupper hvert semester, ligger denne økta til grunn for forbedring og tilpasning fra år til år. Økta er en prototype som testes, selv om jeg heller sier at den gjennomføres. For den behandles som et ferdig produkt for hvert studentkull, av respekt for studentene.

EMPIRI: DE FIRE FØRSTE ØKTENE MED FAGLÆRERSTUDENTER

Som det framkommer av Tabell 1 foregår det parallelle aktiviteter som både er påvirket av – og påvirker tilbake – utviklingen i hvordan Scratch-undervisningen legges opp for faglærerstudentene fra 2018 til 2022. Alle øktene foregår i vårsemesteret, og gjentas 2-3 ganger (hvert studentkull er altså delt inn i to eller tre grupper/klasser). All undervisning foretas av meg. I løpet av perioden blir LK20 lansert, og programmering går fra å være en varslet del til å bli en reell del av de nye læreplanene, også i Kunst og håndverk.

Rammene for de to første kullene, altså i 2018 og 2019, er at tre klasser ble delt inn i to større grupper, og samles i et IKT-rom med stasjonære PCer. Inndelingen hadde med en presset timeplan å gjøre, men en ulempe var at det ikke var nok PCer til alle. Samtidig var det et poeng at studentene skulle samarbeide. I disse øktene satt det 2-3 studenter samlet rundt hver stasjonære PC.

Til de første øktene på nyåret 2018 utviklet jeg en kort PDF-bruksanvisning med skjermbilder fra Scratch sammen med piler og tekst som forklarte grunnleggende funksjoner. Denne bruksanvisningen inneholdt to oppgaver som handlet om å animere figuren alle Scratch-prosjekter begynner med, en kattefigur ved navn Felix. I tillegg inkluderte kompendiet forslag til oppgaver hentet fra Kidsakoder.no (Lær Kidsa Koding, u.å.), som studentene kunne prøve hvis de ville. Jeg la også inn noen forslag til hvordan en av disse oppgavene kunne endres eller videreutvikles. Denne oppstarten var person-avhengig, der faglærer kunne ses som en ildsjel, altså en som er opptatt av noe, og brenner for at programmering er viktig å inn i undervisningen selv om det er nytt og kanskje fremmed for en del av studentene.

2019 er ganske likt som i 2018, men med en vri. Etter noen grunnleggende øvelser der studentene skulle få en figur til å bevege seg mot høyre og venstre, kunne de velge mellom noen problemløsningsoppgaver knyttet til denne utprøvingen, eller de kunne velge å gå videre til en annen oppgave. Det lå ingen løsningsforslag til disse problemløsningsoppgavene, og en oppgave var hvordan få figuren til å hoppe. De færreste valgte denne oppgaven, men en gruppe brukte hele økta på å komme opp med en brukbar løsning. Gruppa eksperimenterte og gjorde flere utprøvinger før de fikk figuren til å hoppe i en jevn bue, og lande et stykke til høyre eller venstre avhengig av hvilken retning den beveget seg. Her jobbet de med et problem, og endte opp med en overbevisende løsning som jeg ikke ville greid like bra selv. Andre studenter jobbet mindre konsentrert, og sa seg fort ferdig etter å ha gjort det som var beskrevet i de første øvelsene.

TABELL 1. Oversikt over undervisningen for FKH1 2018-2022 i sammenheng med andre aktiviteter.

År	Kurs for FKH1 (3 t pr klasse, vårsemester)	Overlappende utdanninger eller kurs	Andre overlappende aktiviteter
2018	Januar: Programmere en ball. Snurrige figurer. Sted: IKT-rom på campus.	Jan: Workshop UDE for nyutdannede lærere.	
2019	April: 1a) Få figur til å bevege seg. 1b) Få figuren til å gå. Problemløsning (hoppe). 2) Snurrige figurer fra kidsakoder. Sted: IKT-rom.	Jan: Kort innlegg KH-konferansen OsloMet (animasjon av analoge tegninger). Mai: Workshop i Processing for videreutd. Okt: Processing for FKH3.	HØST: Starter arbeidet med kompetansepakke i programmering og algoritmisk tenkning (Utdanningsdirektoratet).
2020	Mønster og animasjon (tegne selv) basert på kompetansepakke. Første hjemmeundervisning (Zoom). Resultater på Padlet (digital utstilling).	EVU / DeKomp høst: Workshop for lærere på Lørenskog (av en kollega).	Kompetansepakke utvikles og testes i flere omganger. Vårsemester: Utprøving av pilot på lærerseminar (Gardermoen). Høst: Lansering av kompetansepakke.
2021	Mønster og animasjon (tegne selv). "Hacke koden til en kollega". Hjemmeundervisning (Zoom). Klarere regi enn 2020. Pseudokodemodell innføres. Resultater på Padlet (digital utstilling). Undersøkelse via Nettskjema.	Høst: Undervisning i nytt emne Programmering og algoritmisk tenkning i Kunst og håndverk (5 stp). Følger både kompetansepakke og pseudokodemodell, og lærer noe verdifullt. Høst: Workshop i programmering for masterstudenter i allmennfag: Krasjkurs i Scratch, hel dag med micro:bit.	Desentralisert kompetanseordning (DeKomp), korte workshops for lærere i fellestid. Pseudokodemodell: Høre, prøve selv, dele, gjenta 3 x. 1 = fort i gang. 2 = fordype seg litt, 3= tenke undervisningsopplegg.
2022	Vil omrokere: Først animasjon (ikke tegne selv). Deretter mønster (tegne selv). Lage en side der de kan se og "shoppe" andre muligheter selv (pennefunksjon, quizfunksjon). Padlet. Hybridundervisning, på campus og valgfritt å delta hjemmefra. De fleste deltok på campus.		DeKomp skoleutvikling fortsetter.

