Høgskolen i Østfold
Avdeling for informasjonsteknologi
ITF32012 Bacheloroppgave
18.05.2016

Bacheloroppgave: Programmering i skolen

Hoveddokument

BO16-G12:
Anders Bolt-Evensen
Tobias Alvik Hagen
Anders Heggestøyl

HØGSKOLEN I ØSTFOLD

Avdeling for informasjonsteknologi
Remmen
1757 Halden
Telefon: 69 21 50 00
http://www.hiof.no/

Prosjektkategori: Utredning	Rapporttittel: Programmering i skolen
Omfang i studiepoeng: 20	Dato: 18.05.2016
Fagområde: Informasjonsteknologi, pedagogikk	Forfattere: Anders Bolt-Evensen, Tobias Alvik Hagen, Anders Heggestøyl
Oppgavetilgjengelighet: Fritt tilgjengelig	Veileder: Børre Stenseth
Avdeling / linje: Avdeling for Informasjonsteknologi (alle)	Gruppenummer: BO16-G12
Utført i samarbeid med: Os Skole, Strupe Ungdomsskole, HiØ/IT	Kontaktperson(er): Harald Holone, Odd Tore Kaufmann

Ekstrakt: Bruk av programmering i grunnskolen er et tema som blir stadig mer aktuelt. Bachelorgruppen undersøkte bruken av programmering som pedagogisk verktøy i matematikkundervisningen ved 7. og 9. trinn. Rapporten tar for seg dokumentasjon rundt denne uttestingen, drøftinger rundt bruk av programmering i skolen, samt evaluering av relevante verktøy. Rapporten er bygget opp i tre hoveddeler: Uttestingen ved Os barneskole, uttestingen ved Strupe ungdomsskole, og evaluering av verktøy.

Emneord:

Programmering	Pedagogikk	Undervisning

Sammendrag

Prosjektgruppen gjennomførte kortvarige undervisningsopplegg for grunnskoleelever ved 7. og 9. trinn, med hensikt om å undersøke bruken av programmering som pedagogisk verktøy i matematikkundervisningen. Oppleggene bestod av å lære elevene grunnleggende programmering, og deretter sette dem til å løse matematiske oppgaver ved bruk av programmering.

Elevene ved både 7. og 9. trinn behersket grunnleggende programmering raskere enn forventet. Ved endt undervisning demonstrerte alle elevene evnen til å lese og forstå simpel kildekode, og evnen til å endre kildekoden til eksisterende programmer, for å oppnå gitte mål. Tilnærmet ingen elever utviklet evnen til å lage egne programmer fra bunn av. Dette kan imidlertid skyldes den begrensede tiden som var til rådighet for undervisningen.

Det visuelle programmeringsspråket Scratch viste seg å være egnet for bruk i barneskolen, og for ungdomsskolen viste utviklingsmiljøet Processing seg å være et egnet verktøy. Andre alternativer diskuteres i prosjektrapporten.

Hvorvidt programmering som pedagogisk verktøy har nok nytteverdi for økt matematikkforståelse til å kunne rettferiggjøre investeringen som følger med, er umulig å konkludere ut i fra dette prosjektet alene. Prosjektet viser imidlertid at det er gjørbart å få grunnskoleelever i gang med enkel programmering, som en alternativ måte å løse matematiske problemer. Forhåpentligvis kan dette bachelorprosjektet være med på å bane vei for videre prosjekter av større omfang, med lengre undervisningsopplegg, som kan danne et bedre grunnlag for programmeringens nytteverdi som pedagogisk verktøy.

Takk til

Først og fremst vil vi takke vår veileder, Børre Stenseth. Du satte oss stadig på rett spor, og kom med mange vise ord. Vi setter stor pris på hvor tilgjengelig du har vært gjennom prosjektperioden. Din veiledning har vært vital for prosjektets suksess.

Vi vil også takke Odd Tore Kaufmann, vår kontaktperson i forbindelse med ProgPed. Vi takker for din deltakelse i undervisningsoppleggene – både i forkant av og under undervisningsøktene, samt for veiledning i bachelorprosjektet.

Videre takker vi lærerne som deltok i prosjektet: Aasmund Lunde ved Os barneskole, og Kristine Marie Bukholm og Cecilie Helen Andersson ved Strupe ungdomsskole.

Vi må selvsagt også gi en stor takk til alle de ivrige elevene som deltok i matematikkundervisningen.

Figurliste

Liste over de ulike bildene. Generes til slutt.

Kodeliste

Lister opp kodesnutter som har vært brukt. Først og fremst relevant for kodesnutter i kaptittel 5.

Innholdsfortegnelse

Obs: Generer ToC med sidetall til slutt.

Vedlegg

1 Introduksjon

1.1 Bakgrunn

Dette bachelorprosjektet utgjør grunnlaget for Bacheloroppgaven (ITF32012) ved Høgskolen i Østfold, avdeling for informasjonsteknologi. Prosjektgruppen BO16-B12 består av:

Anders Heggestøyl er 23 år og bor i Halden. Han har bodd mesteparten av sitt liv i Horten, og flyttet til Halden for å studere informasjonssystemer ved Høgskolen i Østfold, avdeling for informasjonsteknologi. Anders er interessert i IT-ledelse, webutvikling og søkemotoroptimalisering.

Tobias Alvik Hagen, 28 år, holder til i Halden. Opprinnelig født og oppvokst i Arendal, men har også vært innom Bergen, Trondheim, Tyskland og USA. Studerer informasjonssystemer ved Høgskolen i Østfold. Tobias er opptatt av samspillet mellom design og teknologi.

Anders Bolt-Evensen er 28 år gammel og kommer fra Sarpsborg, hvor han har bodd hele sitt liv. Fritiden brukes til gaming, være sammen med venner, og å se på Sarpsborgs ishockeylag, Sparta. Anders er glad i å se på ishockey og sport generelt. Av faglige ting liker han å jobbe med databaser og programmering.

Bacheloroppgaven inngår som en del av prosjektet “Programmering som fag og pedagogisk verktøy i skolen”, forkortet “ProgPed”. ProgPed er et forprosjekt som avdelingene for informasjonsteknologi og lærerutdanning ved Høgskolen i Østfold startet i 2015. Tre skoler i Halden (Os barneskole, Strupe ungdomsskole og Halden videregående skole) deltar i dette prosjektet. Oppdragsgiver er Høgskolen i Østfold, avdeling for informasjonsteknologi.

Våren 2016 testet ProgPed ut bruk av programmering som pedagogisk verktøy i matematikkundervisningen ved disse skolene. Bachelorgruppen bidro med observasjon og undervisning av elever, samt testing og evaluering av verktøy. Uttestingen ved Halden videregående skole foregikk etter bachelorprosjektets innleveringsfrist, og vil følgelig ikke dokumenteres i dette sluttdokumentet. Gruppemedlemmene ble i stedet leid inn i etterkant av bachelorprosjektets sluttleveranse, for å delta i undervisningen ved Halden videregående skole.

1.2 Problemstilling

Prosjektgruppen har deltatt i uttestingen av bruk av programmering som pedagogisk verktøy i matematikkundervisningen ved 7. og 9. trinn. Dette inkluderer å hjelpe til i undervisningstimene, å opplære lærere om programmering og bruk av relevante verktøy, og å utvikle undervisningsmateriale.

1.3 Problemformulering

Prosjektet, som bacheloroppgaven er en del av, skal se på hvorvidt programmering kan brukes som pedagogisk verktøy for å lære skoleelever matematikk. Kan det føre til bedre matematikkforståelse for barn? Hvordan kan programmering best implementeres i undervisningen? Hvilke deler av matematikkpensumet drar mest nytte av programmering som verktøy? Vil fordelene rettferdiggjøre investeringen i programmeringen?

1.4 Rapportoppbygging

Hoveddelen av rapporten er delt opp i to undersøkelser: Uttesting ved Os skole, og uttesting ved Strupe ungdomsskole. Disse behandles som separate undersøkelser, og inneholder hvert sitt metode-, obervasjons-, diskusjons- og konklusjonskapittel. Videre har rapporten et overordnet diskusjonskapittel, hvor undersøkelsene sammenlignes med hverandre, og en overordnet konklusjon for hele prosjektet. Mye av innholdet i hver undersøkelse er strukturert kronologisk.

Utenfor undersøkelsene evalueres relevante programmeringsspråk og utviklingsmiljøer. Vurderingene gjøres delvis på grunnlag av teori og testing, og delvis på grunnlag av erfaringer tilegnet gjennom undersøkelsene.

1.5 Kritikk

Det er viktig å avklare at dette prosjektet ikke vil være i stand til å trekke noen definitive konklusjoner. Grunnlaget for undersøkelsene i dette prosjektet baseres på subjektive observasjoner og erfaringer. Det opereres med et svært begrenset datagrunnlag, som

er hentet gjennom kvalitative metoder. Følgelig vil ikke prosjektet kunne påstå noen objektive funn.

Videre må det nevnes at ingen av medlemmene i bachelorgruppen har bakgrunn eller kompetanse innen pedagogikk eller didaktikk. Det blir derfor utenfor bachelorprosjektets rammer å dekke problemstillinger innen didaktikk. Dette overlates til Odd Tore Kaufmann, matematikkdidaktiker og førsteamanuensis ved Høgskolen i Østfold, som også deltar i ProgPed.

2 Teori

For å underbygge de valgene som ble tatt, samt de konklusjoner gruppen har kommet frem til, legges det ved et utvalg av relevant teori. Teoriene omhandler nåværende status om programmering i skolen, observasjonsmetodikk, grunnlaget for prosjektet, samt læringsrammer knyttet til god anvendelse av IKT i undervisning. Dette kapitlet presenterer relevant teori gruppen har anvendt opp mot oppgaven.

2.1 Ønsket om programmering i skolen

Det har oppstått et større fokus på å få gjennomslag for programmering i skolen. Dette gjelder både nasjonalt, så vel som internasjonalt. Det debatteres offentlig i tidsskrifter, blant politikere og akademikere (Klovning, 2015; Hofseth, 2013; Rettberg, 2013; Lær Kidsa Koding, 2016). I Norge er det opprettet kodeklubber, som LærKidsaKoding, flere steder i Norge (LærKidsaKoding, 2016). Internasjonalt har man store initiativer som Code.org, hvor blant annet president Barack Obama har frontet initiativet ved å delta i Hour of Code (Code.org, 2014). I rapporten *Computing our future* (European Schoolnet, 2015) fremgår det hvilke europeeiske land som er aktive i utprøvningen av programmering i skolen. Her presenteres også hvilke konsepter innen programmering det fokuseres på.

2.2 Learning by doing

Det var fra starten av i prosjektet et ønske om å forholde seg til et ikke-symbolsk resultat i oppgavene. En del av oppgavene, som er blitt utarbeidet i løpet av prosjektet, relaterer seg også til figurer og ikke bare tall. En del av verktøyene som presenteres i kapittel 6, samt de som blir anvendt i utprøvningene, kan sies å bygge på Jean Piagets

læringsteorier om konstruktivisme og prinsippet “Learning by doing” (Logo Foundation, 2015). Videre viser Alan Kay teorien doing with images makes symbol i sin videopresentasjon ved samme navn. Her går han gjennom psykologien, filosofien og læringsteorier bak det å bruke datamaskiner som verktøy. Kay viser også til at barn, så unge som 22 måneder, lærer å bruke datamaskiner (*Alan Kay: Doing with Images Makes Symbols Pt 2*, 1987). Dette bekreftes også i undersøkelsen *Kids learning digital skills before life* skills utført av sikkerhetsprogramfirmaet AVG Technologies (AVG Digital Diaries, 2015). At barn lærer å anvende datamaskiner i tidlig alder, kan være med på underbygge innføringen av programmering i tidlig alder.

2.3 Notat: Koding i skolen

Programmering i skolen er per dags dato ikke en del av læreplanene, foruten enkelte studieretninger i den videregående skole. Kristine Sevik presenterer, i notatet *Koding i skolen*, ulike måter å innlemme programmering som fag i skolen (Selvik, 2015). Ett av alternativene som presenteres er å legge programmering inn under ett eller flere realfag. Dette praktiseres i dette prosjektet, ved å inkludere programmering i matematikkundervisningen.

2.4 Hensiktsmessig bruk av IKT i undervisningen

For å lykkes med programmering som et pedagogisk verktøy i matematikkundervisningen, er det viktig å fokusere på rammebetingelser for god bruk av IKT. Det er viktig å tilrettelegge undervisningen, klasserommet og læreren, så godt som mulig, for å få et fruktbart resultat. I notatet *Hensiktsmessig bruk av IKT i undervisningen - en veileder* fra Senter for IKT i utdanningen (2015), trekkes det frem rammebetingelser og krav til læreren.

2.5 Kvantitativ metode

Kvantitativ metode går ut på å måle antall enheter opp mot en hypotese, for deretter å bekrefte hypotesen. Siden dataen består av tall som representasjoner, er det vanskelig å måle opplevelser og erfaringer fra respondentene. Dataene analyseres gjerne i form av statistiske eller økometriske metoder (Dahlum, 2014).

2.6 Kvalitativ metode

For å kartlegge og begrunne menneskers opplevelser og erfaringer, det som ikke lar seg tallfeste, anvender man kvalitative metoder. Her kommer man mer i dybden på informasjonen man har samlet inn (Dahlum, 2014).

2.7 Observering som metode

Ettersom gruppen valgte observasjon som metode for datainnsamling, er det naturlig å se på teori rundt det å observere. Før man observerer er det lurt å vite hvordan man skal håndtere sanseinntrykk, hva som gjør en til en god observatør, hvordan man skal forholde seg til de involverte og problemer ved observasjoner (Bjørndal, 2011, s. 32-62).

En en mulig faktor som er med på å påvirke resultatene fra observasjoner, er entusiasmen til deltakerene i undervisningen. Hawthorne-effekten, også kjent som kontrolleffekten, er når de som observeres endrer adferden av den grunn at de blir observert. Om gruppedeltakerne ikke hadde vært tilstede i undervisningen, kan det hende engasjementet blant elevene ville utspilt seg annerledes (Halle, 2014).

3 Os barneskole

3.1 Introduksjon, Os

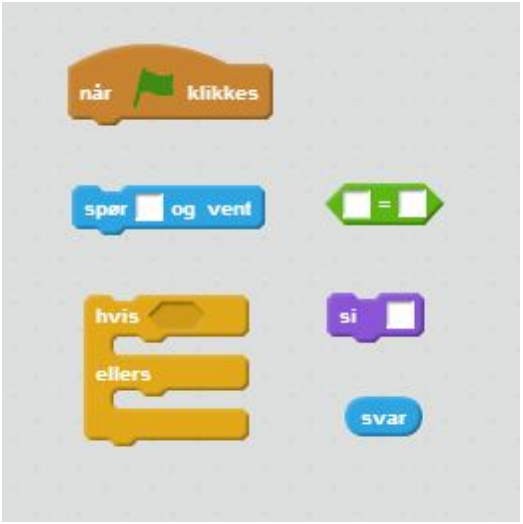
I dette kapittelet gjennomgås uttestingen ved Os skole. Det vil belyses hvordan uttestingen av verktøy foregikk, samt hvilke metoder som har blitt anvendt for å sikre relevante data på bakgrunn av øktene. Videre vil resultatene fra observasjonene fremlegges og danne grunnlag for diskusjon og konklusjon.

Tema for undervisningen ved Os var omgjøring av måleenheter – spesielt med tanke på konvertering fra én enhet til en annen.

3.2 Utviklingsmiljøet Scratch

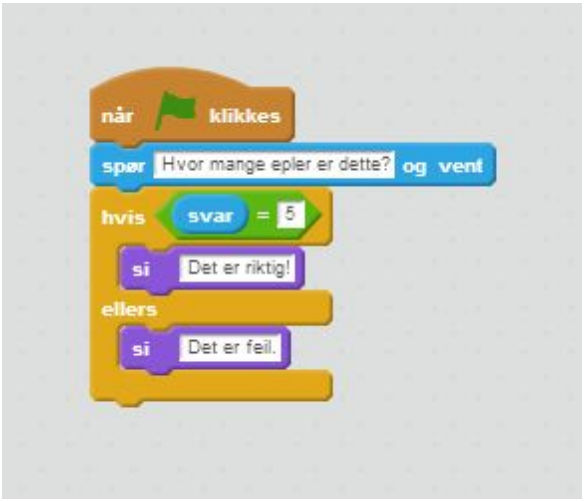
Gjennom uttestingen ved Os ble utviklingsmiljøet Scratch brukt for å lære barn å programmere. Scratch er et grafisk programmeringsspråk hvor syntaksen består av å hekte sammen byggeklosser med forskjellig funksjonalitet. Figur x viser en rekke

forskjellige klosser som kan brukes til å bygge programmer i Scratch. Disse klossene inkluderer blant annet en måte å kjøre programmet, en sammenligning av to verdier, og en variabel som holder på spesifikk input fra brukeren av programmet.



Figur x: Diverse klosser som kan brukes i Scratch

Figur x nedenfor illustrerer hvordan disse klossene kan settes sammen for å bygge et hensiktsfult program. Klossene eksekveres i rekkefølgen de er hektet.



Figur x: Eksempel på kildekode til et program i Scratch

I dette eksempel-programmet ville brukeren blitt spurt “Hvor mange epler er dette?”, med mulighet om å skrive inn svar. Dersom svaret som brukeren skriver inn er 5, vises beskjeden “Det er riktig!” – i alle andre tilfeller vises beskjeden “Det er feil”. Merk hvordan svar-klossen nestes inni sammenligningen, som igjen nestes inni betingelsen til hvis/ellers-klossen.

3.3 Metode, Os

3.3.1 Forberedelsesseminar, Os

I forkant av uttestingene ble det holdt to forberedende seminarer sammen med lærerne fra de deltagende skolene. Ett seminar tok for seg uttestingen ved Os barneskole, og det andre seminaret tok for seg Strupe ungdomsskole. Én av hensiktene bak seminarene var å teste og kvalitetssikre undervisningsmaterialet som var planlagt å brukes under uttestingen ved skolene. Lærerne ble derfor satt til å løse oppgaver, og komme med tilbakemelding på oppgavenes kvalitet og kompleksitetsnivå.

I forberedelsesseminaret for Os gikk barneskolelærer Aasmund Lunde gjennom en rekke oppgaver som var foreslått for bruk i undervisningen. Oppgavene omhandlet omgjøring av måleenheter, som var tema for undervisningsopplegget ved Os. Aasmund hadde allerede en del erfaring med Scratch fra før, og kom med feedback på oppgavene.

3.3.2 Produksjon av undervisningsmateriale

Bachelorgruppen produserte undervisningsmateriale for bruk i uttestingen ved Os skole. Undervisningsmaterialet som ble brukt i uttestingen ved Strupe ungdomsskole, ble produsert av Børre Stenseth. Basert på erfaringene ved Strupe, videreutviklet bachelorgruppa oppgavesettet til Børre Stenseth for bruk ved Halden videregående skole. Alt av undervisningsmateriale ble sett gjennom og godkjent av Odd Tore Kaufmann.

Gjennom uttestingskapitlene vil det hyppig referes til oppgaver som ble utført i undervisningen. Disse oppgavene er listet i vedlegg 1, *Oppgaveindeks*, vedlegg 2, *Oppgavesett, Os*, og vedlegg 3, *Oppgavesett, Strupe*.

3.3.3 Valg av oppgaver

Å komme frem til nøyaktig hvilke oppgaver som skulle brukes i uttestingen var en tidkrevende prosess som inkluderte prototyping, uttesting og ekstern feedback. Videre

diskusjon rundt utvikling av undervisningsmateriale finnes i kapittel 3.6, *Filosofi rundt Scratch-oppgaver*.

3.3.4 Uttesting

Uttestingen ved Os skole foregikk som en kvalitativ feltobservasjon av deltakere i en undervisning-setting. Deltakere var elever fra to klasser i syvende trinn ved Os skole, og én matematikklærer for dette trinnet.

Hensikten med denne uttestingen var å fungere som en pilottest for bruk av programmering som pedagogisk verktøy i matematikkundervisningen på barneskolenivå. Uttestingen var ment å danne grunnlag for passende vanskelighetsgrad og kompleksitet på programmeringsdelen, å undersøke utviklingsmiljøet Scratch sin brukbarhet som pedagogisk verktøy i barneskole-matematikk, og å undersøke barnas evne til å lære matematikk gjennom bruk av programmering. Videre var uttestingen ment å gi inntrykk av hva slags undervisningsmateriale som er egnet for et slikt undervisningsopplegg.

Uttestingen ble utført som fire 90 minutter lange undervisningsøkter i matematikkundervisningen for syvende trinn ved Os skole. Undervisningen ble hovedsakelig holdt av matematikklærer Aasmund Lunde ved Os skole, og Odd Tore Kaufmann, matematikkdidaktiker og førsteamanuensis ved Høgskolen i Østfold. Bachelorgruppen fungerte som assistenter i klasserommet, med ansvar om å svare på spørsmål, hjelpe elever med oppgaver, og stod for enkelte deler av undervisningen.

I hvert klasserom ble det brukt prosjektor for å vise skjermbilde til maskinen til den som stod for undervisningen i plenum. Demonstrasjon på storskjerm ble hyppig brukt i hver undervisningsøkt. Bruksområder inkluderer blant annet introduksjon av Scratch, utføring av demo-oppgaver, og live gjennomgang av oppgaveløsninger, med innspill fra elever.