I 2020 trådte koronapandemiens restriksjoner inn kort tid før semesterets Scratch-undervisning, og selv om vi hadde undervist digitalt ved tidligere anledninger, ble ikke denne økta planlagt godt nok på hjemmeundervisningens premisser. Det vil si at undervisningen med fordel kunne hatt en strammere regi. Noe annet som var nytt i 2020, var påvirkningen fra våre erfaringer med kompetansepakkene vi utviklet på oppdrag for Utdanningsdirektoratet, og vi brukte undervisningen som et sted for å teste

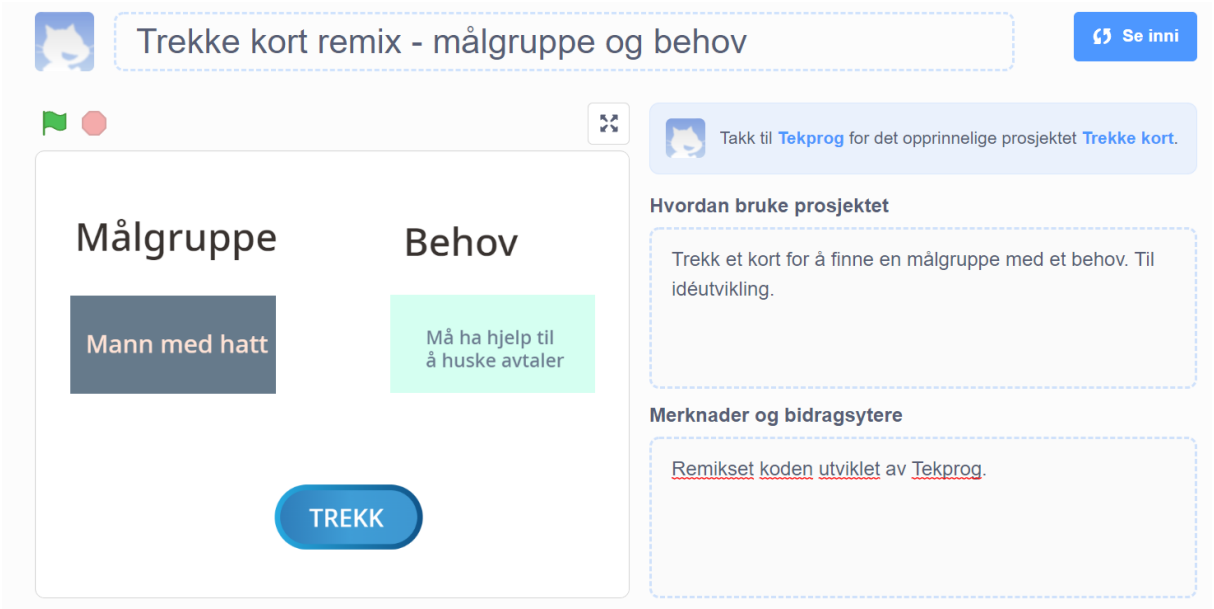
dette innholdet. Første praktiske oppgave i kompetansepakka i Kunst og håndverk gikk ut på at studentene skulle tegne et mønsterelement som de skulle programmere til å duplisere seg selv og danne et mønster (de kunne lage en bord, et flatemønster eller en mandala, som en kan se eksempel på i figur 1). Et skjermbilde av resultatet skulle deles på Padlet – en felles digital tavle – før de gikk i gang med neste oppgave, der de skulle få en figur til å bevege seg. Figuren kunne de tegne selv, eller finne i Scratchs egen samling med figurer. De skulle utvikle en interaktiv animasjon der figurens bevegelser skulle trigges av piltaster, lyd eller annen ytre påvirkning. Resultater fra oppgavene ble delt på Padlet, med en kort muntlig refleksjon i plenum til slutt.

I vårsemesteret 2021 var det også digital undervisning, men den skulle få en strammere regi enn i 2020. Dette etter et korona-år der vi hadde fått god erfaring med hvordan digital undervisning kunne struktureres. Den digitale økta ble brutt ned i mindre bestanddeler som både skulle sikre studentaktivitet og bidra til at faglærer kunne følge aktiviteten tettere enn hvis studentene bare jobbet fritt hjemme hos seg selv i flere timer uten å dele noe. I forkant av økta fikk studentene en klar oversikt over de kommende tre timene, i form av en tredelt oversikt der hver time skulle ha lik struktur, men med ulikt innhold. Jeg presenterte denne oversikten som en slags kode for dagen, en løkke som skulle gjentas tre ganger. I denne løkka var først korte instruksjoner fra lærer (trinn 1), deretter skulle studentene jobbe med en oppgave (trinn 2), før de til slutt – i trinn 3 – skulle legge et skjermbilde av resultatet på Padlet. Etter en kort plenumsgjennomgang om hva de hadde gjort og hvorfor, var det en kort pause, før løkka startet på nytt. Hver løkke varer en time inkludert pause. Hensikten med å legge opp økta slik er både å synliggjøre dagens gang (som jeg alltid gjør), men synliggjøre den på en måte som henger sammen med det jeg underviser i. Det er altså utviklet en egen kode for dagen, som et metagrep. Pseudokode brukes som en måte å planlegge egen kode på, og kan tydeliggjøre ens egen tankegang før en utvikler en reell kode (Nygård, 2018).

Oppgavene var som året før basert på kompetansepakka: 1) Tegne mønsterelement / kode et mønster. 2) Lage en enkel animasjon. Til slutt et nyere element: 3) Hacke koden til en kollega. Her gjentas oppgavene gjennomført i 2020, men altså med en strammere regi. Her skulle studentene i tillegg videreutvikle sin egen eller en medstudents kode gjennom å hacke den. Denne tredje oppgaven ble utviklet for at studentene skulle kunne lære gjennom å fikle med hverandres koder, med et mål om å gjøre kodene bedre, verre, rarere eller annerledes. Remikse er nok et mer riktig begrep enn hacke, men tittelen "hack koden til en kompis/kollega" virket som en mer fengende tittel. En viktig innføring gjort her, var å koble oppgavene til "prompts", eller kreative stikkord som ga begrensninger til oppgavene studentene skulle løse. Slike begrensninger kan bidra til å trigge nye ideer, da de får klare rammer som en tvinges til å tenke innenfor, i motsetning til at alt er fritt (Martinez & Stager, 2019, s. 63). I denne økta var disse begrensningene laget som et program i Scratch der en kunne trekke tilfeldige, digitale kort som kombinerte målgrupper med behov (Figur 3). Disse ofte absurde koblingene ble brukt som triggere i den kreative prosessen når studentene for eksempel skulle utvikle en enkel animasjon i Scratch. Som et metagrep kunne studentene også utforske og remikse koden i dette kort-prompt-programmet. Slike kreative begrensninger kan gjøres i kombinasjon med et mer overordnet tema som setter dagsorden og som er forankret i læreplaner.

I 2021 og 22 ble det samlet inn respons fra studentene med *Mentimeter*, en elektronisk plattform for samhandling, og som kan samle inn rask respons fra en forsamling. I dette tilfellet ble det brukt til å aktivisere studentene med to spørsmål. Det første spørsmålet var en avstemming helt i starten av økta, der studentene skulle rangere sin egen erfaring med programmering. Typisk fordeling av svarene her var ca. 14-15 svar pr klasse, hvor 70-75% svarte at de ikke hadde prøvd programmering før, mens noen svarte at de kunne noe. Det andre spørsmålet ble gitt helt til slutt i økta, der de kunne skrive 1-4 forslag til hvordan de selv ville brukt Scratch i en mulig undervisningsøkt. Disse svarene ble automatisk samlet i en ordsky der større skrift indikerer flere som svarer det samme. Figur 4 viser

resultatet av hva en av studentgruppene svarte på dette. Nederst i høyre hjørne vises antall personer som har svart, og dette fungerte som et bevis på at det var et visst frafall i løpet av den digitale økta.



FIGUR 3: Skjermbilde fra kreative kort (promtps) utviklet i Scratch.

I 2021 skulle studentene også svare på en digital anonymisert spørreundersøkelse i etterkant av undervisningen. Her skulle de blant annet rangere hvordan de opplevde undervisningens struktur og innhold, og de kunne supplere med kommentarer. Det var 16 respondenter i denne undersøkelsen. 4 av 16 rapporterte at de uten tvil mener at programmering er noe de vil utforske videre i studiet og senere i sitt liv som lærer, mens 9 svarte *kanskje* på det samme spørsmålet. Resten (3) svarte *usikker* eller *trolig ikke*. I et spørsmål om hva slags former for programmering de kunne tenke seg å utforske videre, svarte 11 *Scratch*, 3 *blokkbaserte mikrokontrollere som microbit/adafruit*, 3 svarte t*ekstbasert programmering på skjerm (P5.js / Processing etc),* 2 svarte *Arduino* og 4 svarte *usikker* (her kunne de velge flere svaralternativer, derfor blir totalt antall svar høyere enn 16). I spørsmålet om hvilken oppgave de likte best, svarte nesten alle animasjon. Den eneste som svarte noe annet, nevnte hacke-oppgaven. En annen av studentene trekker fram det å se andres koder som det mest lærerike, mens andre trekker fram det å jobbe selvstendig som mest lærerikt.