Hver elev hadde tilgang til sin egen datamaskin, som ble brukt til å løse oppgaver i Scratch. Noen elever jobbet to-og-to på én datamaskin.

For en mer detaljert gjennomgang av hvordan hver økt ble utført, se vedlegg 4, *Gjennomføringsbeskrivelse, Os*. Resultatene referer ofte til informasjon i gjennomføringsbeskrivelsen.

3.4 Observasjoner, Os

Observasjoner i dette kapittelet er dokumentert i relativt høy detalj. Fordi resultatene er hentet gjennom upresise og subkejtive metoder, baseres datagrunnlaget på inntrykk og erfaringer. For å formidle disse erfaringene på en betydningsfull måte, må de beskrives i tilstrekkelig detalj. En annen grunn til å beholde et høyt detaljnivå, er for å sikre at bachelorprosjektet kan levere mest mulig nyttig informasjon til ProgPed. Én av hensiktene bak bachelorprosjektet er tross alt muliggjøringen av fremtidige prosjekter.

3.4.1 Økt én

Økt én begynte med en felles introduksjon som læreren Aasmund Lunde holdt for begge klassene. Han gikk gjennom en rekke PowerPoint-slides, som viste kildekoden til noen enkle programmer, og forklarte hvordan programmet kjørte. Han vinklet dette som at klassen skulle se hva som skjer inni en kalkulator. Han gikk gjennom de fleste typer klosser som er tilgjengelig i Scratch, men hoppet midlertidig over “data”-klossene.

Like etter introduksjonen, som kun viste statiske bilder av ferdige programmer, åpnet Aasmund Lunde et uferdig program i Scratch, som skulle diskuteres i klassen. Da han kjørte programmet, skjedde det ingen ting. Aasmund Lunde hadde hektet fra hverandre programmet inn i fire deler. Basert på det elevene hadde sett i introduksjonen, skulle de komme med forslag til hva som måtte gjøres for å få programmet til å fungere.

Flere elever skjønte fort at man trengte et grønt flagg for å starte programmet, og foreslo å hekte det grønne flagget i begynnelsen av programmet. Det å hekte sammen klosser etter hverandre var intuitivt for elevene. Videre anerkjente flere elever at man måtte hekte på klossene som gjorde utregningen, og klossene som tok imot input. Aasmund Lunde stilte spørsmål underveis om hva hver del gjorde, og det var forståelig for elevene å lagre inputen i “svar”-klossen, til tross for at klassene ikke hadde vært innom variabler ennå. De forsto at det som ble lagret i “svar”-klossen var det som brukeren hadde skrevet inn som svar til “spør”-klossen.

Etter Aasmund Lunde sin introduksjon gikk undervisningen over til live demonstrasjon av demo 1, 2, 3 og 4 (se vedlegg 1, *Oppgaveindeks*). Hver av disse demo-programmene hadde en gitt hensikt, men var uferdige. Alle klossene som var nødvendige for å fullføre programmet lå løst i skriptet. Hvert program ble fullført på storskjerm, mens hvert steg ble forklart og diskutert med klassene.

Da løsning for demo 1 ble diskutert var det mange hender i været. Det å gjøre småendringer i programmet for å endre funksjonaliteten var intuitivt for elevene. I denne demoen viste flere elever forståelse for å bruke “svar”-klossen i en sammenligning, det å bruke sammenligningen som betingelse i “hvis/ellers”-klossen, og hvordan “hvis/ellers”-klossen fungerer. Denne forståelsen ble bekreftet i senere øvelser, der elevene måtte sette sammen et ekvivalent program på egenhånd.

Da demo 2 ble gjennomgått på storskjerm, uttalte flere av elevene at de allerede hadde satt sammen løsningen. Disse ble sjekket av medlemmer av bachelorgruppa, og programmene var løst riktig. Før demo-programmet var ferdig satt sammen på storskjermen, ble én av de elevene som hevdet at de allerede hadde løst den bedt om å løse den på stormskjermen foran klassen. Dette klarte eleven feilfritt. Det var imidlertid noen elever som hadde forsøkt å hardkode utregningen, ved å skrive inn verdien “12” i utreningsklossen, i stedet for å benytte “svar”-klossen for den éne delen av utregninga.

Innholdet i demo 3 og 4 var intuitivt for de fleste, uten nevneverdige problemer eller utmerkelser.

Etter gjennomgang av demo-programmene, ble elevene satt til å gjøre øvingsoppgaver på egenhånd. Øving 1 (Hesten Harald - konvertere mellom enheter for vekt) var intuitiv for elevene. Én av elevene demonstrerte i plenum hvordan man skulle løse den. Dette var en elev som i følge én av lærerne vanligvis var svak i matte, men som gjorde det bra i Scratch.

På øving 2 (konvertere mellom ulike enheter for lengde, temperatur, volum og vekt) var det flere elever som hadde problemer med nesting av klosser. Først og fremst var bruk av join-klossen lite intuitivt for dem. En annen vanskelighet med nestingen var selve utføringen av å plassere klossene riktig. Scratch kan være noe finurlig når det kommer til hvordan klossene plasseres inni andre klosser – spesielt når de nestes i klosser som har flere tomme plasser. Så det var flere ganger elever visste løsningen, men allikevel hadde problemer med å plassere klossen med riktig nesting.

Da elevene fikk utfordringen om å reversere funksjonaliteten til øving 2, slik at den gjorde om til meter fra centimeter, i stedet for centimeter til meter, klarte omtrent halvparten av klassen dette. De elevene som klarte utfordringen anerkjente at operatoren for dividering måtte byttes ut med multiplikasjon-klossen.

Øving 3 (konvertere mellom enheter, men med andre måleenheter) gikk fint for alle gruppene. Elevene viste at de mestret bruk av hvis/ellers-klossen, det å bruke en betingelse i en hvis/ellers-kloss, og det å bruke en sammenlignende operator som betingelsen. Dette var konsepter de hadde lært i demo 3, og det var i denne øvingen de demonstrerte at de forsto hvordan man brukte dem på egen hånd.

Øving 4 (konvertere lengdeenheter - regne hvor langt personen gikk til sammen) ble hoppet over, grunnet manglende oppgavetekst, samt innhold som ikke var direkte ekvivalent med et demo-program.

3.4.2 Økt to

A-klassen: Da økt to begynte med repetisjon av øvingsprogrammene som ble gjort i økt én, var det overraskende få hender i været. Klassen virket langt mer beskjeden og forsiktig enn dagen før. Etter at et par demo-programmer var blitt gått gjennom, og et par konsepter forklart på nytt, virket klassen mer selvsikker, og flere hender dukket opp igjen. Med innspill fra et mangfold av elever ble øvingsprogrammene løst feilfritt. Klassen ble deretter satt til å jobbe med oppgavesettet.

De fleste gruppene i A-klassen klarte alle deloppgavene i oppgavene 1 (Hesten Harald - konvertere mellom enheter for vekt) uten nevneverdige problemer. De klarte å identifisere hvilken operator de måtte endre til i oppgave d (Hesten Harald - gjør endringer slik at programmet regner om fra gram til kg). Noen grupper visste med det samme at de måtte dele på 1000, mens andre grupper eksperimenterte seg fram ved å teste 10, 100 og 1000 og se på resultatet for å se hva som gav riktig omgjøring. Flere studenter svarte at man måtte bytte til deling, fordi kilogram er større enn gram.

Én elev klarte hele oppgave 3 (konvertere mellom enheter, men med andre måleenheter) mens resten av klassen holdte på med oppgave 2 (program som konverterer mellom ulike enheter for lengde, temperatur, volum og vekt). Hun synes oppgavene var litt for enkle, og fikk prøve seg videre i oppgavesettet i forkant av de andre elevene. Hun satt seg fast i oppgave 4 (konvertere lengdeenheter - regne hvor langt personen gikk til sammen) på grunn av vanskeligheter med nesting av klosser, så gruppen gjorde oppgaven enklere ved å redusere mengden nesting som var nødvendig for å løse den. Tanken var at dersom den eleven som gjorde det best plutselig satt bom fast, ville oppgaven sannsynligvis være for vanskelig for resten av klassen også.

I mesteparten av økta var de fleste av elevene i A-klassen ivrige og entusiastiske.

B-klassen: Timen begynte med oppsummering fra gårsdagen. Læreren hentet frem demo-oppgave nummer 3 på smart-boardet, som ble løst i økten dagen før, og spurte ut elevene om oppbygningen av kodeblokkene i oppgaven. Elevene svarte riktig på spørsmålene om kodeblokkenes funksjoner.

Etter oppsummeringen tok læreren opp variabelkonseptet. Dette ble tatt litt på sparket, ettersom oppgavene var produsert kvelden i forveien. Han brukte her et eksempel hvor variabelen *meter* var satt til å inneholde *svar* delt på 100. Dette for å gjøre om svaret i centimeter til meter. Elevene gav utrykk for at variabelkonseptet virker noe vanskeligere enn gårsdagens oppgaver. Læreren velger etter hvert å skjule variablene på sketchen, og heller fokusere på programkoden. Det så ut til at elevene knotet en del, men de fikk det til tilslutt.

Læreren gikk videre til en oppgave hvor volum var i fokus. Programmet inkluderte et rektangel med teksten volum. Når programkoden kjørte skulle man kunne klikke på rektangelet som etterspør mengden centiliter. Rektangelet svarte deretter med svaret omgjort til liter, deciliter og mililiter.

Videre fikk elevene i oppgave å lage en kopi av denne figuren. Den skulle plasseres ved siden av den andre og gjøres om til å behandle lengde.

Elevene endret navnet til lengde og kodeklossene til å gjelde måleenheter for lengde. Elevene som virket til å ha god kjennskap til Scratch, prøvde ut variabeloppgaven før Aasmund hadde gjennomgått hele oppgaveteksten.

Mot slutten av økten gav flere elever litt opp og vandret litt rundt og var generelt urolige. Det var dog enkelte elever som fortsatt arbeidet konsentrert, enten alene eller sammen med andre. Mot slutten av timen var det mye høylydt prating og uroligheter. Aasmund rundet av timen like etterpå.

Etter spørsmål fra lærer svarte elevene at dagens oppgaver var mer komplisert enn gårsdagens oppgaver. De sa at de lærte mye denne timen.

3.4.3 Økt tre

Økt tre begynte med tekniske problemer. Det var umulig å få tilgang til Scratch, fordi Scratch-serverne til MIT hadde nedetid. Serverne var nede fra 11:45 til 12:48. Ingen

lokalløsninger var egnet, for hver PC krever Adobe Air for å kjøre lokalt. Etter utprøving og feiling av en plan B, som inkluderer installasjon av lokalversjonen av Scratch, ble Scratch-økta avlyst, og elevene jobbet med det ordinære matematikkpensumet i stedet. Økt tre ble heller utsatt til datoen som var ment å være økt fire, og økt fire ble flyttet til etter vinterferien. Erfaringer og forslag til forbedret plan B diskuteres videre i diskusjonskapittelet underliggende Os-undersøkelsen.

A-klassen: Dagen etter kunne økt tre utføres uten tekniske problemer. Økta begynte med kort repetisjon av oppgavene som ble løst i forrige økt. Nok en gang var elevene noe beskjedne i starten, til tross for at de fleste hadde løst oppgavene forrige uke. Etter den første oppgaven ble besvart av en elev, ble resten av klassen mer aktiv, og fikk det meste riktig.

Etter oppfriskningen satte klassen i gang med oppgave 3 (konvertere mellom enheter, men med andre måleenheter) og 4 (konvertere lengdeenheter - regne hvor langt personen gikk til sammen). Oppgave 3 var enkel for de fleste elevene. Oppgave 4 gikk fint for noen elever, mens de fleste hadde vanskeligheter med den. De som hadde vanskeligheter hadde problemer med å forstå hvordan programmet fungerte i seg selv. Da de ble forklart at de forskjellige linjene skulle legge sammen enhetene som hver sin enhet, forstod de som regel hvordan utregningene skulle settes opp. De måtte bare vite hva linjene konkret var ment å gjøre. Programmet kan derfor sees på som lite intuitivt.

Da løsning på oppgave 3 (konvertere mellom enheter, men med andre måleenheter) ble gått gjennom i plenum, var det mange hender i været. Klassen bekreftet i kor at de forsto hver av omgjøringene. Omrent halvparten av klassen var ferdig med oppgave 4 (konvertere lengdeenheter - regne hvor langt personen gikk til sammen) i løpet av 09:40. I slutten av økta ble oppgave 4 gått gjennom i plenum. Fordi dette var en dobbelttime uten pause, var det tydelig at flere elever begynte å bli ukonsentrerte. Til tross for dette var hvert innspill fra elevene korrekte. Elevene kunne begrunne hvor hver variabel skulle regnes med hva, f.eks. at mil ganges med 10 000 for å få meter, fordi det er 10 000 meter i en mil. Det var tre par med elever som ikke forstod oppgave 4.

B-klassen: Da elevene jobbet med oppgave 1, var det flere som var forvirret over forskjellen mellom bruk av komma og punktum i desimalregning.

Oppgave 2 (program som konverterer mellom ulike enheter for lengde, temperatur, volum og vekt) virket litt diffus for elevene, da det ble spurt om svaret løsningene. Elevene blir spurt i å finne ut hva 0 fahrenheit er i celsius. Videre blir de spurt om hvordan gjøre om til gallon, km, celsius og hektogram.

Da de i fleste i klassen begynte på oppgave 3 (konvertere mellom enheter, men med andre måleenheter), var 2 av elevene allerede her på oppgave 6 (legge sammen X gram, Y kg og Z hg og konvertere til kg, hg og gram). Det virket som elevene var mer konsentrert enn i de forrige øktene. Dette kan ha noe med tidsrommet på dagen å gjøre. Oppgave 3 virket vanskeligere enn oppgave 2, men de fleste fikk den til til slutt.

Oppgave 4 (konvertere lengdeenheter - regne hvor langt personen gikk til sammen) virket noe knotete og lite intuitiv for elevene. Selv Aasmund hadde vanskeligheter med å fostå hvordan oppgaven var ment å løses. En elev fikk heller i oppgave å demonstrere en egenprodusert sketch hvor han brukte piltaster for å styre figuren. Avslutningsvis hadde ingen gjort hjemmeleksa som Aasmund spurte om forrige time, for de som hadde lyst.

3.4.4 Økt fire

A-klassen: Siste økta begynte med repetisjon av oppgave 3 og 4. Oppgave 3 (konvertere mellom enheter, men med andre måleenheter) var nå triviell for de fleste. Repetisjon av oppgave 4 (konvertere lengde - regne hvor langt personen gikk til sammen) hadde mange deltakere, med kun rette svar. Dette var dog fra de elevene som hadde mestret oppgaven i forrige økt. Det er vanskelig å si hvorvidt de som hadde problemer med den i forrige økt forstod den godt til slutt.

Oppgave 5 (konvertere volum av væske i en kjele) hadde mange likhetstrekk med oppgave 4, som førte til at den også var lite intuitiv for elevene. Dette var også en oppgave prosjektgruppen forenklet underveis. Nok en gang dukket nesting av operatorer opp som et problem.

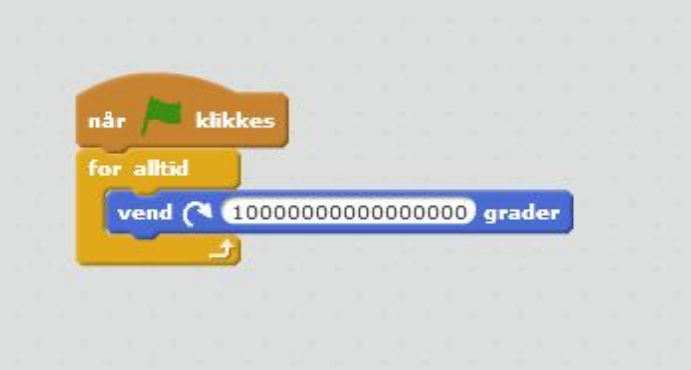
I den siste halvtimen av økta der elevene fikk velge mellom å utforske og endre på eksisterende programmer, eller å lage et eget program fra bunn av, valgte de fleste førstnevnte alternativ. To grupper på to elever valgte å forsøke å lage noe nytt på egenhånd.

Mange av elevene som valgte å endre på eksisterende programmer følte seg overveldet av kildekoden til programmene hadde valgt. Disse programmene var ofte langt større og kompliserte enn programmene som elevene hadde vært borti tidligere. Mange elever hadde valgt programmer som spilte av en animasjon. Det noen av elevene fikk til var å identifisere kildekoden som genererte tekstboblene i animasjonen,

og å redigere dem slik at animasjonen genererte annerledes tekst. Én elev som hadde eksperimentert med å endre på tilfeldige verdier i et program, klarte å endre spektrumet av farger som ble brukt i en abstrakt animasjon, men kunne ikke forklare hvorfor.

Én av gruppene som valgte å lage et eget program fra bunn av, lagde et program med følgende kildekode:

Figur x: Kildekoden til programmet som elevene laget på egen hånd



Figur x: Videre kode som elevene laget. Spinner objektet rundt.

Programmet finnes her: https://scratch.mit.edu/projects/103924962/. Det kjøres ved å trykke på det grønne flagget, og kildekoden kan sees ved å klikke “se inni” eller “see inside”, avhengig av hvilket språk som er valgt.

Delen av programmet som vises i figur x spør brukeren hvilket fotballag man liker. Dersom man svarer nøyaktig “Barcelona og Liverpool”, får man outputen “DET ER BRA FOTBAL LAG” *(sic)*. Dersom man svarer noe annet, får man outputen “DET ER DÅRLIG HAHAHA” *(sic)*, og figuren roteres. Den andre delen av programmet, som vises i figur x, gjør at ball-, sol- og ekorn-objektene snurrer kontinuerlig.

B-klassen: Økten begynte ganske direkte med introduksjon av oppgave 5 (konvertere volum av væske i en kjele). En del elever virket ganske blanke på spørsmål fra læreren om hva man må gjøre for å regne om fra millimeter til meter. Læreren tok derfor elevene over til vanlig tavleundervisning for å friske opp i omgjøring mellom volumenheter. Aasmund viste trinnvis med både deling og ganging. Det virket til at elevene som svarte på Aasmunds spørsmål, om de ulike trinnene, ikke helt husket hva man måtte gjøre, da flere elever tippet feil opptil flere ganger.

En elev fikk, på egenhånd, til oppgaven i mellomtiden. Han ble bedt om å løse oppgaven på smart-boardet foran de andre elevene. Etterpå hjalp læreren de som fortsatt stod litt fast. De som løste oppgave 5 raskt, har nå hoppet over til oppgave 6 (legge sammen X gram, Y kg og Z hg og konvertere til kg, hg og gram).

Videre lot læreren elevene jobbe med egne oppgaver. Han ba de tenke ut egne oppgaver eller hente inspirasjon fra tilgjengelige oppgaver på scratch.mit.edu. Flere var ivrige etter å lage oppgaver basert på interaksjon mellom figuren og piltastene. Det var fremdeles noen som jobbet videre på oppgave 6, for deretter å jobbe med de frie oppgavene.

Med de frie oppgavene var det større grad av prøving og feiling, enn i de gitte oppgavene ellers. Et par elever hentet frem et pong-basert spill fra http://scratch.mit.edu hvor de endret på programkoden for å øke hastigheten og dermed vanskelighetsgraden i spillet.

Før timen ble rundet av gikk læreren gjennom oppgaven gitt i hjemmelekse før vinterferien som regner ut gangetabellen. Elevene gav tilbakemelding på at prosjektet var spennende og læringsrikt. Det ble morsommere jo mer de fikk til. De sa også at prosjektet var gøy.

3.5 Diskusjon, Os

3.5.1 Introdusere elever til Scratch

Den korte introduksjonen som ble gjort i begynnelsen av økt én i PowerPoint virket lite effektiv. Det er flere aspekter ved å bruke Scratch som ikke kan kommuniseres gjennom å vise statiske bilder. Å introdusere Scratch gjennom PowerPoint-slides virker derfor lite hensiktsmessig, når man har det interaktive utviklingsmiljøet tilgjengelig.

Det var først da Aasmund Lunde åpnet Scratch, og viste hvordan klossene kunne plasseres, og hvordan de fungerte i praksis, at elevene begynte å forstå konseptet. Ved å ha en rekke demo-programmer klare, som hver tok for seg funksjonaliteten til forskjellige typer klosser, viste seg å være en god læringsmåte i begynnelsen. Det tillot en jevn progresjon i kompleksitet, og muligheten til å kontinuerlig bygge videre på det man hadde lært underveis. Videre gav det mulighet for hyppig tilbakemelding fra elevene, ved å spørre dem om innspill til hvordan man ville løse enkelte problemer.

3.5.2 Bruk av “svar”-klossen

Et interessant poeng som dukket opp under introduksjonen var hvor raskt elevene forstod bruken av “svar”-klossen. Det var en del skepsis i forkant av uttestingen angående hvordan variabel-begrepet skulle introduseres, eller om det skulle introduseres i det hele tatt. Det som var interessant med elevenes tidlige forståelse av “svar”-klossen er at denne klossen kan sees på som en type variabel. Forskjellen er at den er navngitt fra før, og er begrenset til én funksjon, å lagre en input fra brukeren. Det er allikevel en variabel: Et navngitt uttrykk som representerer en lagret verdi.