Ett av spørsmålene handlet om balansen mellom styrt opplæring og fri utforskning, med andre ord vektingen mellom mye og lite instruksjoner i starten av ei økt. Kun en respondent ønsket seg mer styrt opplæring i plenum. Forankret i Martinez og Stager (2019) har jeg gradvis blitt mer bevisst på å kaste studentene ut i det, med et klart mål, og med få instruksjoner i forkant slik at studentene heller kan prøve seg fram. Dette for å forhåpentligvis ufarliggjøre programmering litt. I stedet for at studenten skal følge lærerens demonstrasjon punkt for punkt, noe som kan gi et feil inntrykk av at det finnes én riktig måte å gjøre dette på. Ikke servere studentene svar på noe de ikke har rukket å lure på ennå. Ved å gå rundt og veilede i klasserommet vil læreren kunne hjelpe studentene der de er. En kan gå mange retninger innen programmering, og selv om en som lærer kan bli fristet til å vise studentene mange ulike muligheter som hen selv er opptatt av, er det også et poeng at studenter (og elever) selv skal kunne oppleve å lure på noe spesifikt.

FIGUR 4. Resultater fra avstemming med Mentimeter (på slutten av en av øktene i 2021).

Neste økt gjennomført mars 2022 var fysisk på campus, men tilbudt som hybrid slik at den også kunne følges hjemmefra. Viktig læring fra det siste året ble tatt med her, som å la studentene prøve seg fram med programmering før de får oppgaver som involverer tegning. Dette ble altså et skritt tilbake til hvordan det ble gjennomført i 2019, samtidig som jeg sluttet å støtte meg til oppskrifter utviklet i PDF. Det ble her en enda strammere regi på økta, med pseudokodemodellen som en konkret måte å synliggjøre vekselvirkningen mellom instrukser og studentenes egenarbeid. Ved at vi var tilbake på campus ble det enklere å oppfordre til samarbeid. Som lærer ble det igjen mulighet for spontane samtaler om hva studentene jobbet med, nå som det igjen var mulig å gå rundt i klasserommet og se hva studentene faktisk jobbet med. Kun noen få studenter valgte å følge undervisningen digitalt.

DRØFTING

Denne artikkelen forsøker å belyse hvilke perspektiver som kan være relevante å trekke inn når en skal introdusere programmering for faglærerstudenter i Kunst og håndverk. Basert på empirien deles drøftingen inn i tre perspektiver: 1) Vekselvirkning med overlappende prosjekter, 2) Instruksjoner og oppgaver, og 3) Videreutvikling og egenutvikling. Et sentralt mål har vært å gi alle studentene en ufarlig og leken vei inn i hva programmering kan brukes til, og la dem prøve seg fram. Et annet, mer indirekte mål, er at en viss prosent av hvert kull med nyutdannede studenter forhåpentligvis vil ta med seg programmeringens kjente og uoppdagede muligheter inn i sine yrkesliv.

Vekselvirkning med overlappende prosjekter

Undervisningen fra 2018 – 2022 kan anses som iterasjoner i en designprosess, der hver gjennomføring evalueres og endres før neste gjennomføring påfølgende semester. Samtidig har det foregått andre aktiviteter som påvirket og ble påvirket av denne undervisningen. Den første økta i 2018 hadde innflytelse på hvordan vi svarte på Utdanningsdirektoratets anbud om å utvikle kompetansepakke i programmering og algoritmisk tenkning i 2019, mens prosessen med å utvikle denne kompetansepakka i

stor grad har påvirket undervisningen i 2020. Utvikling av Kunst- og håndverks-modulen i denne kompetansepakka hjalp oss til å tenke kritisk over hvordan vi skulle balansere mellom instruksjoner og oppskrifter og konkrete og praktiske oppgaver. Fordi kompetansepakkene skulle designes for å bli brukt av lærere i fellesskap uten å ha noen eksterne kursholdere til stede, måtte vi tenke nøye igjennom hvor overkommelige og omfangsrike oppgavene kunne være for at lærerne skulle kunne jobbe kreativt og utforskende, lære noe nytt, men uten at de oppleve at det var altfor mye informasjon å sette seg inn i på forhånd. Denne vekselvirkningen har vært sentral i planleggingen og gjennomføringen av undervisningen også.

Arbeidet med kompetansepakka ga oss et repertoar som igjen kunne benyttes i undervisningen, og oppgavene i 2020 og 2021 var preget av dette. Her var det for eksempel fokus på at studentene skulle tegne selv før de kodet. Men da videreutdanningen i programmering i 2021 ble gjennomført fikk vi lengre tid til hvert opplegg, og i tillegg til mer fordypning i oppgavene ble det også mulighet for tettere dialog med studentene underveis. Her kom det blant annet noen muntlige tilbakemeldinger fra enkeltstudenter om at de gjerne skulle ha kodet litt før de begynte å tegne, for å få en bedre forståelse av hva tegningen skulle brukes til. Dette henger sammen med Martinez og Stagers (2019, s. 60) anbefalinger om å la elever (og studenter) oppleve mening med det de jobber med. En endring fra 2021 til 2022 var dermed å gå bort fra at alle skulle tegne selv før de begynte med kodingen. Det å starte med å tegne hadde allerede vist seg å være tidkrevende, og flere visste ikke helt hva tegningen skulle brukes til når de ikke hadde prøvd å kode. Det var altså mer hensiktsmessig at de heller måtte komme fortere i gang med å leke seg med koden for deretter forstå at dette også kunne kombineres med egne tegninger. Selv om jeg er tilhenger av at elever og studenter skal skape egne visuelle uttrykk i stedet for å bruke eksisterende bildemateriale, måtte dette nedprioriteres for å vie mer tid til å leke seg med programmeringen. Et mer langsiktig prosjekt kunne med fordel åpnet for bruk av egne digitale eller analoge tegninger eller fotografier.

Instruksjoner og oppgaver

I løpet av de fem gjennomførte undervisningsoppleggene ble informasjon i forkant av en økt gradvis kuttet ned til det minimale. I skapende, kreative aktiviteter i tråd med *maker movement* / skaperbevegelsen, er det et ideal om å unngå å gi for mye informasjon i forkant av en økt (Martinez & Stager, 2019, s. 54, 79; Skaperskolen, u.å.). Dette begrunnes blant annet med å holde motivasjonen oppe, og å sørge for å holde en viss energi blant elevene. Ifølge denne tankegangen er det bedre å komme i gang med noe litt raskt, prøve seg fram, utforske og bli kjent med materialet eller verktøyet, framfor å måtte høre en teoretisk forelesning først. Deretter er det stor sjanse for at en blir mer mottakelig for å videreutvikle kunnskap og ferdigheter underveis, over tid. Martinez og Stager (2019, s. 77) trekker fram en parallell artikulert av en av deres kollegaer som sammenligner skapende aktiviteter med baseball. Hvor mye teori om baseball trenger elevene før de faktisk kan begynne å spille? I samme kapittel trekker de fram en artikkel som viser at studenter som fikk mange instruksjoner i forkant av en økt hadde større vanskeligheter med å omstille seg til å løse et åpent problem i etterkant – og en annen som sammenlignet to studentgrupper der den ene gruppa fikk prøve seg fram med en simulering før de leste teorien, mens den andre gruppa leste teori eller så videoer før de fikk jobbe praktisk med simulatoren. Her gjorde førstnevnte gruppe det bedre på tester (2019, s. 80).

Dette handler ikke om å slutte å gi instruksjoner, men å holde litt igjen på informasjon som kanskje ikke er nødvendig før man setter i gang med en aktivitet. Samtidig bør en viss opplæring til for å sikre at eleven behersker et verktøy. Martinez og Stager (2019, s. 78) foreslår en iterativ tilnærming, der hver gjennomføring bygger på den neste. På denne måten kan en komme raskt i gang med noe, uten for mye instruksjon i forkant. Instruksjonene kan heller spres slik at de kommer på strategiske tidspunkter senere i prosessen eller undervisningsøkta, etter hvert som eleven har fått mer erfaring å bygge videre på (s. 78-80). Hvor mye må elever og studenter vite for å komme i gang med en aktivitet,

og hva kan de heller høre om på et senere tidspunkt? Ved å strukturere en undervisningsdag med pseudokodemodellen, legges det til rette for flere runder av tilsynelatende samme opplegg, men med mulighet for gradvis dypere forståelse for hver gjennomføring. Pseudokodemodellen har vært en måte å skape forutsigbarhet i timen, og samtidig synliggjøre hvordan en økt er bygget opp. Samtidig er den tenkt som et metagrep som viser hvordan en pseudokode kan se ut, da dette egentlig er en måte å planlegge en kode på før man utformer den i et programmeringsverktøy.