Forståelse for bruk av denne klossen ble demonstrert flere ganger: Både gjennom innspill under gjennomgang av demo-programmer, samt gjennom individuell løsing av oppgpavesettet. Omtrent halvparten av klassen viste forståelse for konseptet i løpet av føste økt, mens resten forstod det underveis. Noen av eleven forstod “svar”-klossens rolle i eksisterende programmer, men hadde vanskeligheter med å forstå hvor “svar”-klossen skulle plasseres i uferdige programmer.

3.5.3 Problematikk med nesting av klosser/operatorer

Et problem som dukket opp relativt tidlig var nesting av klosser – ofte operatorer. Problemet oppstod først under individuell løsing av øving 2. I øvingen fikk elevene tildelt

de klossene de trengte for å lage et program som gjør om fra centimeter til meter, som vist i figur x. Oppgaven løses ved å sette sammen klossene som vist i figur x.

Figur x: Den oppgitte kildekoden for øving 2

Figur x: En mulig fasit for løsing av øving 2

Mange elever hadde vanskeligheter med dette. Problematikken kan ha oppstått av to årsaker. Først og fremst krever det et høyere nivå av problemløsning. Antallet mulige kombinasjoner av klosser øker drastisk, i tillegg til at det krever at elevene ser på problemet som en helhet.

Den andre faktoren er at selve utføringen av nestingen kan være overraskende kinkig i Scratch-grensesnittet. For å neste en kloss inni et tomt felt på en annen kloss, må klossen som ønskes å nestes plasseres på et veldig spesifikt område – hvis man

bommer på dette området, risikerer man å flytte på de underliggende klossene, mens klossen man prøvde å plassere, ofte flyr til andre siden av skjermen. Hvis man har plassert klossen riktig for å neste den, vises en subtil indikator i feltet man prøver å plassere klossen i. Dette var noe elevene ikke var tilstrekkelig klar over i forkant av oppgaveløsingen. I fremtiden må det demonstreres bedre i introduksjonen hvordan nesting av klosser fungerer, for å forhindre unødvendig forvirring.

På grunn av problematikken med nesting av klosser, ble oppgavesettet justert underveis. Som nevnt i observasjoner-kapittelet fikk den eleven som hadde utmerket seg mest vesentlige problemer med oppgave 4 (konvertere lengdeenheter - regne hvor langt personen gikk til sammen). Dette var grunnet mengden med nesting av klosser som eleven var forventet å utføre selv. Hun uttrykket frustrasjon over at klossene ikke landet der hun ville, på grunn av feilplassering av klosser under nestingen.

Elevene hadde sjeldent problemer med å forstå funksjonaliteten til nestede klosser – problemet oppstod først da elevene skulle utføre nestingen selv. Som resultat ble vanskelighetsgraden på oppgave 4 (konvertere lengdeenheter - regne hvor langt personen gikk til sammen) og ut justert ned, ved å redusere mengden nesting som elevene måtte utføre selv. Det betyr ikke at elever på dette nivået er ute av stand til å utføre nesting av klosser, men det kreves nøyere introduksjon av tema, spesielt hvordan det i praksis utføres.

3.5.4 Vanskelighetsgrad av undervisningsmateriale

For å se hva de forskjellige oppgavene inneholder, se vedlegg 1, *Oppgaveindeks*.

Demo 1, 2, 3 og 4: Alle disse hadde passende vanskelighetsgrad. Da elevene ble spurt om forslag til løsning, var det mange hender i været, med mange gode svar. Elevene viste allerede her god forståelse for hvordan programmet kjøres, hekting av klosser, bruk av “svar”-klossen, samt bruk av “hvis/ellers”-klossen. De fleste grunnleggende konseptene ble gjennomgått i god rekkefølge, med unntak av nesting av klosser, som burde vært gjennomgått nærmere under introduksjonen.

Øving 1, 2 og 3: Det eneste problemet som oppstod i disse øvingene var nestingen i øving 2, som nevnt tidligere. Ellers var øvingene godt egnet, og intuitive for elevene. Alle gruppene fikk til det meste på disse. Øving 4 hadde utilstrekkelig oppgavetekst, og ble hoppet over.

Oppgave 1, 2 og 3: De aller fleste fikk til oppgave 1 (Hesten Harald - konvertere mellom enheter for vekt) i oppgavesettet uten problemer. Selv om noen klarte det på første forsøk, var det noen grupper som eksperimenterte seg fram til løsningen, ved å teste ulike utregninger. Dette kan sees på som én av fordelene ved programmering. Elevene har mulighet til å få kontinuerlig feedback på det de gjør. Ved å observere resultatet av flere forskjellige input-verdier, lar det elevene se en overordnet sammenheng mellom verdiene i utregningen, utover det som er mulig å uttrykke gjennom papir. Oppgave 2 (konvertere mellom ulike enheter for lengde, temperatur, volum og vekt) og 3 (konvertere mellom enheter, men med andre måleenheter) var begge av passende vanskelighetsgrad, og nærmest alle løste dem på en god måte.

Oppgave 4, 5 og 6: Disse var alle litt for vanskelige. Oppgave 4 (konvertere lengdeenheter - regne hvor langt personen gikk til sammen) var i begynnelsen for vanskelig grunnet mengden nesting som elevene var forventet å utføre selv. Dette ble justert slik at elevene ikke lenger måtte gjøre nestingen selv, og oppgaven ble i etterkant svært passende. Oppgave 5 (konvertere volum av væske i en kjele) og 6 (legge sammen X gram, Y kg og Z hg og konvertere til kg, hg og gram) var begge lite intuitive. Her var selve oppgavenes struktur problemet – måten å løse dem på var såpass annerledes fra hvordan man vanligvis løser omgjøringsoppgaver, at det ikke var programmeringen som var vanskelig, men å forstå hvor oppgavene ville hen. Disse oppgavene ville vært uegnet for alle nivåer, ettersom at de var forvirrende, selv for folk med kompetanse innen programmering og matematikk. De få elevene som klarte disse oppgavene fikk fikk mye hjelp fra studentgruppa – oppgaver med såpass fremmed struktur bør unngåes.

3.5.5 Program laget fra bunn av elevgruppe

I forkant av uttestingen ble det definert tre forskjellige nivåer bachelorgruppen ønsket å utforske:

- Nivå én – forståelse: Evnen til å lese og forstå kildekoden til et program.
- Nivå to – videreutvikling: Evnen til å gjøre endringer i et eksisterende program, for å oppnå et gitt mål.
- Nivå tre – selvstendig programmering: Evnen til å skape et helhetlig program fra bunnen av.

Det var skepsis innad i prosjektgruppa om hvorvidt det var mulig å komme innom nivå tre i løpet av såpass kort tid, og på et såpass ungt klassetrinn. Det er en betydelig forskjell i forståelsen som er nødvendig for å lage noe helt fra bunnen av, kontra det å

gjøre endringer i noe som allerede eksisterer. Det krever at eleven er i stand til å planlegge en helhet bestående av flere deler, som skal passe sammen for å løse noe. Det krever også at eleven mestrer hver kloss som er nødvendig for å realisere denne planen. Allikevel var det aktuelt å se hvorvidt det var mulig å oppnå det tredje nivået, så den siste halvtimen i den siste økta ble dedikert til dette. To grupper på to elever hver klarte å lage sine egne programmer, uten hjelp fra studentene. Ett av disse programmene er illustrert i figur x og x, i resutater-kapittelet.

Dette programmet ble laget av en gruppe bestående av to elever. Begge elevene var blant de mer ivrige gjennom uttestingsperioden. Hoveddelen av programmet illustrerer at elevene mestrer en rekke konsepter innen både Scratch og programmering. Ikke bare forstod de hvordan de skulle bruke inputen som var lagret i “svar”-klossen, men de forstod hvordan den skulle sammenlignes med en tekststreng for å utgjøre en betingelse i “hvis/ellers”-klossen. De demonstrerte også en evne til å utforske egne klosser selv, ettersom hverken rotasjon, eller endring av drakter (et objekts utseende) var gått gjennom i undervisningen. De viste også forståelse for hvordan enkelte kodesnutter kunne tilhøre egne objekter, ved å legge forskjellig kode i forskjellige objekter.

3.5.6 Praktiske problemer

Som nevnt i observasjoner-kapittelet, ble undervisningen i økt 3 avbrutt grunnet nedetid hos Scratch-serverene til MIT. Serverene var oppe like før undervisningen, men gikk ned i det undervisningen begynte, og var nede helt frem til rett før undervisningen var slutt. Uten kommunikasjon til MIT har man ikke tilgang til online-versjonen av Scratch. Det betyr at man hverken kan bruke den nettbaserte Scratch-editoren, eller åpne eksisterende programmer som er lagret online. Dette vil også være tilfelle dersom klasserommet mangler tilgang til internett. Elevene jobbet i stedet med det ordinære matematikkpensumet i denne økta.

Bachelor-gruppa hadde anerkjent at en eventuell server-nedetid var en risiko, og hadde lagt en plan i tilfelle nedetid skulle inntreffe. Det var forberedt lokale installasjonsfiler for den lokale versjonen av Scratch på minnepinne. Planen var å installere lokalversjonen av Scratch på PC-en tilkoblet prosjektor, slik at man i det minste kunne demonstrere programmeringen på storskjerm, og heller utføre oppgaver i plenum.

Det gruppa ikke hadde forutsett var at selv den lokale installasjonsfila trengte å kommunisere med MIT, så programmet kunne ikke bli installert. Det vil si at dersom

lokalversjonen av programmet skal brukes som backup-plan i tilfelle MIT sin servere ikke er online, må lokalversjonen installeres i forveien.

Dette støter på nok et problem: Lokalversjonen av Scratch krever at Adobe Air er installert. Dette vil være gjørbart dersom plan B innebærer at kun PC-en som kobles til prosjektoren trenger Scratch, og at all programmering gjøres i plenum. Dette vil derimot være en vesentlig byrde dersom det forutsettes at hver elev skal kunne kjøre lokalversjonen på sin egen PC.

Byrden minskes dersom skolens IT-avdeling har mulighet til å fjerninstallere programvaren på hver PC i forveien. Dersom elevene bruker PC-er utdelt fra et tredje parti (slik som i Strupe-undersøkelsen) er byrden minimal: Både Adobe Air og Scratch vil kunne installeres i forkant uten problem, og hver elev vil kunne bruke offline-versjonen av Scratch på sin maskin.

Det betyr at dersom man ikke har mulighet til å installere programvare på elevenes maskiner i forveien, vil en akseptabel plan B være å installere lokalversjonen av Scratch og Adobe Air på maskinene til de ansvarlige for undervisningen. Disse maskinene brukes dermed til demonstrasjon på storskjerm, og diskusjonsbasert oppgaveløsning i plenum. Dersom skolen sin IT-avdeling har mulighet til å distribuere programvare til hver av elevene sine maskiner, eller dersom en tredje part som låner ut PC-er fritt kan forhåndsinstallere programvare, vil installasjon av Adobe Air og lokal Scratch være den beste løsningen. Da får hver elev mulighet til å programmere i Scratch, uten nødvendig tilgang til MIT, eller uten generell nettilgang.

3.6 Filosofi rundt Scratch-oppgaver

3.6.1 Innledning

Å produsere oppgaver for bruk i et undervisningsopplegg er ikke trivielt. I dette kapittelet drøftes noen av aspektene rundt det å produsere hensiktsmessige oppgaver.

Den viktigste tanken som måtte holdes i bakhodet under produksjonen av oppgavene, var at undervisningsopplegget først og fremst var ment å lære barn matematikk. Sluttmålet var ikke å lære programmering, men å bruke programmering som et pedagogisk verktøy for å forbedre matematikkforståelsen. Dette måtte følgelig reflekteres i oppgavesettet.

3.6.2 Kritier for oppgaver

For å sikre at de få timene som var til rådighet ble utnyttet best mulig, var oppgavene nødt til å oppfylle en rekke kriterier. Først og fremst måtte de bidra til økt matematikkforståelse.

Et annet viktig punkt er at de måtte dra nytte av fordelene som programmering bringer. Én av disse fordelene er hvordan programmering lar barn eksperimentere med input-verdier, som gir umiddelbar feedback om hvilke resultater/konsekvenser endringene forårsakte. Dette kan for eksempel være nyttig for å dynamisk utforske sammenhengen mellom forskjellige ledd i en utregning. Innen tema for undervisningen ved Os, kan det hjelpe barn med å se hvilket forhold diverse måleenheter har til hverandre.

En annen fordel er hvordan programmering lar barn se “bak kulissene” til utregninger. I stedet for å vite at en utregning skjer, kan de i stedet observere hvordan den utføres. Det finnes en rekke flere fordeler som bruk av programmering kan bidra med.

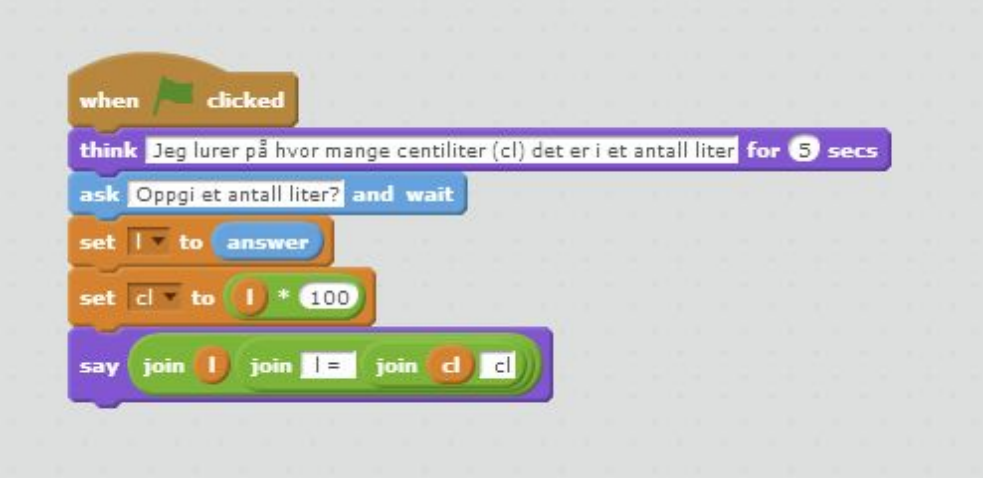
Poenget er at dersom disse fordelene ikke tas i bruk, kan ikke innføringen av programmering rettferdiggjøres. Å bruke programmering som pedagogisk verktøy er en investering. Det kreves tid og kunnskap. Utbyttet må kunne utveie investeringen. Derfor må oppgavene være på et nivå som reflekterer denne tankegangen.

3.6.3 Uegnede oppgaver

Kriteriene som prosjektgruppa satte til oppgavene, førte til at majoriteten av oppgave-prototypene ble forkastet. Det ble laget mange førsteutkast til oppgaver tidlig i prosessen, men gruppa anerkjente straks at mange av oppgavene kunne kategoriseres i forskjellige typer av oppgaver som burde unngås. De to største arketypene som gruppa lærte å unngå ble internt kalt “kalkulator-oppgaver” og “tekstbok-oppgaver”.

3.6.3.1 Kalkulator-oppgaver

Flere av de første prototypene som ble laget var oppgaver der elevene åpnet et program som konverterte fra én måleenhet til en annen, basert på bruker-input. Eleven kunne dermed skrive inn verdien som skulle gjøres om, og programmet ville skrive ut den omgjorte verdien. Kildekoden til et slikt program var på formen som vist i figur x:



Figur x: Kildekoden til et program brukt i en forkastet oppgave

En slik oppgave er lite hensiktsmessig i seg selv. Alt eleven får vite er hva diverse verdier tilsvarer i en annen måleenhet, noe som like så godt kunne vært gjort gjennom en ordinær kalkulator.

Et viktig poeng er at dersom eleven ikke har insentiv til å se inni kildekoden til programmet, gjør oppgaven noe feil. Programmer av typen som vist i figur x kan ha en nytteverdi, men her er et veldig viktig verktøy god bruk av oppgavetekst. Et eksempel på dette er oppgave 1 i oppgavesettet som ble brukt ved Os. Programmet med Hesten Harald faller under kalkulator-kategorien, men her brukes oppgaveteksten for å gjøre øvelsen hensiktsmessig.

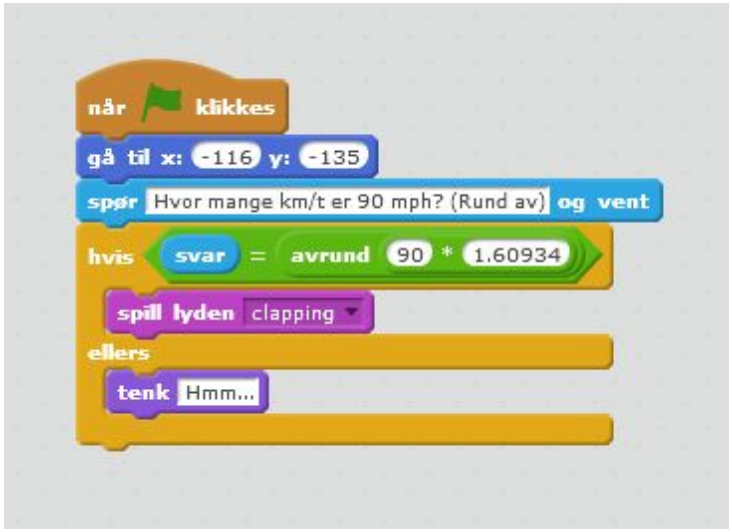
Først og fremt blir elevene oppfordret til å se i programkoden, og å analysere/drøfte. Deretter inkluderer oppgavene å gjøre endringer i programkoden. Hver oppgave øker gradvis i kompleksitet – fra å endre en multiplikator, som krever å endre en verdi i en eksisterende kloss, til å endre en operator, som krever å bytte ut selve klossen.

3.6.3.2 Tekstbok-oppgaver

En annen kategori som bør unngås er oppgaver som er funksjonelt like dersom de hadde stått i en tekstbok. Et eksempel på en slik oppgave er prototypen illustrert i figur x, med kildekoden vist i figur x. Den første figuren viser outputen til programmet, som for praksis skyld også inneholder oppgaveteksten. Når programmet blir kjørt, blir brukeren bedt om å skrive inn svaret på problemet i oppgaveteksten. Dersom svaret er riktig, spilles av en animasjon hvor bilen kjører videre, samt avspilling av en lydfil.

Figur x: Program-output og oppgavetekst til en forkastet oppgave

Denne oppgaven feiler i å rettferdiggjøre bruken av programmering av flere grunner. Først og fremst er dette en oppgave som kunne stått i en tekstbok. De eneste forskjellene ville vært mangel på animasjon, samt automatisk fasit. Disse forskjellene er imidlertid ikke grunn nok til å investere tid og ressurser på å innføre programmering i matematikkundervisningen.



Figur x: Kildekoden til et program brukt i en forkastet oppgave

Elevene ville fortsatt være nødt til å utføre konverteringen manuelt, som om de løste oppgaven fra en mattebok. Igjen dukker problemet opp om at elevene ikke har insentiv til å se inni, eller gjøre endringer i kildekoden til programmet. Her utnyttes ingen av fordelene som programmering bringer – følgelig vil ikke en slik oppgave rettferdiggjøre investeringen.

3.7 Konklusjon, Os

Introduksjon av Scratch for barneskoleelever bør gjøres gjennom live demonstrasjon av programmering i Scratch – Statisk PowerPoint bør unngås. Eventuelle undervisningsopplegg bør følge en progresjon gjennom tre nivåer: Først utvikle evnen til å lese og forstå kildekode, deretter evnen til å gjøre endringer i eksisterende kode for å oppnå et gitt mål, og til slutt evnen til å programmere noe eget fra bunn av. Det tredje nivået er oppnåelig for barneskoleelever ved 7. trinn etter fire to-timers økter, men ikke alle vil komme dit. Nesting av klosser/operatorer må introduseres nærmere enn hva som ble gjort i denne uttestingen.

Majoriteten av barn ved 7. trinn kan forventes å forstå kildekoden bak simple, eksisterende Scratch-programmer, og å kunne gjøre endringer i disse programmene for å oppnå gitte mål. Kun et fåtall av elever vil kunne forventes å lage egne programmer fra bunn av etter et undervisningsopplegg på åtte skoletimer. Dette er imidlertid noe som kan endres for fremtidige undervisningsopplegg av større omfang.

Hvorvidt bruk av programmering som pedagogisk verktøy kan være med på å fremme matematikkforståelsen hos barn vil være umulig å konkludere på grunnlag av dette prosjektet alene.

Scratch som verktøy har vist seg å være et passende utviklingsmiljø til å introdusere barneskoleelever for programmering. Ved å ha et intuitivt grensesnitt med mye grafisk feedback gjøres programmeringen svært tilgjengelig for barn. Begge klassene uttrykte at Scratch var gøy å jobbe med, noe som kan forbedre motivasjon i matematikkundervisningen. Noen barn som vanligvis var svakere i matte, gjorde det bra i Scratch, som kan tyde på Scratch kan være med på å inkludere barn som har vanskeligheter med ordinær matematikkundervisning.

Online-versjonen av Scratch har egenskaper som gjør den foretrukket over lokalversjonen. Det er allikevel en risiko for at MIT sine servere har nedetid, som gjør det nødvendig med en plan B. Dersom det er mulighet for å distribuere programvare for

alle elevenes maskiner, vil den sikreste løsninger være å forhåndsinstallere lokalversjonen av Scratch og Adobe Air på hver maskin, samt alt undervisningsmateriale.

4 Strupe ungdomsskole

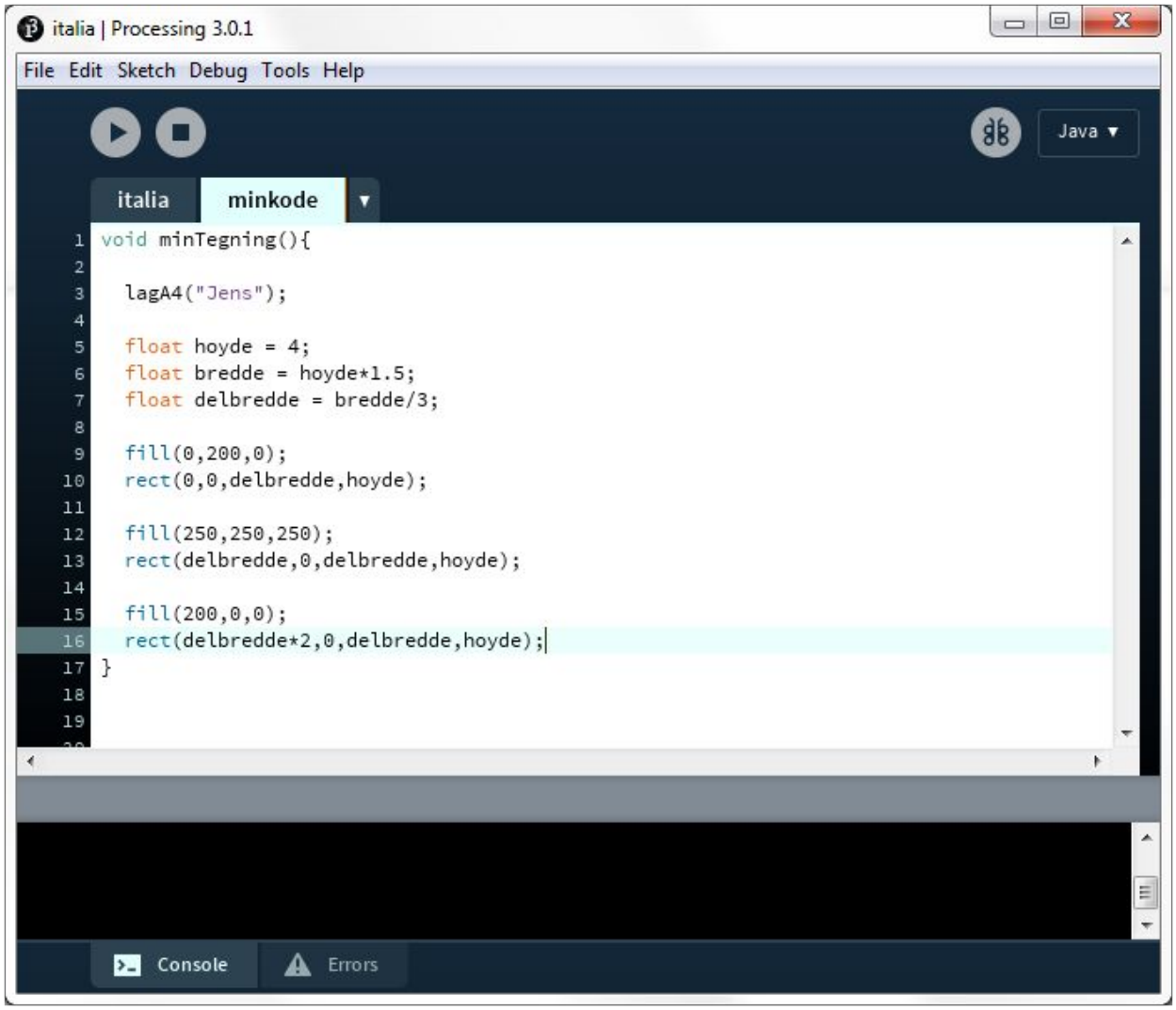
4.1 Introduksjon

I dette kapittelet gjennomgås uttestingen ved Strupe ungdomsskole. Det vil belyses hvordan uttestingen av verktøy foregikk, samt hvilke metoder som har blitt anvendt for å sikre relevante data på bakgrunn av øktene. Videre vil resultatene fra observasjonene fremlegges og danne grunnlag for diskusjon og konklusjon. I dette kapittelet henvises det til ulike oppgavesett. Med mindre annet er oppgitt hvenviser oppgavene under til det ordinære oppgavesettet, produsert av Børre Stenseth (se vedlegg 3.1, *Ordinært oppgavesett*).

4.2 Utviklingsmiljøet Processing

Gjennom uttestingen ved Strupe ble utviklingsmiljøet Processing brukt for å lære ungdomsskoleelever å programmere. Processing benytter et språk som er en dialekt av Java, men med forenklet syntaks. Ett av hovedmålene bak Processing er å introdusere nybegynnere til programmering, gjennom fokus på visuell feedback.

Figur x nedenfor viser grensesnittet til Processing, med eksempelkode fra én av oppgavene brukt i undervisningen ved Strupe.



Figur x: Grensesnitt til Processing, med eksempelkode

I motsetning til Scratch gjøres programmeringen gjennom skriving av kode. Den visuelle delen av Processing ligger i outputen. Processing har mange innebygde metoder for tegning av figurer, og bruk av farge. Disse metodene manipulerer et lærret som genereres når man kjører programmet.

4.3 Forberedelse Processing

Ved fullendt løsning av enkelte av oppgaver, lagres et bilde i PNG-format for videre utskrift, hvor størrelsen på bildet er identisk med målene til et A4-ark. For å gjøre det lettere for elevene å tegne med Processing, ble det i forveien sørget for for at de kunne operere med centimeter, i stedet for piksler. For å gjøre dette mulig, utviklet Børre Stenseth et sett med komplimentære funksjoner som kunne ligge i bakgrunnen og

støtte opp programmene som elevene lagde. Denne støttekoden gjorde blant annet så outputen fra Processing-programmene ble skrevet ut som et A4-ark, slik at elevene kunne bruke centimeter direkte i parametre og variabler, i stedet for piksler.

Nøyaktigheten er optimal ved utskrift på både Windows og Mac, så fremt at man skriver ut ved å velge å fylle hele arket på Windows eller velge 100% skalering på Mac. Det kan riktignok oppstå minimale skjevheter på milimeternivå, ved trykking, men dette er ikke av stor betydning.

4.4 Metode, Strupe

4.4.1 Forberedelsesseminar, Strupe

På samme måte som med Os, ble det holdt et forberedelsesseminar med lærerne ved Strupe ungdomsskole og Halden VGS. Seminaret ble holdt felles for begge skolene, ettersom undervisningsopplegget for hver av skolene omhandlet samme tema – beregning rundt to- og tredimensjonale figurer, samt formelregning. Ingen av lærerne her hadde erfaring innen Processing, men én av lærerne ved Strupe var IT-ansvarlig for skolen, og hadde følgelig IT-kunnskap.

Også i dette seminaret fikk lærerne se på foreslåtte oppgaver, og komme med tilbakemelding på disse. Én av lærerne ved Halden videregående skole uttrykket bekymring for at undervisningsopplegget med programmering kunne komme i veien for pensumet. For et best mulig eksamensresultat var læreren mer komfortabel med å ha underivsningsopplegget med Processing etter at klassen hadde gjennomført matematikkeksamen. Dette var grunnen til at undervisningen ved Halden VGS ble utelatt fra dokumentasjonen i denne rapporten, ettersom undervisningen skjedde etter bacheloroppgavens innleveringsfrist.

4.4.2 Metode

I likhet med uttestingen ved Os, foregikk uttestingen ved Strupe Ungdomsskole som en kvalitativ feltobservasjon av deltakere i en undervisning-setting. Deltakere var 24 elever på 9. trinn ved Strupe Undomsskole, og to matematikklærere for dette trinnet. Uttestingen hadde samme hensikt som uttestingen ved Os, med unntak av at verktøy-analysen nå omhandlet Processing, i stedet for Scratch. Videre skulle tema for matematikkundervisningen nå dreie seg om formelregning rundt to- og tredimensjonale figurer.

Uttestingen ble utført over åtte skoletimer, fordelt på tre økter: To økter på 90 minutter hver, og én økt på 180 minutter. Øktene erstattet den ordinære matematikk-undervisningen.

Øktene ble brukt til undervisning i Processing, og arbeid med oppgaver fra oppgavesettet som Børre Stenseth ved Høgskolen i Østfold hadde laget (se vedlegg 3, *Oppgavesett, Strupe*). Anders Heggestøyl tok rollen som lærer for klassen, og ledet undervisningen, mens Anders Bolt-Evensen og Tobias Alvik Hagen fikk ansvar for å hjelpe elevene underveis, og å notere resultater.

Hver økt tok plass i et klasserom, med unntak av økt én, som tok plass i en kantine. Kantina var imidlertid utformet som et ordinært klasserom, med pulter, stoler og en prosjektor. Elevene fikk utdelt datamaskiner eid av Høgskolen i Østfold. Datamaskinene inneholdt all programvare som var nødvendig for å fullføre oppgavesettet. Dette inkluderer Processing, et bildebehandlingsprogram, en nettleser, og en katalog som inneholdt alt av materiale relatert til oppgavesettet. Fordi det kun var 15 maskiner tilgjengelig, satt noen elever i grupper på to, mens andre satt alene.

Undervisningen ble gjort ved at Anders Heggestøyl satt ved en PC som var koblet opp mot en prosjektor, som viste skjermbildet til maskinen for resten av klassen. Ved bruk av prosjektor var det mulig å illustrere konsepter innen programmering ved å skrive kode live for klassen. Hver elev ble oppfordret til å skrive den samme koden som ble gjennomgått på storskjerm. Elevene fikk ofte underveis oppgave om å gjøre endringer i denne koden, for å oppnå et gitt mål. Koden som ble demonstrert live ble også ofte basert på innspill fra klassen – for eksempel hva slags kommandoer man trenger videre, og hvilke parametre disse skal ha.

4.5 Observasjoner, Strupe

Observasjoner i dette kapittelet er dokumentert i relativt høy detalj. Fordi resultatene er hentet gjennom upresise og subkejtive metoder, baseres datagrunnlaget på inntrykk og erfaringer. For å formidle disse erfaringene på en betydningsfull måte, må de beskrives i tilstrekkelig detalj. En annen grunn til å beholde et høyt detaljnivå, er for å sikre at bachelorprosjektet kan levere mest mulig nyttig informasjon til ProgPed. Én av hensiktene bak bachelorprosjektet er tross alt muliggjøringen av fremtidige prosjekter.

4.5.1 Økt én - onsdag 09.03.2016

Etter første økt virket det som at elevene skjønnte og forstod det meste av det som ble gjennomgått da Anders Heggestøyl gjennomgikk "pensum". De virket ut til å forstå RGB-systemet for farger, et koordinatsystem hvor origo er oppe til venstre og positiv Y-akse går ned fra origo istedenfor opp, samt hvordan parametere fungerer med tanke på ulike funksjoner som rect(), ellipse() og translate(). Elevene forstod godt hvordan man brukte variabler da Anders satte opp et par enkle regnestykker i et Processing-program. Alt dette ble bekreftet på flere måter, ved at Anders for hver eneste ting han gikk igjennom for elevene, spurte hva svaret skulle være, for eksempel om hvilken farge som ville dukke opp dersom han brukte *fill(200,0,200),* og elevene helt korrekt svarte lilla.

Uansett hva det gjaldt, parametere, variable, koordinatsystem, rekkefølge av kode, så svarte elevene stort sett riktig på spørsmålene fra Anders. I de tilfellene hvor eleven svarte feil, var det som regel en annen som svarte riktig.

4.5.2 Økt to - fredag 11.03.2016

Resultatet etter økta på fredag var oppløftende og over all forventing. Elevene kom lenger enn forventet. Oppstarten av økta ble imidlertid forsinket av at midlertidige passord som var gitt til elevene, var utgått og at man ikke kunne endre dette passordet. Dermed var det flere som ikke kom seg inn på systemet. Dette resulterte i at man i starten måtte sitte fire og fire sammen, frem til problemet ble løst i lunsjpausen. I tillegg skal det også nevnes at printeren som var ment å bli brukt i dette prosjektet, ikke fungerte.

Først i del to av økta kom man seg i gang med oppgavene. Inntrykket etter at oppgave 1 var ferdig, var at stort sett alle forsto hva de skulle gjøre. Oppgave 1 gikk kort fortalt ut på å lage 2 kvadrater - ett på 3x3 cm og ett på 5x5 cm. Oppgave 2, som gikk ut på å tegne det italienske flagget, gikk også relativt greit for seg. Elevene virket ivrige og prøvde seg frem med variabler og ulike fargekoder for å få frem fargene i det italienske flagget.

Den tredje oppgaven (tegne en utbrettet terning med sidekanter på 5 cm) bød på noen utfordringer for enkelte av elevene. Dette var en oppgave hvor man skulle lage en utbrettet terning hvor alle sidekantene var 5 cm.

Her var problemet visualisering av hvordan en utbrettet terning så ut, og hva man burde gjøre for å kunne få tegne det riktig. Det var også 1-2 elever som lurte på hvordan man flytter origo bakover på X-aksen.

Oppgave 4 (tegne en utbrettet terning med et samlet overflate på 220 cm^2). var den siste oppgaven alle rakk å fullføre før timen var ferdig. I denne oppgaven fikk man oppgitt at en terning hadde et volum på 220 cm^2. Elevene skulle så tegne denne terningen utbrettet.

Her var det en del som lurte på hvordan denne oppgaven skulle løses rent matematisk, i og med at man visste hva overflaten til terninga var, og at man skulle regne seg fram til hvilken lengde alle sidene hadde for å kunne lage en utbrettet terning med den rette overflaten. Noen slet også med å finne ut av hvordan man skulle finne sidelengdene når de hadde regnet seg fram til hva arealet av hver sidekant var.

Mange elever rakk å fullføre både oppgave 5 (tegne en eske med volum på 170 $cm^{3,}$ med 2 cm høyde og lengde på 8 cm) og oppgave 6 (sylinder med volum på 200 cm^3 hvor eleven skal bestemme diameter og tegne sylinderen utbrettet) i løpet av dagen. Den femte oppgaven bød også på sine utfordringer for noen få av elevene, da man hadde en eske med et volum på 170 cm^3, hvor høyden var 2 cm og lengden er 8 og eleven skulle finne ut hva bredden/dybden var. Deretter skulle de lage denne figuren utbrettet i programmet, skrive den ut og klippe og lime for å få figuren i 3 dimensjoner.

3-4 av elevene som kom til oppgave 5, gikk videre til oppgave 6 hvor det var en sylinder som hadde et volum på 200 cm^3, og oppgaven var å bestemme diameter og regne seg frem til lengden og tegne sylinderen utbrettet, for så å klippe og sette sylinderen sammen. Til slutt skulle de måle volumet til sylinderen og se om det stemmer med det gitte volumet.

4.5.3 Økt tre - tirsdag 15.03.2016

Den tredje dagen ble begynt ved at det ble delt ut et alternativt oppgavesett for dem som enten synes de opprinnelige oppgavene ble for vanskelige, eller for dem som ville ha seg en ekstra utfordring. Resultatet av dette ble at de fleste jobbet med det opprinnelige settet, mens andre valgte å prøve seg på det alternative settet. Elevene som valgte å bruke det alternative settet, brukte et av Børres programmer til å skrive løsninga, da de alternative oppgavene var kompatible med dem som allerede var utgitt.

Når det gjelder de som valgte å fortsette med det opprinnelige oppgavesettet, var det noen som hadde problemer med oppgave 5 (tegne en eske med volum på 170 cm3, med 2 cm høyde og lengde på 8 cm). Her gikk problemet på hvordan man skulle regne for å finne fram til bredden/dybden når man har et gitt volum, høyde og lengde.

En annen oppgave som bød på problemer for enkelte, var oppgave 6, hvor det var en sylinder som hadde et volum på 200 cm^3. Her var oppgaven at eleven skulle bestemme seg for en diameter og regne ut hva høyden til sylinderen var, slik at volumet forble på 200 cm^3. Med litt hjelp fra prosjektgruppen klarte han til slutt å regne ut høyden og bredden til den utbrettede sylinderen. (Oppgave 6 - tegne en sylinder som har et volum på 200 cm^3)

Oppgave 7 (tegne norske flagg i ulik størrelse på ett og samme ark) var forvirrende på enkelte elever fordi det i oppgaveteksten stod at man burde lage seg en ny funksjon, *void tegnFlagg()*. Her var det en del forvirring rundt hva en funksjon er og hva "void" betyr.

I oppgave 8 (tegne en likesidet trekant, deretter en som er likebeit og en med ulike sider) skulle man tegne flere ulike typer trekanter. Mer nøyaktig gikk oppgaven ut på å lage en likesidet trekant med sider på 5 cm, en likebeint trekant med 2 sider på 5 cm og en trekant der alle sidene har ulik lengde.

Det var litt vanskelig for enkelte fordi trekanten som ble lagd med funksjonen triangle, var en rett linje. Eleven som hadde dette problemet så ikke at man ved å hele tiden ha lineært økende koordinater, for eksempel 3,3, 6,6, 9,9, endte opp med én lang, sammenhengende linje istedenfor en vanlig trekant. Eleven valgte til slutt å heller bruke metoden med line(), translate() og rotate(), som tegner en linje, flytter origo og deretter roterer origo for på den måten å lage en trekant hvor eleven selv kunne bestemme lengden på linjene.

Eleven som hadde spørsmål om oppgave 7 (tegne norske flagg i ulik størrelse på ett og samme ark), rakk å komme seg til oppgave 9 (Tegne en pyramide med kvadratisk grunnflate på 5 cm og høyde på 6 cm). Den niende oppgaven gikk ut på at en pyramide hadde en kvadratisk grunnflate på 5x5 cm og en høyde på 6 cm.

Elevene skulle tegne denne pyramiden utbrettet med riktige dimensjoner. Problemet med denne oppgaven var at man her var inne på temaet med trigonometri, noe som ikke er del av ungdomskolepensum. Eleven som rakk å komme til denne oppgaven, var

dermed en av de elevene som mot slutten av den tredje økta ble helt ferdig med hele det opprinnelige oppgavesettet.

Én av gruppene prøvde seg på oppgave 12 i det alternative oppgavesettet, som går ut på å fullføre et program som tegner en kube i tre dimensjoner. (se vedlegg 3.2, *Alternativt oppgavesett i Processing for Strupe*) Dette var imidlertid mot slutten av siste økt, så gruppa rakk ikke å komme så langt i denne oppgaven.

Problemene som er nevnt ovenfor, gjaldt bare noen få av elevene. De aller fleste virket ut til å klare seg veldig bra. Flesteparten av elevene virket også ut til å like dette eksperimentet. Da læreren spurte om dette var bedre enn vanlig undervisning, svarte elevene et klart og tydelig ja.

En ting som det er viktig å nevne, er at det under oppgaveløsninga ikke ble benyttet printer. Meningen var at elevene skulle skrive ut hver tegning på papir og brette dette sammen til en figur i tre dimensjoner. Grunnen til at printeren ikke ble brukt diskuteres i kapittel 4.6.5.

For noen elever var det i starten forvirrende om de skulle bruke piksler eller centimeter når de skulle angi verdier på x- og y-aksen. Programmene var laget slik at det ble gjort en omregning fra piksler til centimeter, slik at hvis en oppgave ba elevene om å tegne en rekangel der startpunktet for rektangelet var 1,1, trengte elevene bare å regne seg 1 cm bortover x- og y-aksen, istedenfor å først angi antall piksler og deretter måle seg fram til 1 cm. Dette virket det ut til at noen av elevene hadde glemt, slik at de isteden regnet med piksler med det resultatet at rektangelet dukket opp på feil sted eller ikke dukket opp i det hele tatt.

4.6 Diskusjon, Strupe

4.6.1 Introdusere elever til Processing

Det var svært effektivt å introdusere programmering gjennom live demonstrasjon, hvor elevene samtidig skrev ned koden som ble gjennomgått. Det forhåndsplanlagte “Hallo, verden!”-programmet, som er beskrevet i vedlegg 5, *Gjennomføringsbeskrivelse, Strupe*, gjorde det mulig å gradvis introdusere konsepter i en gjennomtenkt rekkefølge.

Det kan være fordelaktig å la elevene samtidig skrive ned all koden som blir gjennomgått av flere grunner. Det er en fordel at elevene får syntaksen “inn i fingrene” i et tidlig stadium, slik at elevene i forveien av oppgaveløsingen er vant med å for eksempel ende kommandoer med semikolon, og å omringe parametre med parenteser. Å eliminere disse problemene tidlig sikrer at de ikke kommer i veien for de reelle problemene senere. Videre lar det elevene oppleve hvilke feil som enkelt kan oppstå, og raskt få hjelp med disse. Det gjør det også mulig for elevene å eksperimentere med eksemplene selv, og å bruke disse eksperimentene som grunnlag for forslag underveis i timen.