Resultatene fra spørreundersøkelsen i 2021 viser en positiv holdning til undervisningen i programmering. De fleste skrev at kurset var lærerikt, mens noen meldte fra at dette var vanskelig. På et spørsmål om hvordan det fungerte med vektingen av få instruksjoner i forkant og mer tid til utprøving, svarte 10 studenter "god balanse" mens 6 svarte det forhåndsdefinerte alternativet "Ok, men skulle hatt mer tid til å jobbe med oppgavene". Det vil si at ingen huket av på valget om at de ønsket seg flere instruksjoner i forkant. Selv om det er få respondenter, kan dette allikevel fungere som en bekreftelse på at det er en verdi i å la studenter (og elever) prøve seg fram på egenhånd snarere enn å bruke tiden på å høre på lærerens instrukser (Martinez & Stager, 2019, s. 54 og 79). En erfaring fra de siste gjennomføringene, i 2021 og 22, etter at instruksjoner i forkant ble kuttet drastisk ned, er at studentene gjerne har stilt spørsmål som før kunne være en del av den mer instruksjonsbaserte undervisningen. Med andre ord har studentene lurt på ting etter hvert som de har prøvd seg fram, og dermed fått veiledning relevant for det de holder på med. Dette i stedet for at de skulle få alle svarene servert på forhånd, svar på spørsmål de ikke har stilt, og som deretter trolig ikke vil feste seg. Med færre instruksjoner og mer åpen utprøving ut fra bestemte kriterier, erfarer studentene mer selv, og rekker å lure på bestemte ting. De stiller altså spørsmål – og får svar – i stedet for å få en haug med svar før de i det hele tatt vet hva de lurer på.

Resultatene fra Menti-ordskyen i 2021 viser at de fleste skriver animasjon når de skal nevne hva som kunne vært mest aktuelt for dem å undervise i dersom de skulle undervise i Scratch selv. De fleste skriver forslag basert det de har gått igjennom i dagens økt, og det er forståelig at de ikke kan se for seg andre temaer når de ikke har rukket å utforske flere muligheter selv. En felles erfaring fra 2020 og 2021 er at ikke alle studenter er delaktige når undervisningen foregår digitalt, verken i den noe løse økta i 2020 eller i den bedre strukturerte økta i 2021. Flere av studentene som tilsynelatende var til stede på Zoom, var der muligens kun for å unngå å få fravær. Av omtrent 14-15 deltakere var det ikke alle som løste oppgavene som ble gitt, og da jeg spurte i plenum om de faktisk var til stede og hvorfor de ikke løste oppgavene, kom det ingen respons. Mentimeter-avstemmingen avslørte også at en del falt fra i løpet av økta i og med at det var færre svar i spørsmålet som ble gitt på slutten av dagen sammenlignet med introduksjonsspørsmålet. I stedet for å fokusere på studentene som ikke var aktive, var det mer konstruktivt å holde oppmerksomheten rettet mot de som faktisk deltok i undervisningen. Som forventet var det mer aktiv deltakelse i 2022, da undervisningen igjen var fysisk. Her ble oppfølgingen av studentene tettere, i og med at jeg endelig kunne gå rundt og se hva de holdt på med.

Studentene valgte ulike tilnærmingsmåter til oppgavene. Fra 2020 til 22 valgte de fleste å utvikle visuelle uttrykk basert på de to oppgavene mønster eller animasjon. Noen gjorde seg fort ferdig, og sa seg fornøyd med første versjon av animasjonen eller mønsteret de skulle lage, mens andre hadde prosesser der de fikk utvikle og videreutvikle egne ideer. Der noen fokuserte mer på å utvikle egne visuelle uttrykk, som for eksempel å tegne egne figurer til bruk i koden, hadde andre mer fokus på å jobbe med selve koden mens det visuelle elementet kunne være hentet fra figurene og bakgrunnene som ligger inne i Scratchs eget bildebibliotek. I hackeoppgaven valgte noen å endre på det visuelle snarere enn på koden, mens andre jobbet med å endre på og videreutvikle selve koden til sin medstudent. Poenget med denne oppgaven var egentlig å hacke eller remikse *koden* til en kollega, ikke det visuelle uttrykket, men det var kanskje ikke understreket fra min side. Det er også lærerikt å se hvordan studenter griper fatt i lærers instrukser som kan være åpne for tolkning i retninger som ikke var forutsett. Til framtidige økter kan det være konstruktivt å beholde dette tolkningsrommet fordi det

kan åpne opp for enda flere muligheter. Hvorfor ikke beholde en kode, men forandre på det visuelle uttrykket?