4.6.2 Læringsterskel for konsepter innen programmering og Processing

Den første økta, som bestod utelukkende av undervisning, gav et inntrykk av hvilke konsepter som elevene forstod raskt, og hvilke som opplevdes som vanskelig. Følgende konsepter var trivielle for elevene å forstå: Hvordan koden kjøres linje for linje, bruk av parametre, grunnleggende Processing-syntaks, og grunnleggende tegning ved hjelp av kommandoer som rect() og text(). Forståelse for disse konseptene ble demonstrert gjennom hyppige innspill fra klassen gjennom introduksjonen, samt i senere oppgaveløsing.

Det var konsepter som var forståelige for klassen, men som trengte egne forklaringer. En viktig del for bruk av Processing er å forstå det omvendte koordinatsystemet. Elever ved 9. trinn er vant til bruk av ordinære koordinater, men ikke systemer der positive verdier på y-aksen illustreres nedover. Å tegne opp to koordinatsystemer, ett av hver type, og å tegne eksempel-koordinater på begge systemene, var en effektiv måte å introdusere konseptet på. Elevenes forståelse for dette ble bekreftet gjennom oppgaveløsingen, hvor ingen hadde problemer med y-aksens retning.

Å uttrykke farger gjennom RGB-systemet var noe prosjektgruppen forutså kunne vise seg å være en hindring. Til gruppens fornøyelse var dette overraskende raskt forstått – allerede i introduksjonen kunne elever svare på hvilke farger som ville være resultatet av diverse fargekommandoer, uten å ha kjørt koden. Dette viste seg heller ikke å være noe problem gjennom oppgaveløsingen.

En mulig årsak til at RGB-systemet ble såpass enkelt tatt i mot av elevene, er at systemet trekker paralleller til fysisk blanding av farger. Å øke verdier i RGB-notasjon kan sammenlignes med å legge til mengder maling. Så ved å definere én mengde rødt, én mengde blått, men ikke noe grønt, gjennom for eksempel *fill(200,0,200),* får man lilla,

akkurat som i den fysiske verden. Dette vil være en intuitiv måte for elever å definere farger på i programmering, i motsetning til gjennom heksadesimale verdier.

Translate() faller også under kategorien av konsepter som var forståelig for de fleste, men som krevde en del forklaring og eksempler.

4.6.3 Utfordrende med selvstendig bruk av variabler

Av konsepter som ble introdusert i økt én, var det variabel-begrepet som var vanskeligst for elevene. Det er naturligvis umulig å vite hva hver elev i klassen forstod eller ikke, men da klassen fikk spørsmål til diverse variabel-eksempler, var det vesentlig færre innspill enn ved tidligere temaer. Under oppgaveløsingen i senere økter var det også svært få elever som brukte variabler uoppfordret for å løse oppgavene. Dette er noe som speiler uttestingen ved Os – selv om flere elever forstår hvordan en variabel kan lagre og representere en verdi, er det svært få elever som mestrer å benytte egne variabler på en konstruktiv måte.

På grunn av denne manglende forståelsen for hvordan variabler kan brukes for å gjøre problemer enklere, selv der de ikke er nødvendige, valgte Anders å demonstrere to forskjellige løsninger på oppgave 2 (tegne det italienske flagget) – én med bruk av variabler, og én uten. Å demonstrere to konkrete eksempler ved siden av hverandre virket som en brukbar måte for å forklare nyttig bruk av variabler. Eksempelet gav direkte kontekst som elevene kunne ta utgangspunkt i i senere oppgaver. Den viktigste delen av demonstrasjonen var hvordan man ved hjelp av variabler kunne skalere hele det italienske flagget ved å endre på kun én verdi, fordi hver variabel hadde et forhold til hverandre. Noen få elever viste god forståelse for bruk av variabler under oppgaveløsingen, mens flesteparten forsømte det, med mindre bruk av variabler var aktivt oppfordret.

4.6.4 Vanskelighetsgrad av undervisningsmateriale

Oppgave 1 (tegne 2 kvadrater 3x3 cm og 5x5 cm) var triviell, ettersom svaret allerede var oppgitt i mal-programmet. Noe av poenget med oppgaven var å gi elevene muligheten til å leke seg litt rundt, og bli kjent med å både redigere eksisterende kode, og å skrive egne kommandoer. Det var flere elever som var forvirret over denne oppgaven, ettersom kvadratene som oppgaveteksten ber om allerede er tegnet. Enkelte elever valgte av den grunn å hoppe videre til neste oppgave. For å unngå forvirring burde denne oppgaven enten vært omformulert, eller fjernet svaret fra malen. Dersom

løsningen på rektanglene var fjernet fra malen, ville oppgaven fremdeles vært av passende vanskelighetsgrad til å være oppgave 1. Utforskingsbiten kunne heller vært en valgfri b-oppgave.

Oppgave 2 (tegne det italienske flagget) var en veldig egnet oppgave, som tok for seg grunnleggende bruk av assorterte konsepter som var introdusert i økt én. Alle fikk til denne oppgaven, og elevene gav inntrykk for at det var en morsom oppgave å jobbe med. Det eneste problemet med denne oppgaven var at noen elever var forvirret over proporsjonen mellom. Dette skyldtes imidertid uklar formulering i oppgaveteksten, og ikke manglende kunnskap om proporsjoner.

Oppgave 3 (Tegne en utbrettet terning med sidekanter på 5 cm) var en god måte å introdusere tredimensjonale figurer, ettersom figuren i seg selv var ganske simpel. Mange elever demonstrerte her god beherskelse av translate-funksjonen, samt mestring av det omvendte koordinatsystemet. Noen elever hadde imidlertid vanskeligheter med å forestille seg hvordan en terning ville se ut utbrettet.

Etter oppgave 4 (Tegne en utbrettet terning med et samlet overflate på 220 cm^2) eskalerte vanskelighetsgraden svært raskt, noe som vises ved at mesteparten av klassen fikk til oppgave 4 uten nevneverdige problemer, men svært få elever klarte oppgave 5 (tegne en eske med volum på 170 cm^3, med 2 cm høyde og lengde på 8 cm) og videre uten betydelig hjelp. Dette skyldes sannsynligvis to ting: Først og fremst er det mulig at "formelsnu"-aspektet ved oppgavene ble introdusert for tidlig. Det vil si oppgaver der elevene må kunne omgjøre en formel for å kunne løse oppgaven. Dette er noe som virket gjørbart, men relativt utfordrende i seg selv for elever ved 9. trinn. Problemet oppstår når de i tillegg skal programmere det, uten å ha særlig øving i å programmere bruk av formler uten å snu på dem.

Et eksempel er oppgave 6 (sylinder med volum på 200 cm^3 hvor eleven skal bestemme diameter og tegne sylinderen utbrettet), der man får oppgitt volumet av en sylinder, og man definerer sin egen omkrets. Dette er en veldig lur oppgave, ettersom man på grunnlag av det statiske volumet, og den egendefinerte høyden, må komme frem til riktig lengde – noe som vil verifiseres av den utskrivbare figuren. Den er imidlertid litt for komplisert til å stå såpas tidlig i oppgavesettet.

Formelen som vanligvis tas utgangspunkt i når det gjelder sylindere er $V=\pi r^2 h$. De fleste elevene viste kjennskap til bruk av denne formelen. Trikset med oppgaven var imidlertid å snu om formelen for å finne høyden, gjennom $h=\frac{V}{\pi r^2}$. Problemet lå ikke i

at elevene var ute av stand til å gjøre om formelen, dersom de tenkte seg om. Det lå heller i at de ble utfordret på for mange kanter på én gang, for tidlig i settet. En mer passende vanskelighetsgrad kunne vært å først programmere utregning av volum, med høyde og radius oppgitt, slik at elevene først kunne bli kjent med programmering av formler. Deretter kunne man fortsatt til den originale oppgave 6, som tar skrittet videre med formelsnu.

Oppgave 8 (tegne en likesidet trekant, deretter en som er likebeit og en med ulike sider), 9 (Tegne en pyramide med kvadratisk grunnflate på 5 cm og høyde på 6 cm) og 10 (Tegne en pyramide med volum 100 cm^3, grunnflate som likesidet 8 cm trekant), som tok for seg tegning av forskjellige typer trekanter, havnet utenfor pensumet for de øktene prosjektet hadde til rådighet. I begynnelsen av økt tre holdt Anders Heggestøyl en kort introduksjon for bruk av line(), rotate() og triangle(), men denne var merkbart mindre effektiv enn tidligere introduksjoner. *line(), rotate()* og *triangle()* diskuteres i avsnitt 6.2.2.2 Det var ikke nok tid til å kunne gi den introduksjonen som var nødvendig for å løse trekant-oppgavene på en god måte. Det vil være gjørbart i opplegg som har tilgang på flere skoletimer, men i fremtidige opplegg med samme timebegrensning bør heller trekant-temaet droppes, i stedet for å halvveis begynne å komme innpå det.

4.6.5 Praktiske problemer

En ansatt fra IT-avdelingen ved Høgskolen i Østfold fikk i oppdrag å sette opp printeren før undervisningen. Det oppstod derimot problemer med oppsettet, da han ikke hadde riktig DNS- og IP-adresse til nettverket. Det var kun DNS og IP-addresser fra Halden Kommune som kunne benyttes. Disse var ikke tilgjengelig for den ansatte.

Datamaskinene elevene disponerte var satt opp med egne brukerkontoer på forhånd av IT-avdelingen ved Høgskolen i Østfold. Ved starten av økt 2 oppstod det påloggingsproblemer med rundt halvparten av datamaskinene. Elevene fikk beskjed om at passordet var utløpt for brukeren. Dette resulterte i at halvparten av elevene måtte fordeles på de restrerende datamaskinene. Det ble nå ca 4 elever per datamaskin. Dette viste seg å være lite fruktbart for elevenes læring.

Tobias og Anders H. løste dette ad-hoc under øktens pause. Ved å ringe en IT-ansatt fikk de tak i administrator-passordet til datamaskinene og Googlet fremgangsmåten for å endre utløpsdatoen på passordene. Problemet var løst etter rundt 30 minutter, i løpet av skolens lunsjpause.

4.7 Konklusjon, Strupe

Å demonstrere konsepter innen programmering ved å kode live for klassen var en svært effektiv måte å introdusere programmering på. Å oppfordre klassen til å skrive og kjøre koden som blir gjennomgått underveis byr på mange fordeler.

Mange grunnleggende konsepter innen programmering og Processing er intuitivt å forstå for elever ved 9. trinn. Dette inkluderer blant annet konsepter som tegning gjennom bruk av metoder som *rect()* og *ellipse()*, definering av farger gjennom RGB-verdier, hvordan kode i Processing eksekveres (og hvordan rekkefølgen på kommandoer påvirker dette), hvordan parametre fungerer, og grunnleggende syntaks for Processing.

Bruk av variabler er lite intuitivt for elevene. Det er ikke nødvendigvis hvordan variabler fungerer som er vanskelig å forstå, men heller deres applikasjon i problemløsning. Flere elever demonstrerte forståelse for hvordan variabler fungerer, men svært få brukte variabler uoppfordret for å løse et problem. Det er uvisst hvorvidt dette skyldes det begrensede omfanget på undervisningen, eller om kompleksiteten er for høy for trinnet.

Processing viste seg å være et godt egnet verktøy for å introdusere elever ved 9. trinn til programmering. Utviklingsmiljøet er simpelt og hurtig å bruke, er egnet til å demonstrere i plenum, og har et intuitivt grensesnitt. Språket har en veldig enkel syntaks, og programmer krever minimalt med kode som ikke har med selve funksjonaliteten å gjøre. Fokuset på tegning og visuell feedback gjør programmeringen mer tilgjengelig og engasjerende for elevene.

Begynnelsen av oppgavesettet brukt i undervisningen ved Strupe hadde egnet kompleksitetsnivå i henhold til elevenes matematikk- og programmering-kunnskaper. Fra oppgave 5 (tegne en eske med volum på 170 cm3, med 2 cm høyde og lengde på 8 cm) og ut ble økningen i vanskelighetsgrad noe bratt, og bør justeres ned i fremtidige undervisningsopplegg. Ambisjonen om bruk av printer for disse oppgavene hadde god hensikt, men bød på for mange logistiske problemer til å kunne rettferdiggjøres.

5 Overordnet diskusjon

5.1 Introduksjon

Dette kapittelet tar for seg en kortfattet diskusjon på tvers av uttestingene. Her diskuteres det som ikke er begrenset til selve uttestingene, i tillegg til sammenligning av det som var felles for begge.

5.2 Investering vs nytteverdi

Én av bekymringene rundt innføring av programmering i skolen er investeringen som kreves for å dra nytte av programmeringen som verktøy. Her må nytteverdien sammenlignes med investeringen. Å bruke programmering krever en tidsinvestering i form av undervisningstid, det forutsetter kunnskap fra lærere, og det kan i enkelte tilfeller forutsette teknologikunnskap, for eksempel rundt bruk av datamaskiner.

Naturligvis blir inngangsterskelen for bruk av programmering et viktig poeng. Hvor mye undervisning må til før barn og ungdom er i stand til å løse matematiske problemer gjennom programmering? Vil denne bruken av programmering øke matematikkforståelsen nok til å være verdt investeringen? Dette prosjektet alene vil ikke kunne konkludere noe konkret rundt hvorvidt bruken av programmering er med på å øke matematikkforståelsen hos elever. Prosjektet kan imidlertid komme med erfaringer rundt den andre delen av problemstillingen, nemlig den nødvendige investeringen.

Selv om dette er et prosjekt av relativt lite omfang, og med tilnærmet ubetydelig datagrunnlag, kan man allikevel få en vag idé om hva som skal til for å få barn og ungdom i gang med grunnleggende programmering, hva som er intuitivt for dem, og hva som er vanskelig. Videre vil man også kunne si noe om hva man kan forvente elever til å være i stand til, i etterkant av et kort undervisningsopplegg. Hva som er mulig etter lengre undervisningsopplegg er det opp til fremtidige prosjekter å vise.

5.3 Inngangsterskel

Ett av de viktigste punktene rundt investeringen som kreves for bruk av programmering er inngangsterskelen. For å dra nytte av programmering, er man selvsagt nødt til å kunne programmere, noe som er fremmed for de fleste elever. Her spiller verktøy en stor rolle, og verktøyene som ble brukt gjennom uttestingene gjorde en jobb med å holde inngangsterskelen så lav som mulig.

Processing har en svært forenklet syntaks, som gjør at man ikke behøver å ende opp med masse ekstra kode som er vanskelig å forklare for elevene. Dette betyr at mesteparten av inngangsterskelen består av å forstå hvordan et dataprogram kjøres, grunnleggende syntaks, og et repertoire av funksjoner, noe som ble dekket i løpet av fire timer med undervisning. Hva som ikke ble dekket tilstrekkelig i løpet av denne tiden var bruk av variabler, og mer avansert tegning gjennom linjer og rotasjon. Det er åpenbart at bruk av variabler krever en nøyere gjennomgang enn hva som var gjort i dette prosjektet.

Scratch senker inngangsterskelen desto lavere, ved å gå vekk i fra skriftlig syntaks totalt. Man kan argumentere for at det finnes en syntaks også i Scratch, men her er den visuelt fremstillet, og ble godt tatt i mot av barneskoleelevene. Det tok ikke mye undervisning i Scratch før elevene var i stand til å begi seg ut på oppgaver på egenhånd. For å avansere videre utover det helt grunnleggende, er man imidlertid nødt til å introdusere variabler nøye, noe som øker den nødvendige tidsinvesteringen.

Det er ikke tvil om at elevene på både Os og Strupe kom godt i gang med problemløsing gjennom programmering. I begge tilfellene ligger den største “fartsdumpen” helt i begynnelsen, det vil si den nødvendige bolken med forståelse som er nødvendig for å beherske grunnleggende programmering. Når denne er på plass, blir det langt enklere å utvide nytteverdien gjennom å gradvis introdusere nye metoder og konsepter. Dette betyr at jo kortere undervisningsopplegget er, jo mindre kosteffektivt vil forholdet mellom investering og nytteverdi være. I et lengre undervisningsopplegg vil ikke denne første fartstumpen utgjøre en like betydelig del av undervisningsperioden.

Fordi vi ikke ennå vet nok om takhøyden på den potensielle nytteverdien, blir det for tidlig å si noe om hvor langt et optimalt undervisningsopplegg ville vært.

6 Evaluering av verktøy og utviklingsmiljøer

Målsettingen til ProgPed er å teste ut hvordan man kan bruke programmering som et pedagogisk verktøy for å gjøre matematikk mer forståelig for elever.

6.1 Introduksjon

I kapittel 6 gjennomgås ulike verktøy som er gode til bruk i undervisning på ulike måter og som alle har en enkel læringskurve. Scratch (5.2.1) som ble brukt under uttestingen

på Os (se kapittel 3) og Processing (5.2.2) som ble brukt på Strupe (se kapittel 4) er de verktøyene som blir gjennomgått først. Deretter gjennomgås ulike verktøy som ligner på disse. Disse verktøyene står listet alfabetisk etter navn. Python, JavaScript og Logo er de som på ulike måter ligner mest på Processing, mens MIT App Inventor og Tynker er dem som ligner mest på Scratch. I noen av underkapitlene introduseres begreper som det var nødvendig å gå gjennom for elevene under undervisningen. Verktøyene gjennomgås ikke på detaljnivå, ettersom konseptene til verktøyene bør samsvare med elevenes nivå og de oppgaver som ble utført. Det introduseres derfor ikke konsepter som pekere, arrays, metoder og klasser. Det forutsettes ellers at leseren har kjennskap til grunnleggende programmering. I kapittel 2.2 blir begrepene “Learning by doing” og “Doing with images makes symbols” diskutert. Verktøyene fokuserer mer på resultater ved å prøve og feile for på den måten å lære programmering.

6.2 Aktuelle programmeringsspråk

6.2.1 Scratch

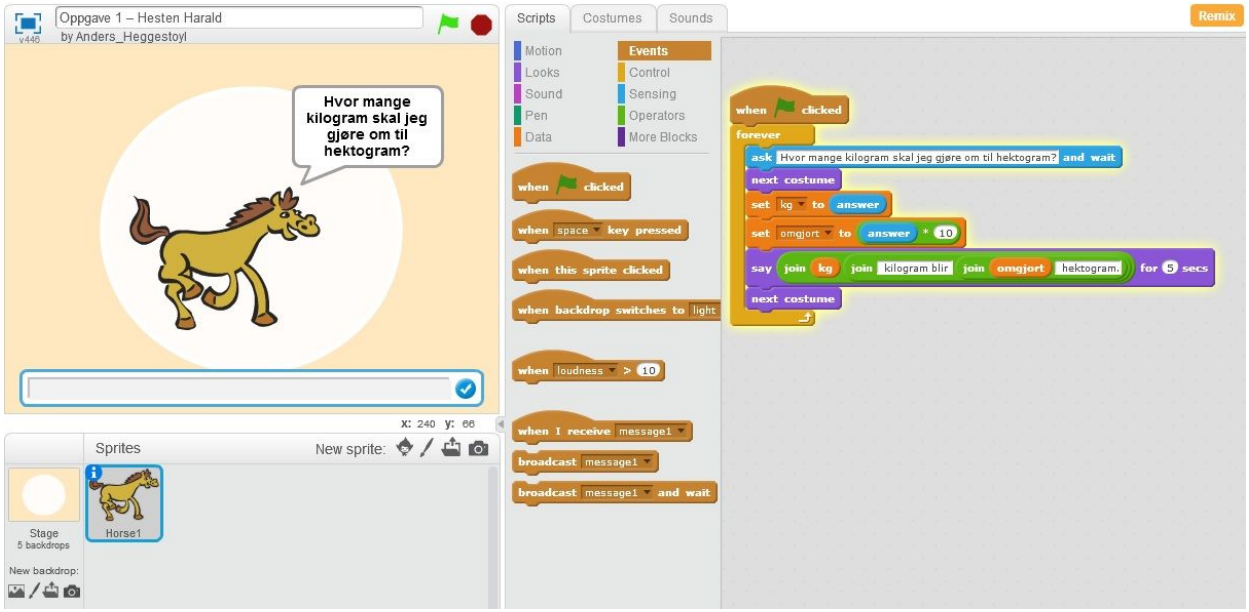
Scratch er et grafisk programmeringsspråk utviklet ved Massachusetts Institute of Technology (MIT) i 2004 (Logo Foundation, 2015), som bygger på Logo (se avsnitt 6.2.4).

Scratch er laget slik at hver "kommando" er lagt i ulike grupper. For eksempel så finner man addisjon, subtraksjon, multiplikasjon og divisjon, samt likhet, større/mindre enn og andre matematiske operasjoner under "Operatorer", løkker og hvis-tester under "Styring", variable under "Data", og så videre.

Grensesnittet i Scratch kan grovt sett deles i 2 deler. I den delen som er lengst til venstre, er det et vindu hvor programmet kjøres (øverst) og en oversikt over figurer og bakgrunner (nederst). Den andre delen er delt inn i 3 faner: Scripts, Costumes og Sounds. I fanen Costumes finner man ulike “kostymer” av figuren som er valgt, mens man i Sounds finner ulike lyder den valgte figuren kan lage. De ulike gruppene med kommandoer som nevnes over, ligger i fanen Scripts. I denne fanen er det et tomt vindu til høyre hvor man bygger opp programmet ved å koble sammen ulike klosser man kan “dra” ut fra de ulike gruppene.

Figur X illustrerer hvordan selve grensesnittet ser ut. Delen til høyre i denne figuren viser klossene som er koblet sammen, delen i midten viser de ulike kategoriene, mens

delen til vestre viser kjøringa av programmet (øverst) og en oversikt over figurer og bakgrunner (nederst).



Figur X: Kjøring av oppgave 1 på Os (Hesten Harald - konvertere mellom enheter for vekt).

Under uttestingen på Os ble dette verktøyet brukt for å konvertere mellom målenheter. Vår erfaring er at dette er et fint verktøy å bruke på barneskolenivå. Scratch er også et verktøy som er rettet mot barn og som er mye brukt over hele verden. Koden er et sett med bilder/blokker, hvilket gjorde det lett leselig og forståelig for elevene. På den måten kan man enkelt lage oppgaver som går på det å fullføre et uferdig program, for eksempel ved å be elevene fullføre et program som regner om fra meter til centimeter. Da kan man "utelate" den matematiske delen, slik at eleven blir nødt til å finne ut om han/hun må plusse, trekke fra, gange eller dele og sette denne operatoren på rett plass i programmet for å få programmet til å komme fram til korrekt svar.

6.2.1.1 Variabler i Scratch

I løpet av økt nr 2 ble begrepet “variabel” introdusert for elevene på Os. Variabler i Scratch opprettes ved å klikke på Data-gruppen og “Make a variable”. I motsetning til Processing, er ikke variabler i Scratch typeavhengig - hva slags variabel det er snakk om, blir avgjort av hvilken verdi variabelen får av brukeren. For å gi den nye variabelen en verdi, drar man “Set variabelnavn to []” ut i det store vinduet til høyre og skriver for eksempel 5 hvis variabelen er et tall eller “Hei, verden” hvis det er en tekststreng.

Scratch finnes både online, på https://scratch.mit.edu, og offline for Windows, Mac og Linux. Offline-versjonen krever at man har installert Adobe Air.

6.2.1.2 Tester og løkker i Scratch

I løpet av gjennomgåelsen av oppgavene i Scratch (se vedlegg 3, *Oppgavesett, Os*), ble det brukt noen ulike kontrollstrukturer som løkker og tester. Det forutsetter at leseren av denne rapporten vet hva løkker og kontrollstruktur er, men siden det var nødvendig å introdusere dette for elevene, kan det være greit å likevel ta det med. En løkke i Scratch er en kloss med teksten “Gjenta til”, samt blokken med teksten “For alltid.” Denne kjøres helt til betingelsen i løkka er oppfylt. Skal man gjennomføre en hvis-sjekk i Scratch, kan man enten bruke blokka “If” eller blokka “If - Else”.

6.2.2 Processing

Processing er et objektorientert open-source programmeringsspråk som bygger og kjører på Java. Dette er et språk for de som liker å jobbe med både programmering og grafikk. Verktøyet introduseres i kapittel 4.

Matematisk fungerer Processing fint når man jobber med geometriske figurer, både i to og tre dimensjoner, da man raskt og enkelt får en grafisk indikasjon på om det man holder på med er riktig eller ikke. Dessuten gir Processing en god indikasjon på forståelse av bruk av variabler (ref. algebra og ukjente i matematikk) og X-, Y- og eventuelle Z-koordinater (3D).

Processing er et objektorientert språk. Man kan selv velge om man vil bruke en tradisjonell C-lignende måte å lage programmer på Siden Processing er objektorientert, kan man bruke en mer moderne måte å lage programmer på. Man kan da bruke klasser og objekter. Ulempen med å bruke den objektorienterte metoden med klasser og objekter, er at programmeringsbiten kan virke mer utfordrende med tanke på at eleven i tillegg til å måtte forstå konseptet med variabler basert på primitive datatyper også må forstå konseptet med variabler som pekere til objekter, uten at det nøvdvendigvis er med på å øke elevenes matematiske forståelse. Under uttestingen på Strupe, var det den “tradisjonelle” metoden som ikke bruker klasser og objekter som ble brukt. Dette med objektorientering vil i mange tilfeller være den naturlige veien videre, spesielt om man går på videregående skole.

Processing finnes for de vanligste systemene som Windows, Linux og Mac, samt i en eldre utgave for BSD-baserte systemer. Hjemmesiden til Processing finnes her: https://processing.org/

6.2.2.1 Variabler i Processing

Under uttestingen på Strupe ble variabler av typene float, int og String introdusert for elevene. Float er en primitiv datatype som består av desimaltall. Int er datatypen for heltall. String er et objekt av en klasse som behandler tekststrenger. Da begrepet variabel ble introdusert, og spesielt av typen flyttall (float) for elevene ved Strupe, ble dette definert som en form for algebra og ligninger hvor man skal finne ut hvilken verdi den ukjente har, hvilket kode X under viser et eksempel av:

```
int a = 3 + 7;
int b = a * 2;
```
Kode X: Eksempel på bruk av variabler i Processing.

6.2.2.2 Metoder brukt i Processing

Dette kapittelet er ikke ment som noen generell definisjon på hva en metode er. Dette er derimot et kapittel som lister opp de metodene det var nødvendig å introdusere for elevene og/eller metoder som er brukt i forbindelse med oppgavene elevene fikk og er således kun relevant i forbindelse med kapittel 4, som omhandler undervisningen på Strupe. Under undervisingen og oppgaveløsningen ble det brukt følgende metoder:

size(float width, float height) setter størrelsen på vinduet som lages når man kjører et program. *size* tar bredde og høyde som parameter.

translate (float x, float y, float z) flytter origo, slik at den ikke forblir oppe til venstre. Som parametere tar denne metoden antall piksler bortover X- og Y-aksen. Man kan angi både positive og negative verdier, slik at man kan flytte origo i alle tenkelige retninger.

rotate(float angle) roterer et gitt antall grader rundt origo. Parameteret som *rotate()* tar, er antall grader i radianer som det skal roteres, det vil si verdier mellom 0 og 2 * π.

draw() er én av 2 hovedmetoder i Processing. Draw-metoden tegner opp de objektene som programmet skal tegne. Draw-metoden kjøres automatisk og gjentas helt til man avslutter programmet.

setup() er den andre av de 2 hovedmetodene i Processing. Dette er en metode som kjøres automatisk når programmet starter. I motsetning til draw() kjøres denne metoden bare én gang, og kjøres alltid før draw(). Det er i setup-metoden man angir egenskaper som antall bilder i sekundet (frameRate) og hvilken størrelse programvinduet skal ha gjennom metoden *size()*.

triangle(float x1, float y1, float x2, float y2, float x3, float x3) tegner opp en trekant, der parameterne er punktene for hvert hjørne i koordinatsystemet.

quad(float x1, float y1, float x2, float y2, float x3, float y3, float x4, float y4) tegner en firkant med ulike sidelengder og dermed vinkler som ikke er 90 grader. En god idé når man jobber med denne funksjonen, er å følge en struktur som går enten med eller mot klokka med utgangspunkt i det første hjørnet. Parameterne som *quad()* tar, er koordinatene for de 4 hjørnene.

line(float x1, float y1, float x2, float y2) tegner en rett linje mellom 2 punkter. Variant 1 tegner en linje i 2 dimensjoner, mens variant 2 tegner en linje i 3 dimensjoner. De fire parameterne som *line()* tar, er X- og Y-koordinater for hvor hjørnene skal være.

ellipse(float a, float b, float c, float d) lager en oval ellipse. De to første parameterne angir startpunktet på X-og Y-aksen, mens de to siste angir ellipsens bredde og høyde. De to første parameterne er X- og Y-koordinater for sentrum av ellipsen, mens de to siste er ellipsens bredde og høyde. Man kan fint bruke metoden *ellipse()* til å lage en sirkel ved at c og d har samme verdi. Da vil både c og d fungere som en radius.

rect(float a, float b, float c, float d) tegner opp et rektangel. Parameterne a og b angir hvor på koordinatsystemet det øverste, venstre hjørnet i rektangelet er, mens c og d angir henholdsvis bredden og høyden.
Man kan bruke *rect()* til å lage et kvadrat ved å gi c og d like verdier.

stroke() lager fargede kanter og linjer rundt et objekt.
Den finnes i flere utgaver. De utgavene som ble brukt i oppgavene, var

- *stroke(float v1, float v2, float v3)*. Dette er den vanligste måten å bestemme fargen på, ved hjelp av det tradisjonelle RGB-systemet. Parameterne har en

verdi mellom 0 og 255. De tre parameterne som metoden tar, er verdien for rødt, grønt og blått.
- *stroke(int rgb)*. Denne varianten følger også RGB-prinsippet, men bruker heller et heksadesimalt tall for å avgjøre fargekoden. Før selve fargekoden bruker man tegnet # eller 0x.

noStroke() ble kun brukt i forbindelse med ekstraoppgaven som gikk ut på å fullføre et program som lager en kube i 3 dimensjoner. Metoden fjerner kantlinjer på objektet som tegnes.

fill() bestemmer hvilken farge et objekt skal fylles- og fargelegges med. I likhet med metoden *stroke()* over, kommer *fill()* i flere varianter.
- *fill(float v1, float v2, float v3)* Dette er den utgaven som oftest ble brukt til å løse de ulike oppgavene som elevene fikk. I denne varianten av metoden har hver av de tre parameterne en verdi på mellom 0 og 255. De tre parameterne som metoden tar, er verdien for rødt, grønt og blått.
- *fill(int rgb)*. Denne varianten er akkurat som den forrige, med den forskjellen at man her bruker heksadesimale tall istedenfor "vanlige" tall mellom 0 og 255. Bruker man denne varianten, må verdien til parameteren begynne med # eller 0x.

noFill() gjør at objektet som tegnes, ikke kan "fylles", det vil si at det ikke kan fargelegges. Dersom denne metoden brukes sammen med *noStroke()* ovenfor, vil ingenting tegnes på skjermen. Denne metoden ble bare brukt i det alternative oppgavesettet.

frameRate(float fps) avgjør hvor mange ganger bildet skal oppdateres i sekundet. Dette er en funksjon som man ikke er så veldig nødvendig, men som kan være nyttig dersom man jobber med objekter i 2 dimensjoner og dermed ikke bruker noen animasjoner, ved at man da kan be programmet om å oppdatere seg i for eksempel ett bilde i sekundet.

6.2.2.3 Kort om objekt

Et objekt i programmering er en del av et program som inneholder en gitt type data og som kan kjøre metoder på disse dataene (Trætteberg, NTNU, 02.01.2014). Dette leder inn mot det som kalles objektorientert programmering. Denne rapporten går ikke dypere inn på dette, men objektorientert programmering vil være det naturlige steget videre, spesielt for elever på videregående skole.

6.2.3 JavaScript

JavaScript er et språk som kom i 1995 (W3.org, 2012) og som har en ganske enkel syntaks. Det er veldig tett bundet opp mot HTML og CSS. Det er et språk som er lett å lære seg, men som har store begrensninger i hva man kan gjøre med det dersom man ikke kombinerer JavaScript med HTML og CSS. Hvis man derimot gjør det, er JavaScript et språk som er svært allsidig og som dermed kan brukes i alle sammenhenger. Dermed kan kompleksiteten med JavaScript bli ganske stor.

JavaScript er syntaksmessig sammenlignbart med Processing. Skal man lage en funksjon som ikke tar parametere, bruker man ordet “function” foran funksjonsnavn etterfulgt av to vanlige parenteser. Til sammenligning bruker for eksempel Java, Processing og C/C++ typenavn, etterfulgt av navnet på funksjonen og de to vanlige parentesene. Funksjonskallene i JavaScript foregår på samme måte som i Processing.

Vil man opprette en variabel, bruker man ordet “var” foran variabelnavn. Man skriver med andre ord ikke hvilken datatype det er – det avgjøres ut fra verdien til variabelen.

Kode X viser en eksempelkode omsluttet av HTML-taggene “script”:

```
<body>
  <script type="text/javascript">
     kjorEksempel("Bruker");
     function kjorEksempel(navn)
     {
       var korrektNavn = false;
       if(navn == "Bruker"){
            korrektNavn = true;
            document.write("Du skrev korrekt brukernavn.");
       }
       else{
            korrektNavn = false;
            document.write("Du skrev feil brukernavn.");
       }
     }
  </script>
</body>
```

Kode X: Eksempelkode skrevet i JavaScript. Koden gjør et enkelt sjekk av et brukernavn

Kode i JavaScript kan legges i en HTML-fil, både i hodet (mellom <head> og </head>) og "kroppen" (mellom <body> og </body>), slik som i eksempelet over eller i en egen fil som man kan kalle på fra HTML-filer, slik som i kode X:

```
<script type="text/javascript" src="js/main.js"></script>.
```

Kode X: HTML-kode som laster inn et skript fra filen js/main.js

I kode X ser man hvordan man kan laste inn en skripfunksjon inne i "body"-taggen med hjelp av *onload*:

```
<body onload="funksjonsnavn();">
```

Kode X: Kode for å laste inn en skriptfunksjon i body onload.

6.2.4 Logo

Logo er et Lisp-basert programmeringsspråk som kom i 1967. (Logo Foundation, 2015).

I avsnitt 6.2.1 diskuteres programmeringsmiljøet Scratch. Dette er et Logo-basert programmeringsspråk som kom 2004 og som ble utviklet hos MIT (Logo Foundation, 2015). Likheten mellom Scratch og Logo er at Scratch benytter seg av blokker som igjen kan sies å inneholde Logo-kode, for eksempel så tilsvarer blokken "Gå N steg" i Scratch kommandoen "forward N", der N er det antallet piksler figuren skal bevege seg.

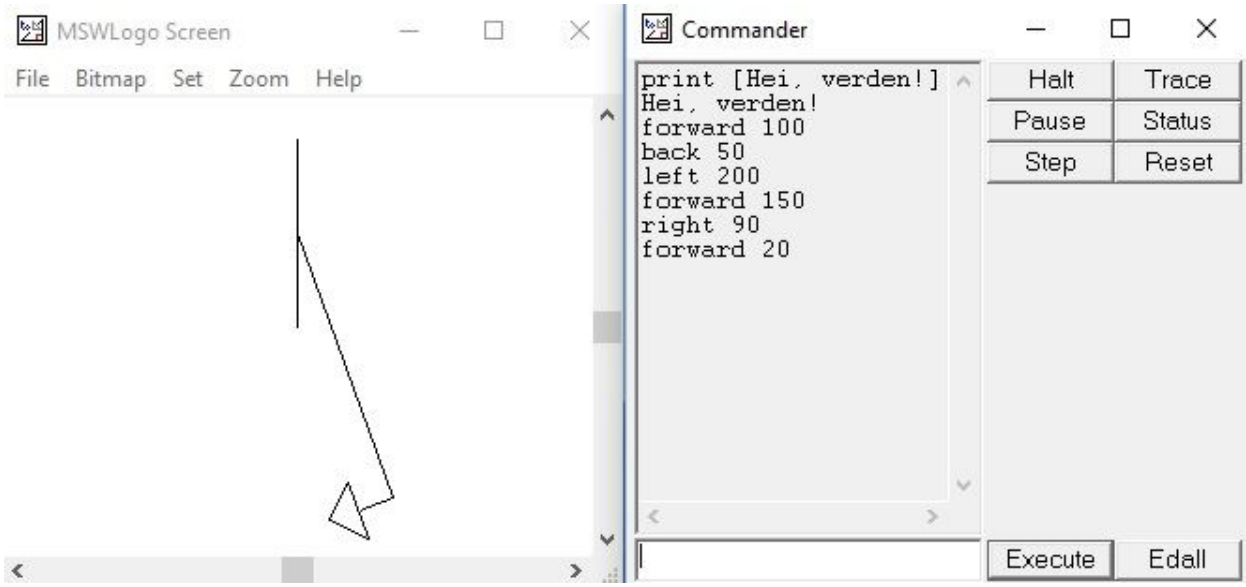
I følge MSWLogos nedlastningsside er den store fordelen med Logo at det er et språk som er lett å lære seg, selv for barn og voksne som ikke kan programmere fra tidligere (Mead, s.a.). Som man kan se av figuren nedenfor, kommer programmet med et enkelt grensesnitt. Ulempen med Logo er at brukbarheten er begrenset, det er lett å tegne trekanter og firkanter med det, men utover dette virker ikke Logo ut til å være brukbart til så mye annet. Opprinnelig var Logo laget for å styre en robot på gulvet og ikke en "skilpadde" på skjermen.

Noe av det nærmeste man kan komme til opprinnelig Logo, er et program som heter MSWLogo.

Figur X nedenfor viser eksempelkode i Logo, eller nærmere bestemt MSWLogo. Logo-koden skrives i vinduet til høyre, mens utførelsen (hvis det ikke er utskrift) vises i vinduet til venstre. Strekene viser hvor figuren har beveget seg. Merk også at man bruker [og] istedenfor vanlige parenteser.

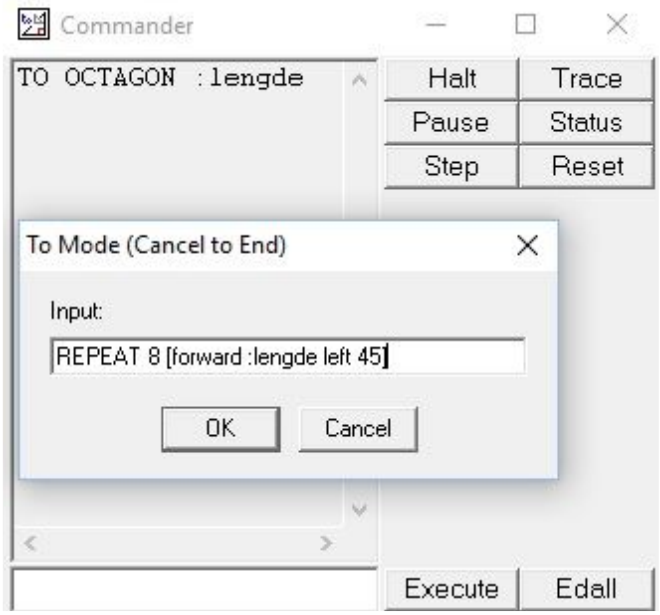
Programmet er tilgjengelig for Windows, men kan også kjøres i bl.a. Mac, Linux og BSD gjennom emuleringsprogrammet Wine.

Programmet starter opp med en trekantlignende figur på en hvit bakgrunn, samt et konsollignende vindu, kalt Commander. Det er i dette vinduet man skriver Logo-kode. I eksempelet under skrives “Hei, verden!” ut. Så går figuren 100 piksler fram, deretter 50 piksler bakover, for så å rotere 200 grader mot venstre, gå 150 piksler fram, rotere 90 grader mot høyre og til slutt gå 20 piksler fram.



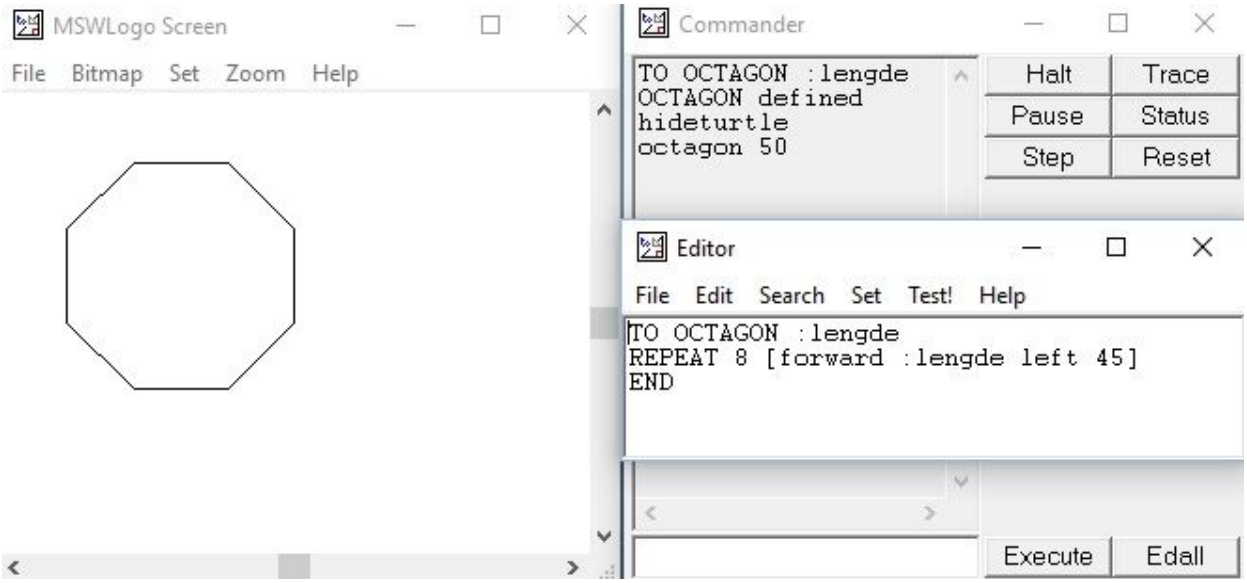
Figur X: Dette er det nærmeste man kommer til opprinnelig Logo. Programmet som denne koden er skrevet i, heter MSWLogo.