Videreutvikling og egenutvikling

Scratch har vært et tilgjengelig valg for enkel innføring i programmering fordi det ikke krever forkunnskaper eller ekstrautstyr utover de digitale enhetene som allerede brukes i et vanlig klasserom, enten lærerne og elevene bruker laptop eller nettbrett. Med Scratch kan en gå direkte til den skapende og problemløsende delen i programmering i stedet for å først måtte lære seg språket i et tekstbasert programmeringsspråk og hva det innebærer av krav til korrekt syntaks og tegnsetting (Martinez & Stager, 2019, s. 180). En kan fort oppleve mestring, og samtidig er det mulig å utvikle mer avanserte prosjekter. Når en justerer sin egen kode eller remikser prosjektet til en klassekamerat, jobber man direkte i materialet – som om den digitale koden er en form for materiale (Hoebeke et al., 2021). Elever kan diskutere et prosjekt med hverandre eller med læreren, med andre ord bruke muntlige ferdigheter til å sette ord på egen prosess, i tråd med både grunnleggende muntlige ferdigheter og kjerneelementet kunst- og designprosesser i Kunst og håndverk (Kunnskapsdepartementet, 2019). Koden og resultatet på skjermen blir en måte å synliggjøre en prosess på, inkludert verdien av å sette ord på feilene en gjør i koden (Papert, 1993, s. 180). Det kan også anses som et "object-to-think-with", ifølge Seymour Papert (1993, s. 11) i sin omtale av et tidligere programmeringsspråk han utviklet, en forløper til Scratch. Det vil si at barn og unge som bruker dette i stedet for å høre en lærer snakke om et tema, opplever å jobbe direkte med et *educational object* som fungerer som en krysning mellom *cultural presence, embedded knowledge, and the possibility for personal identification* (s. 11). Gjennom fikling og eksperimentering, prøving og feiling, testing og endring, kan visuelle uttrykk og interaktive produkter utvikles. Det viktigste her er at programmeringen er enkel å starte med, men det er må ikke nødvendigvis være Scratch. Mange alternativer kunne fungert, og tekstbasert programmering ville også åpnet for andre muligheter. Men spesielt når tiden er begrenset er det nok en fordel at programmeringen er blokkbasert i starten. Da kan vi beholde fokus på å løse oppgaver og problemer samtidig som det ikke er nødvendig med innlæring av et tekstbasert kodespråk (Martinez & Stager, 2019, s. 180).

En faktor vi har til gode å se nærmere på, er studentenes resultater og ambisjonsnivå i mer langvarige prosjekter. Det nærmeste vi foreløpig kommer, er de få studentene som valgte programmering i sine arbeidskrav eller eksamensbesvarelser i etterkant av undervisningen i Scratch. Disse har gjerne hatt noe erfaring med programmering fra før. En kandidat som leverte eksamen i 2021 hadde en del forkunnskaper, og valgte et tekstbasert programmeringsspråk framfor Scratch, mens kandidaten i 2022 utviklet et ganske omfattende mattespill i Scratch. Dette spillet kombinerte visuelle uttrykk med interaktive matteoppgaver spredt over flere brett. En slik dypere forståelse vil nok oppnås gjennom oppgaver som går over en lengre periode. Dette har vi også sett tendenser av i videreutdanningskurset med flere og lengre programmeringsøkter. En mer kompleks utvikling i Scratch kunne for eksempel kombinert digital bildebehandling, som ved å utvikle en egen figur i et vektortegneprogram eller redigere fotografier på forhånd. Eller en kan jobbe langsiktig med å utvikle en mer kompleks kode, som for eksempel et spill eller en interaktiv fortelling.

Gjennom planlegging av undervisningsøkter over flere år, i tillegg til utvikling av fagstoff i kompetansepakker, har jeg og et par kolleger selv merket både glede, mestring og motivasjon gjennom muligheten til selv å fordype oss i ulike programmeringsverktøy over tid. Ved å planlegge ny undervisning i et noe ukjent fagområde som i begynnelsen kan virke fremmed for studenter motivert for mer tradisjonelle kunst- og håndverksaktiviteter, har ideen om å anse hver økt som en prototype som kan forbedres fra gang til gang vært en måte å nærme seg dette nye på. Her har det vært en fordel at øktene er korte, med lavere fallhøyde dersom en skulle feile helt. Når nye økter planlegges har jeg jobbet videre med ideer fra egen utprøving i Scratch, noe som igjen har generert andre ideer til nye utprøvinger. Noen av ideene har kanskje vært for sære til å ta fram i undervisningen, men denne egenutforskningen har

kommet gradvis gjennom prøving og feiling, testing og deling med en kollega. Sammen med denne kollegaen har hacking, eller remiksing, av hverandres koder har også vært en kilde til læring, noe som igjen har påvirket oppgavene etter hvert som de har utviklet seg. Med andre ord merker jeg at jo mer jeg selv jobber i Scratch og andre programmeringsverktøy, og jo mer jeg samarbeider med denne kollegaen, jo flere ideer får jeg til hva dette kan brukes til. Dersom denne utforskertrangen kan overføres til noen av studentene, og indirekte elever i skolen, vil vi ha oppnådd et stort mål i tråd med læreplanens idealer om skaperglede og utforskertrang (Kunnskapsdepartementet, 2017b), samt kjerneelementet kunst- og designprosesser (Kunnskapsdepartementet, 2019).