Figur X viser hvordan man lager en funksjon i MSWLogo. Når man klikker Execute etter å ha definert tittelen på en funksjon, i dette tilfellet “Octagon” (gresk for “åttekant”), kommer det opp et nytt vindu med en tekstboks. I denne tekstboksen skriver man inn koden som funksjonen skal gjennomføre, én linje om gangen. Når funksjonen skal avsluttes skriver man END i tekstboksen. “REPEAT” tilsvarer en while- eller for-løkke i språk som C/C++, Processing og Java.



Figur X: Denne figuren viser innskriving av kode som skal inn i funksjonen Octagon.

For å vise koden som er skrevet i funksjonen OCTAGON, må man klikke på knappen "Edall". Da dukker det opp et nytt vindu ("Editor") som viser koden. Man kan da endre på koden dersom det er nødvendig. For å kalle på funksjonen skriver man inn navnet på funksjonen og verdier til eventuelle parametere, i dette tilfellet lengden til hver sidekant. Dette illustreres i figur X. Denne figuren viser også kjøring av programmet. Ved å skrive "hideturtle" blir skilpadden skjult.

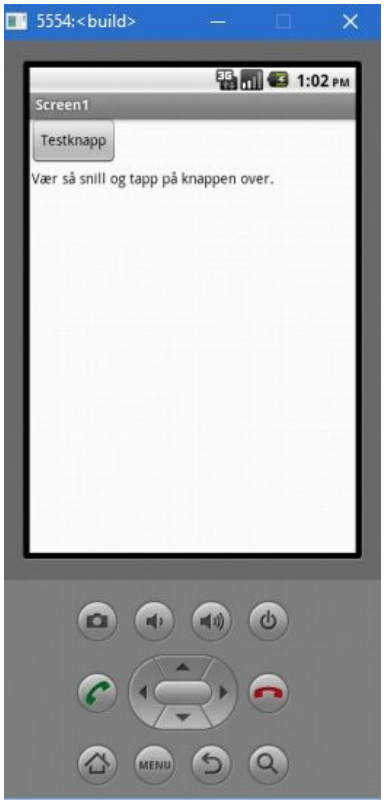


Figur X: Kjøring av programmet som ble opprettet i figur X samt illustrering av hvordan man viser koden som er skrevet inn i funksjonen octagon.

6.2.5 MIT App Inventor

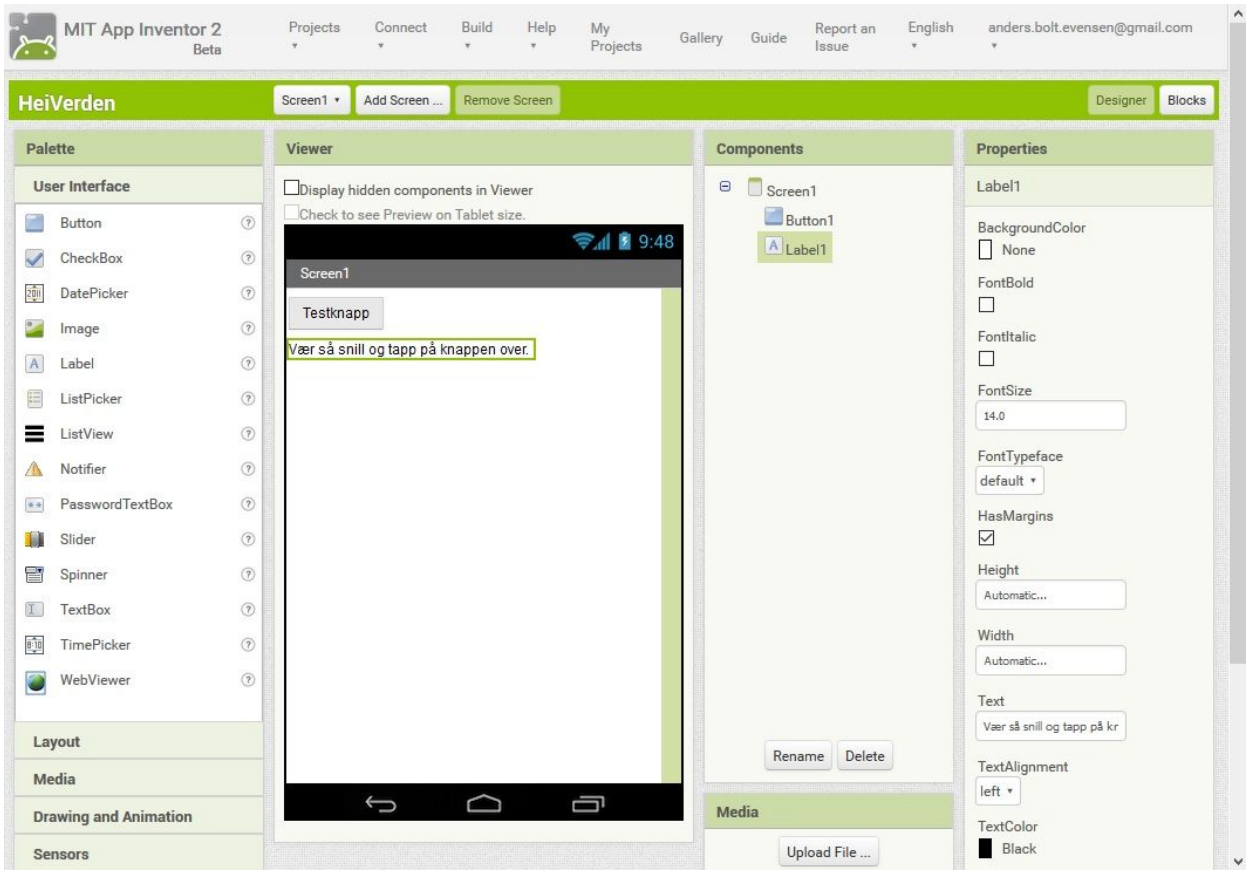
MIT App Inventor er et system som er utviklet av MIT (Massachusetts Institute of Technology) hvor man lager kjørbare apper for Android. Hvis man ikke har en Android-telefon, kan man kjøre emuleringen i en nettleser (Massachusetts Institute of Technology, s.a.). Dette er et verktøy som ligner mye på Scratch, og som kommer fra det samme universitetet. Da MIT App Inventor ble testet, var det metoden med emulering i nettleseren Firefox som ble brukt.

For å kunne starte opp hele prosessen med denne metoden, må man installere programmet aiStarter fra MIT. Dette er et kommandolinjeverktøy man trenger når man i steget etter oppretter et nytt prosjekt online. Grunnen til det er at man "emulerer" en Android telefon gjennom dette programmet. Deretter starter man dette programmet, går tilbake til nettleseren og enten lager et nytt prosjekt eller åpner et som allerede finnes fra før. Her ble det valgt å opprette et nytt prosjekt. Dette gjøres ved å klikke på “Projects” --> “Start new Project”, tilsvarende “Create” i Scratch. Så ble emulatoren koblet til ved å klikke på Connect --> Emulator. Da dukker det opp et nytt vindu hvor "telefonen" vises, se figur X. Det at man enten må installere et emuleringsverktøy eller ha en Android-basert enhet for å kunne bruke programmer som er laget i MIT App Inventor er den største av forskjellene mellom Scratch og MIT App Inventor.



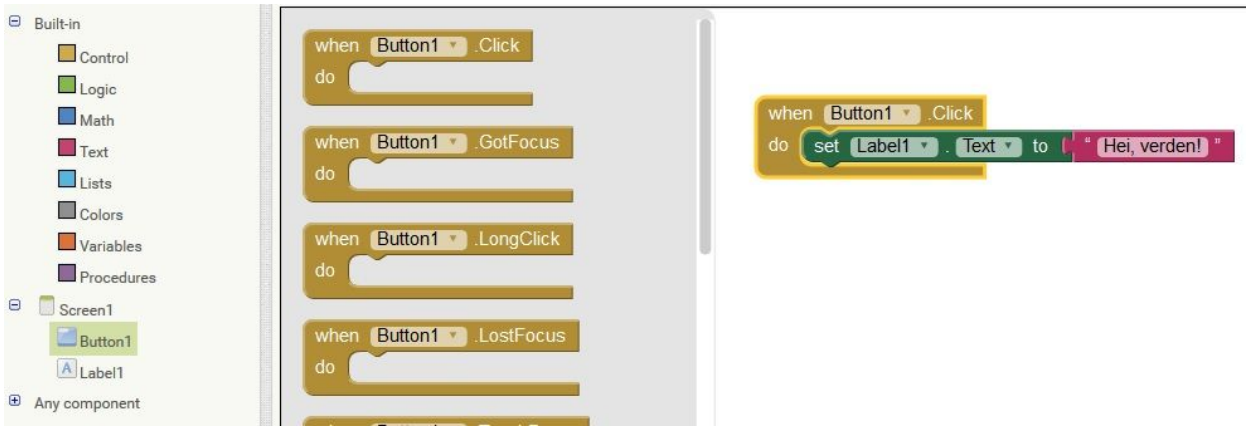
Figur X: "Telefonen" som emuleres

Selve MIT App Inventor-systemet består av 2 grensesnitt. Ett som tar for seg den grafiske delen av systemet hvor man blant annet finner knapper, tekstbokser, etiketter, og et "kodegrensesnitt" bestående av blokker al á de man finner i Scratch. Figur X viser hvordan det grafiske grensesnittet ser ut, mens figur X og X gir inntrykk av hvordan "kodegrensesnittet" ser ut. Her ble det tatt i bruk en knapp og en etikett. Endringer man gjør i prosjektet, synkroniseres automatisk til telefonen med én gang.



Figur X: Det "grafiske" grensesnittet i MIT App Inventor

Figur X viser hvordan man endrer teksten til en etikett når brukeren klikker/tapper på en knapp. I vinduet til venstre har man en oversikt over innebygde kategorier og i "vinduet" i midten har man en oversikt over ulike funksjoner avhengig av hva man har markert. Blokken med den funksjonen man vil bruke, drar man over i vinduet til høyre.



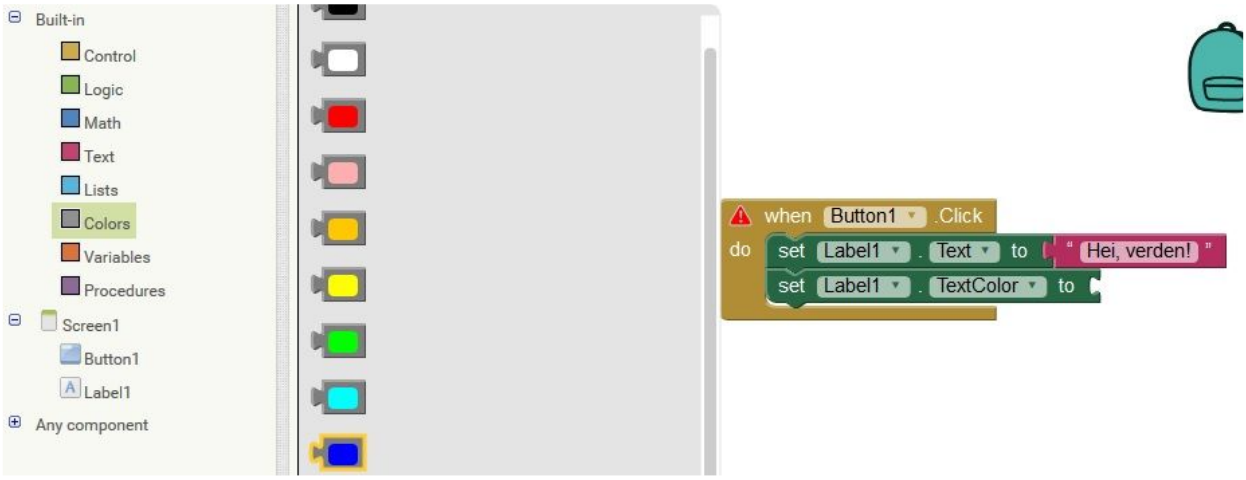
Figur X: Koden for å endre teksten til en etikett når en knapp er trykket.

Figur X viser hvordan man endrer fargen til en etikett når brukeren klikker/tapper på en knapp. Figuren demonstrerer også bruk av hvis-tester og variabler. Legg merke til den gule kanten rundt klossen med den blå fargen. Systemet sier også ifra når det er noe som mangler (se rødt utropstegn i den brune klossen).

Siden kodegrensesnittet i MIT App Inventor har et grafisk brukergrensesnitt som er mye likt Scratchs grensesnitt, er det naturlig å sammenligne dette verktøyet med Scratch. Som nevnt ovenfor, og som man ser i figur X på forrige side og figur X under, har man ulike kategorier som igjen består av ulike blokker, mye likt det grensesnittet man har i Scratch. Man finner blant annet klosser for kontrollfunksjoner under “Control”, logiske funksjoner, dvs, sann, usann, “ikke”, likhet, “and” og “or” under “Logic”, matematiske operasjoner under “Math” og tekstblokker under “Text”.

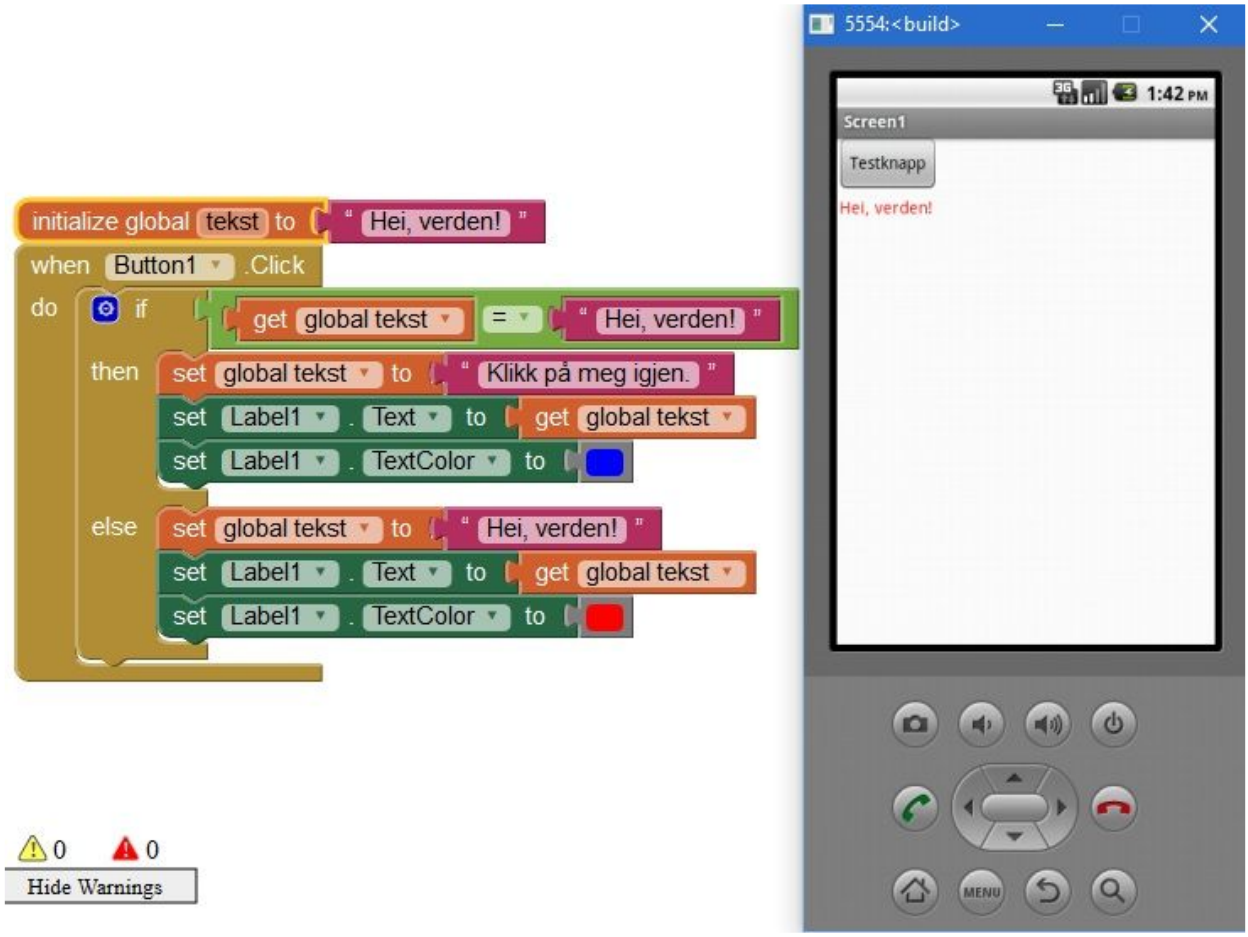
I figur X på neste side, er det opprettet en variabel kalt tekst.
Sammenlignet med Scratch er prosessen for å opprette en variabel noe annerledes og enklere enn det er i MIT App Inventor 2. I Scratch klikker man på “Data” og “Make a variable”. Da får man opp et vindu hvor man skriver navnet på variabelen. For å gi den variabelen en verdi, drar man ut blokken “Set variabelnavn to “ og skiver for eksempel 0 eller “Hei, verden”.

I MIT App Inventor klikker først på “Variables” for så å dra “Initialize global ‘name’” ut i det store, tomme feltet i det høyre “vinduet”. Først nå kan man gi den nye variabelen et navn som så kan brukes i programmet. Skal man gi den nye variabelen et heltall som verdi, klikker man på Math og kobler klossen med “0” innpå blokken med “Initialize global …” Vil man at variabelen skal ha en tekststreng som verdi, slik som i figur X, klikker man på Text og kobler blokken med “( )” til “Initalize global …”



Figur X: Her fullføres en kodelinje som endrer fargen til etiketten fra svart til blå (legg merke til den gule kanten på den blå klossen i det midterste vinduet).

Figur X viser hvordan eksempelprogrammet så ut til slutt. Programmet initialiserer en variabel med navn "name" og verdien "Hei, verden". Når brukeren klikker på knappen, kjøres en test for å se om verdien til variabelen stemmer mot teksten "Hei, verden!". Dersom det stemmer, endres teksten på etiketten til "Klikk på meg igjen" og fargen endres til blått. Hvis ikke, endres teksten til "Hei, verden!" og fargen blir rød isteden.



Figur X: Kjøring og visning av eksempelprogrammet.

Hvor stort utbytte vil elevene få med å bruke dette verktøyet?

Elever som ikke før har programmert vil ha ganske stort utbytte av dette, siden man kan leke seg og eksperimentere uten å risikere å ødelegge noe. Et annet stort pluss her, er at tekstbasert koding er byttet ut med koding med blokker, slik teamet bak Scratch også gjorde. I så måte er læringskurven heller ikke så bratt som den kan være i andre

programmeringsspråk. Rent matematisk vil MIT App Inventor kunne fungere like godt som det Scratch gjør.

Vår konklusjon er at dette er et system som er godt egnet for bruk i barneskolen, og som er et godt alternativ til Scratch, da mye av det som kan gjøres i Scratch også kan gjøres i MIT App Inventor. Dette gjelder spesielt dersom elevene har en ekte Android-basert telefon man kan bruke, samtidig som emuleringsmetoden også fungerer helt fint.

6.2.6 Python

Python er et programmeringsspråk med en enkel syntaks og som, i følge Orsini (2014), har en relativt jevn læringskurve. I så måte virker Python ut til å være et godt alternativ for de som skal lære seg programmering for første gang. Python er, i følge Levin (2011) et språk som er “kraftig” nok til at man kan jobbe profesjonelt med dette programmeringsspråket på heltid, men det er også lett nok til at folk som ikke har drevet med programmering raskt kan bli ganske gode. I tillegg består Python av mange tilleggsbiblioteker, noe som gjør at Python kan brukes til mye mer enn basiske ting som regning og utskrift.

I tillegg til den enkle syntaksen har Python også den fordelen at den kan kjøres på mange systemer. Det er installert som standard i Mac og de vanligste Linux-distribusjoner, samtidig som det kan installeres på Windows og andre systemer som for eksempel FreeBSD, NetBSD og Solaris. Python er et skriptspråk, hvilket betyr at man ikke trenger å kompilere koden for at den skal bli kjørbar.

Python er et språk som har mulighet til å koble til mange biblioteker. I tillegg til dette brukes Python også i forbindelse med maskinlæring og webutvikling, ved hjelp av rammeverket Django.

I kapittel 4.1 nevnes Python som et alternativ til Processing i den delen av matematikken som har med geometri å gjøre. Legger man til et bibliotek som heter pygame, kan man lage grafiske figurer og vise dette i et eget vindu, akkurat som i Processing. Python er også sammenlignbart med Processing i det at, som figur X under viser, at man kan jobbe med geometriske figurer, samt at det har det tradisjonelle tekstredigeringsgrensesnittet og ikke et grafisk grensesnitt, slik Scratch, MIT App Inventor og Tynker har. I motsetning til Processing trenger man ikke i Python å angi

hvilken datatype en variabel skal ha. Det avgjøres ved å angi en verdi til den aktuelle variabelen.