Hvor står vi etter disse fem iterasjonene på fem år? Undervisning med programmering er et pågående arbeid som vi vil tilpasse og justere, og fortsette med så lenge det er aktuelt som utgangspunkt for utvikling av interaktive og visuelle uttrykk i Kunst og håndverk. Vi utdanner lærere som kan bruke programmering med elever på fagenes premisser, og på måter som legger til rette for motivasjon, skaperglede, utforskertrang, problemløsning, deling og samarbeid. Scratch kan fungere som et objekt som synliggjør tenkning og læring i og på tvers av fag. Sammen med dette vil arbeidsmetoder som tinkering (eller fikling), som også kan ses som en leken tilnærming (Martinez & Stager, 2019, s. 39-40), føre til andre resultater enn med oppgaver der hele læringsforløpet er forhåndsdesignet av læreren. Pseudokodemodellen brukes som en struktur av en økt, og et metagrep, og er mitt foreløpige bidrag til nyskaping. Et sentralt neste steg i Scratch-undervisningen vil nok være å la hackingen – eller remiksingen – av andres kode få større plass, da studentenes respons i spørreskjemaet viste at flere meldte at de lærte av å se og å fikle med andres koder. En forutsetning er nok at de har fått nok tid til å utvikle en egen kode først. Ved hjelp av pseudokode vil ens egne ideer bli synliggjort på papiret, og ved å bryte en økt ned i mindre bestanddeler med klare tidsrammer vil et mulig steg også være å holde en økt der selve pseudokoden får større fokus. Dette for å se hvordan det vil påvirke utfallet av studentenes egne arbeidsprosesser.

REFERANSER

Bakker, A. (2018). *Design Research in Education : A Practical Guide for Early Career Researchers*. Routledge. https://doi.org/10.4324/9780203701010

Hoebeke, S., Strand, I., & Haakonsen, P. (2021). Programming as a New Creative Material in Art and Design Education. *Techne serien - Forskning i slöjdpedagogik och slöjdvetenskap*, *28*(2), 233–240. https://journals.oslomet.no/index.php/techneA/article/view/4325

Høibo, I.H. (2022, 18. januar). Eit nettbrett til kvar elev gir ikkje digital kompetanse. *Forskersonen.no.* https://forskersonen.no/barn-og-ungdom-kronikk-meninger/eit-nettbrett-til-kvar-elev-gir-ikkje-digital-kompetanse/1965235

Kunnskapsdepartementet. (2017a). *Mer koding og teknologi inn I skolen* (Pressemelding nr. 84/17). Regjeringen Solberg. https://www.regjeringen.no/no/dokumentarkiv/regjeringen-solberg/aktuelt-regjeringen-solberg/smk/pressemeldinger/2017/mer-koding-og-teknologi-inn-i-skolen/id2568375/

Kunnskapsdepartementet. (2017b). Skaperglede, engasjement og utforskertrang. *Overordnet del – verdier og prinsipper for grunnopplæringen.* Fastsatt som forskrift ved kongelig resolusjon. Læreplanverket for Kunnskapsløftet 2020. https://www.udir.no/lk20/overordnet-del/opplaringens-verdigrunnlag/1.4-skaperglede-engasjement-og-utforskertrang/?lang=nob

Kunnskapsdepartementet. (2019). Kjerneelementer. *Læreplan i Kunst og håndverk (KHV01-02).* Fastsatt som forskrift. Læreplanverket for Kunnskapsløftet 2020. https://www.udir.no/lk20/khv01-02/om-faget/kjerneelementer?lang=nob

Lær Kidsa Koding. (u.å.). *Scratch*. https://oppgaver.kidsakoder.no/scratch

Martinez, S. L. & Stager, G. (2019). *Invent to learn : Making, Tinkering, and Engineering in the Classroom* (2. utg.). Constructing Modern Knowledge Press.

McKenney, S. & Reeves T.C. (2018). *Conducting Educational Design Research* (2. utg.). Routledge. https://doi.org/10.4324/9781315105642

Nygård, K. (2018). *Programmering i grunnskolen- hvordan komme i gang?* Pedlex.

Papert, S. (1993). *Mindstorms : Children, Computers and Powerful Ideas.* (2. utg.). Basic Books.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11) , 60–67. https://doi.org/10.1145/1592761.1592779

Resnick, M. & Rosenbaum, E. (2013). Designing for tinkerability. In M. Honey & D. Kanter (Eds.), *Design, make, play: Growing the next generation of STEM innovator*s (pp. 163–181). Routledge. https://doi.org/10.4324/9780203108352

Sevaldson, B. (2010). Discussions & Movements in Design Research. *FormAkademisk*, *3*(1). https://doi.org/10.7577/formakademisk.137

Skaperskolen. (u.å.). Kløkt. *Pedagogisk verktøykasse*. https://skaperskolen.no/pedagogisk-verktoykasse/#klokt

Steenbuch, B. (2016, 8. okt). Oslo kommune skal lære niåringer å programmere. *Aftenposten*. https://www.aftenposten.no/oslo/i/dBV2X/oslo-kommune-skal-laere-niaaringer-aa-programmere

Utdanningsdirektoratet. (2018). Kunst og håndverk/Doudji/Duodje/Duedie (Høringsbrev av 05.03.2018). https://hoering.udir.no/Hoering/v2/197?notatId=356

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

Utdanningsdirektoratet. (2022). *Kompetansepakker.* https://www.udir.no/kvalitet-og-kompetanse/kompetansepakker/