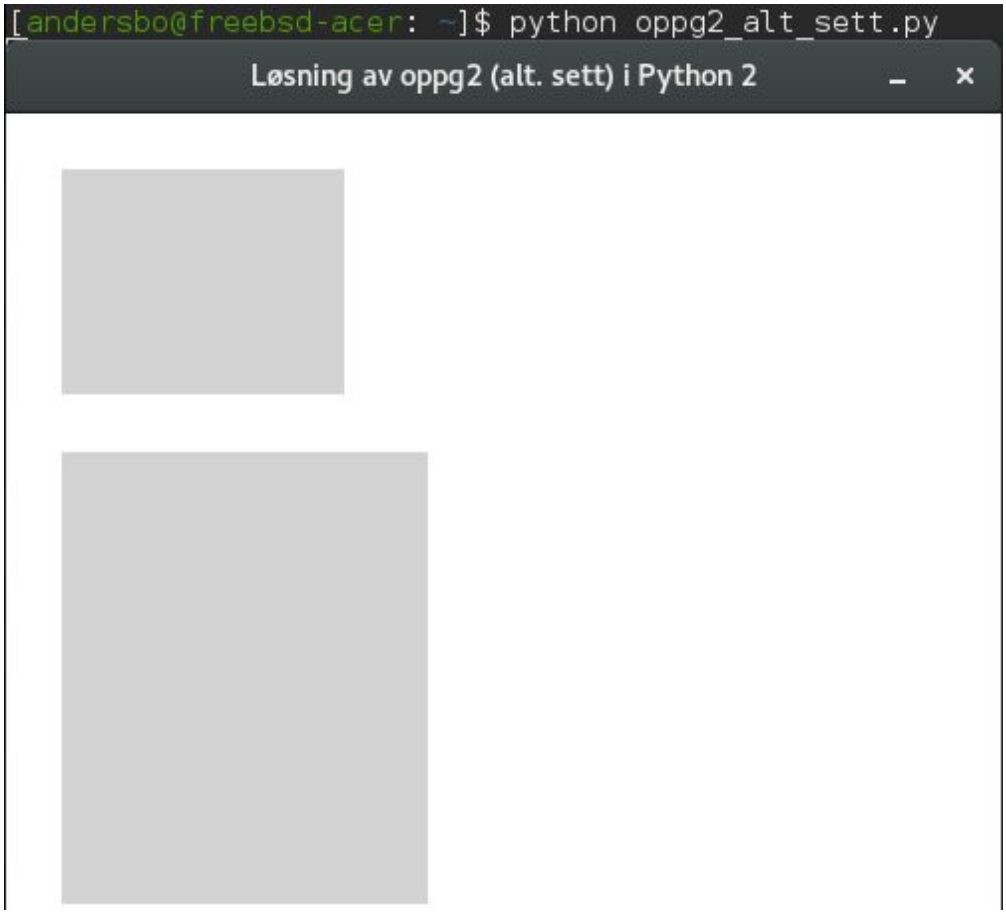
Kode X nedenfor viser en mulig løsning av oppgave 2 i det alternative oppgavesettet skrevet i Python (Oppgave 2 (Strupe - alternativt oppgavesett) - Tegne to rektangler). Variabelen factor brukes til skalering slik at man kan få omtrentlig centimetermål hvis man hadde skrevet bildet ut på papir. Løkka som går over de siste 4 linjene gjør at programmet ikke lukker seg automatisk etter noen sekunder.

```
#coding=UTF-8
import pygame
import sys

pygame.init()
factor=28.4
background_color=(255,255,255)
screen = pygame.display.set_mode((500,400))
screen.fill(background_color)
pygame.draw.rect(screen, (211,211,211),(1*factor,1*factor,5*factor,4*factor),0)
pygame.draw.rect(screen, (211,211,211), (1*factor, 6*factor, 6.5*factor, 8*factor),0)
tittel="Løsning av oppg2 (alt. sett) i Python 2"
pygame.display.set_caption(tittel)
pygame.display.update()
kjorer = True
while kjorer:
        for hendelse in pygame.event.get():
                if hendelse.type == pygame.QUIT:
                        kjorer = False
```

Kode X: Mulig løsning av oppgave 2 i det alternative oppgavesettet i Python

Figur X på neste side viser kjøring av koden ovenfor, i Python versjon 2:



Figur X: Kjøring av Python-koden som står på forrige side.

6.2.7 Tynker

Tynker er et verktøy som minner om Scratch. I følge utvikleren, Neuron Fuel, kan man lage apper og spill (Neuron Fuel, 2016). Verktøyet har etter vår mening en godt utviklet dokumentasjon som man finner ved å søke i hjelpemenyen i verktøyet.

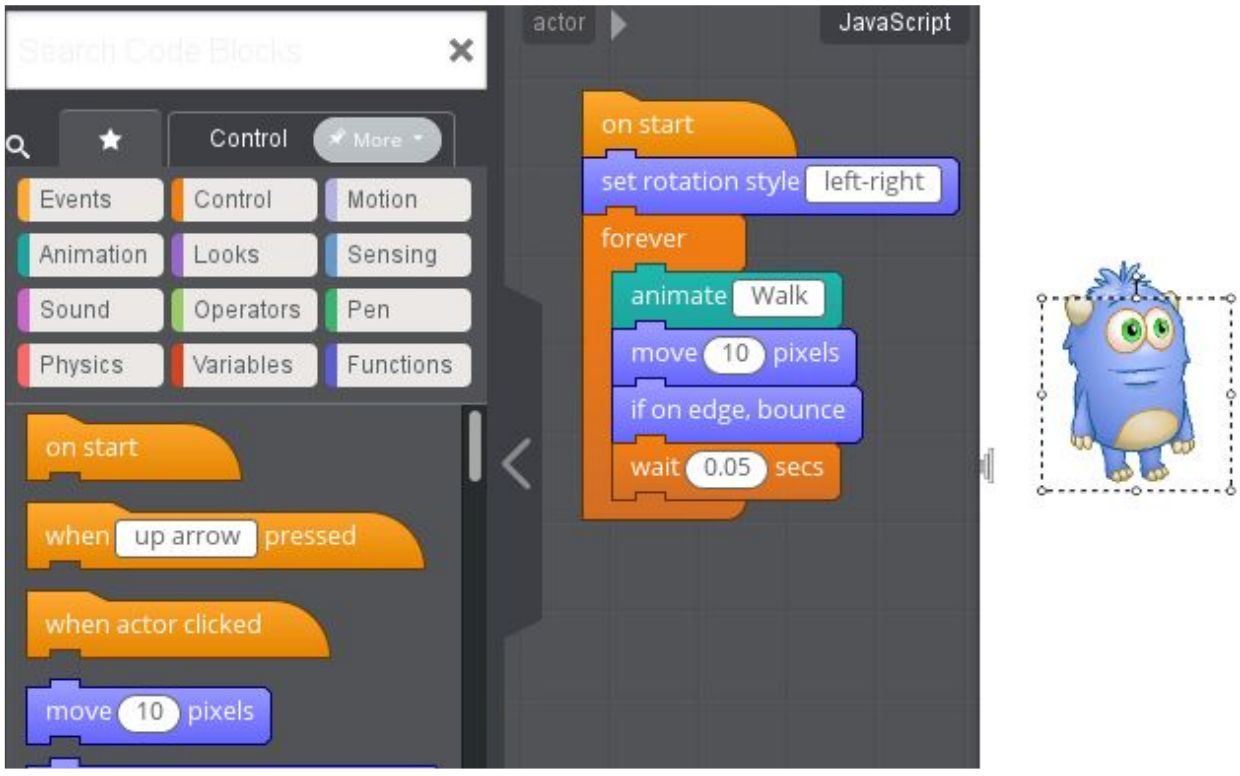
Dette er et program med et grensesnitt som minner veldig mye om brukergrensesnittet til Scratch (se avsnitt 6.2.1). Se figur X for en illustrasjon om hvordan grensesnittet ser ut. Man oppretter et nytt prosjekt på samme måten som man gjør i Scratch. Merk at når man lager et tomt prosjekt i Tynker, dukker det opp en automatisk generert kode (se figur X). Dette er en forskjell fra Scratch, hvor man hele tiden starter med et blankt prosjekt.

Lengst til venstre er det et "vindu" som består av en tekstboks hvor man kan søke etter gitte bokser. Under denne tekstboksen finner man ulike knapper med kategorier på, for

eksempel "Events", "Control", "Motion", "Animation" og "Sensing". Man kan til sammen velge blant 12 kategorier som hver har sin unike farge. For eksempel har "Variables" rød farge, "Operations" en lys grønn farge og "Sound" en rosa farge. Nedenfor knappene med de 12 kategoriene, har man kodesnuttene som dukker opp, avhengig av hvilken kategori man velger. Dersom man ikke har valgt kategori og heller klikker på fanen med en stjerne, dukker de vanligste blokkene opp. Se figur X for en illustrasjon.

"Vinduet" i midten består av koden som brukeren lager. Man lager koden ved å dra ulike blokker fra "vinduet" til venstre inn i det tomme, grå vinduet i midten og koble de ulike blokkene sammen. Øverst til høyre i dette "vinduet" er det en knapp/lenke som gir mulighet til å se koden i JavaScript.

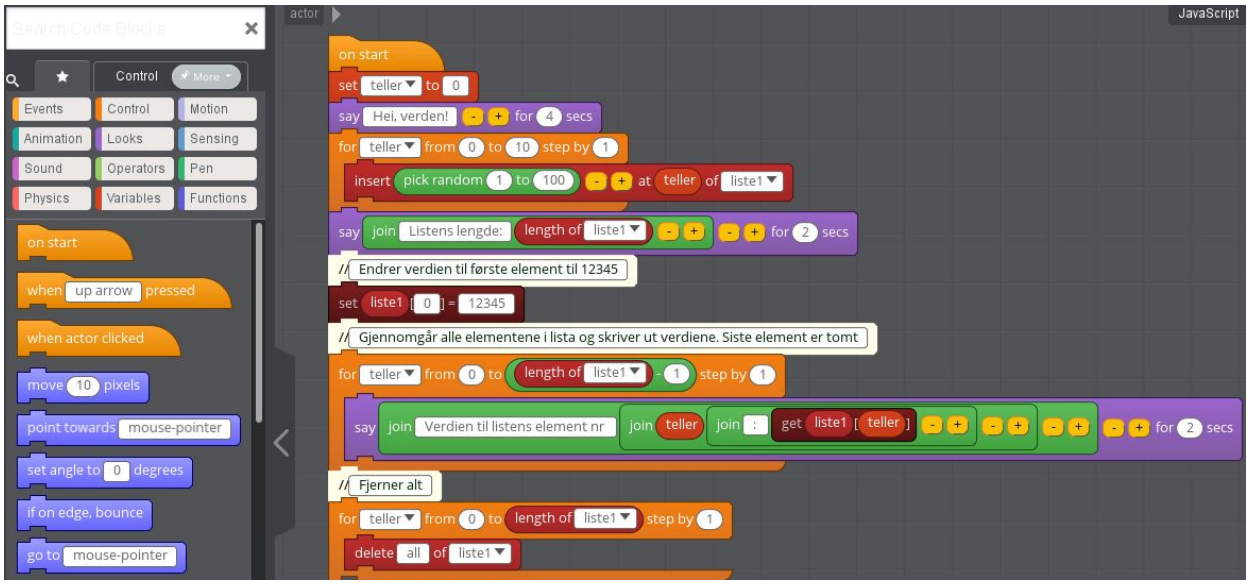
Vinduet til høyre består av en oversikt over scener og aktører (nederst) og et "kjøringsvindu" som viser kjøring av programmet (øverst).
Over disse 3 vinduene er det en meny, mulighet til å endre tittelen til prosjektet, samt mulighet til å bytte fra "Code" til "Level editor".



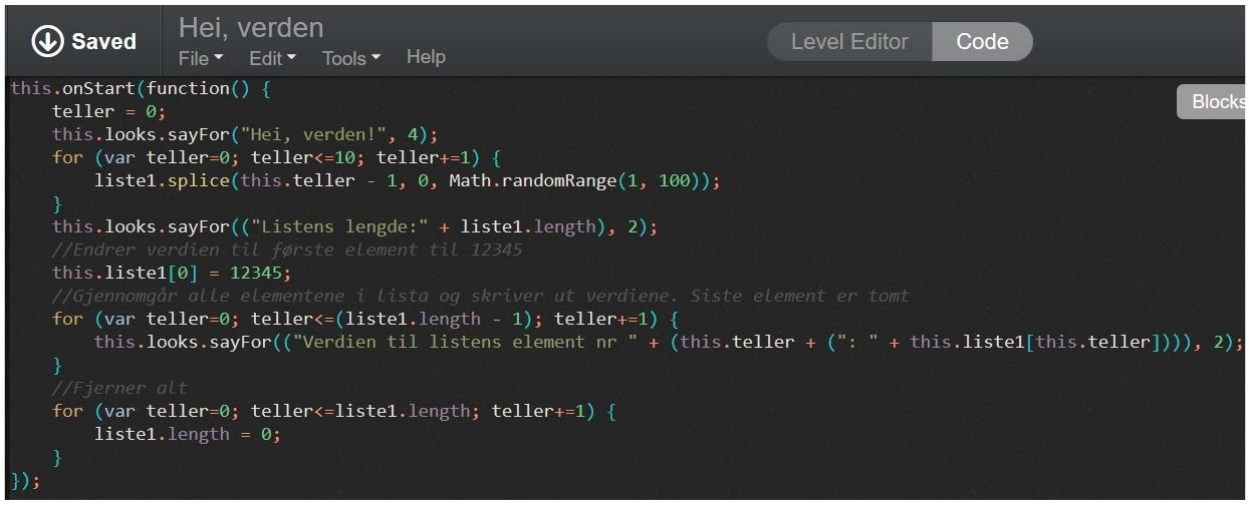
Figur X: Slik er grensesnittet i Tynker. Koden du ser i vinduet i midten er automatisk generert og dermed ikke min.

Er man ikke fornøyd med koden man har laget, markerer man blokkene man vil fjerne. Da dukker det opp en søppelkasse over "vinduet" til venstre. Drar man de valgte blokkene dit, slettes disse. Man kan opprette egne variabler der hvor det er nødvendig. Variablene fungerer på samme måte som i Scratch med at man ikke trenger å oppgi datatype når man lager en variabel. I tillegg til vanlige variabler kan man i Tynker også jobbe med lister og jobbe med objektoperatorer mot disse listene. Objekter diskuteres i avsnitt 6.2.2.3.

Figur X viser det klassiske "Hei, verden"-eksempelet samt bruk av ei liste og bruk av både liste- og objektoperasjoner. Figur X viser den tilsvarende koden i JavaScript.



Figur X: Et program som demonstrerer bruk av liste og liste- og objektoperatorer, i tillegg til det vanlige Hei, verden.



Figur X: Den tilsvarende koden i JavaScript

Hvor stort utbytte får elever av dette og hvordan er læringskurven?

Tynker faller i samme båt som Scratch, siden det er begrenset hva man kan jobbe med innenfor matematikken og at takhøyden ikke er så stor. Verktøyet fungerer veldig bra til å lage "kalkulator"- og konverteringsoppgaver. Når det er sagt, så virker Tynker til å være hakket med avansert enn det Scratch er, siden man i Tynker også jobber med tabeller/lister. Likevel har Tynker har omtrentlig samme læringskurve som det Scratch har. Kan man tankegangen i Scratch, kan man også veldig mye av tankegangen i Tynker. Derfor mener han at Tynker vil egne seg godt på barneskolenivå.

6.3 Programmeringsverktøy som også ble vurdert

Under skrivingen av denne bacheloroppgaven ble det vurdert flere andre programmeringsverktøy i tillegg til dem som er nevnt ovenfor. Disse verktøyene fungerte som spill som skulle lære barn å programmere. Disse spillene gikk ut på å kontrollere en karakter slik at den gjør en gitt handling eller kommer seg fra A til B for å komme til neste nivå. Ett spill brukte JavaScript og tekstbaserte funksjoner, mens de andre spillene brukte en mer Scratch-lignende tilnærming. Sammenlignet med verktøyene som allerede er diskutert, førte de utelatte verktøyene lite nytt med seg, samtidig som at bruksområdene til disse verktøyene var mer begrenset. De ble derfor fjernet fra evauleringen.

7 Konklusjon

Å demonstrere kode live er en effektiv måte å introdusere grunnskoleelever til programmering på. Spesielt er det fordelaktig dersom elevene selv samtidig skriver ned koden som blir gjennomgått. Dette lar elevene få syntaksen inn i fingrene på et tidlig stadium, i tillegg til å åpne muligheten for elevene til å eksperimentere og prøve seg fram underveis.

Scratch har vist seg å være et godt verktøy for å lære barn å programmere. Det gjør programmering lett tilgjengelig for elever ved 7. trinn, og oppleves som gøy å jobbe med, noe som kan føre til økt motivasjon i matematikkundervisningen.

Processing har vist seg å være egnet for bruk i ungdomsskolen. Grunnleggende konsepter innen programmering og Processing er intuitivt å forstå for elever ved 9. trinn.

God bruk av variabler er imidlertid noe som bør introduseres nøyere enn hva som ble gjort i dette prosjektet.

Elever ved 7. trinn kan forventes å beherske evnen til å lese og forstå kildekode til programmer, samt evnen til å gjøre endringer i eksisterende programmer for å oppnå gitte mål. Kun et fåtall av elever ved 7. trinn kan forventes å beherske evnen til å lage programmer fra bunn av, etter et såpas kort undervisingsopplegg.

Elever ved 9. trinn kan etter kort tid forventes å beherske evnen til å skrive egne Processing-programmer fra bunnen av. Dette inkluderer evnen til å tegne todimensjonale figurer, og å utføre enkle beregninger rundt disse. Noen elever kan også forventes å mestre bruk av programmering for å løse oppgaver om formelsnu, samt mer avansert tegning, gjennom bruk av linjer og rotasjon.

I hvilken grad bruk av programmering bidrar til økt matematikkforståelse er for tidlig å konkludere. For å kunne sammenligne investeringen opp mot nytteverdien, behøves et klarere bilde av den potensielle nytteverdien som kan oppnåes. Dette er imidlertid ikke noe man vil komme i nærheten av med et såpass kort og begrenset undervisnings-opplegg. Dette vil være opp til videre prosjekter å se videre på.

8 Referanser

Alan Kay: Doing with Images Makes Symbols Pt 2 (1987). [Videoklipp]. Hentet 16.05.2016 fra https://archive.org/details/AlanKeyD1987_2

All You Need is Code. (s.a). *About*. Hentet 26.04.2016 fra http://www.allyouneediscode.eu/about

AVG Digital Diaries. (2015). *Digital skills*. Hentet 16.05.2016 fra http://www.avg.com/digitaldiaries/2010

Bjørndal, C.R.P. (2011). *Det vurderende øyet.* Oslo: Gyldendal akademisk

Dahlum, S. (2014). Kvantitativ analyse. *Store Norske Leksikon.* Hentet 16.05.2016 fra https://snl.no/kvantitativ_analyse

European Schoolnet. (2015). *Computing our future: Computer programming and coding - Priorities, school curricula and initiatives across Europe* (2015 update). Hentet 17.05.2016 fra http://www.eun.org/publications/detail?publicationID=661

Halle, N. H. (2014). Hawthorneeffekten. *Store Norske Leksikon.* Hentet 16.05.2016 fra https://snl.no/Hawthorneeffekten

Hofseth, A. (2013, 07.03). Programmering i norske skoler? *NRKBeta*. Hentet 16.05.2016 fra https://nrkbeta.no/2013/03/07/programmering-i-norske-skoler/

Klovning, E. (2015, 04.06). – Programmering bør være allmennkunnskap. *Dagens Næringsliv*. Hentet 16.05.2016 fra http://www.dn.no/talent/2015/06/04/1944/Utdanning/-programmering-br-vre-allmennkunnskap

Levin, M. (2011, 05.01). Python Programming Language Advantages & Disadvantages. Hentet 10.03.2016, fra http://mikelev.in/2011/01/python-programming-language-advantages

Logo Foundation. (2015). *Logo History*. Hentet 10.03.2016 fra http://el.media.mit.edu/logo-foundation/what_is_logo/history.html

Lær Kidsa Koding. (2016). *Kodeklubboversikt.* Hentet 02.05.2016 fra http://kidsakoder.no/kodeklubben/kodeklubboversikt/

Massachusetts Institute of Technology. (s.a.). *Setting Up App Inventor.* Hentet 29.04.2016 fra http://appinventor.mit.edu/explore/ai2/setup.html

Mead, N. (s.a.). *Very simple Logo programming environment.* Hentet 10.03.2016 fra http://mswlogo.en.softonic.com/

Neuron Fuel. (2016). *Try an hour of code*. Hentet 10.03.2016 fra https://www.tynker.com/hour-of-code/

Orsini, L. (2014, 08.07). Why Python Makes a Great First Programming Language. *ReadWrite*. Hentet 30.03.2016 fra http://readwrite.com/2014/07/08/what-makes-python-easy-to-learn/

Partovi, H. (2014). *The Hour of Code is here! President Obama wrote his first line of code*. Hentet 26.04.2016 fra http://blog.code.org/post/104684466538/hourofcode2014

Retteberg, J. W. (2013, 25.03). Hvorfor lærer vi ikke barna å kode? *Aftenposten*. Hentet 16.05.2016 fra http://www.aftenposten.no/meninger/kronikker/Hvorfor-larer-vi-ikke-barna-vare-a-kode-7157804.html

Senter for IKT i utdanningen. (2015). *Hensiktsmessig bruk av IKT i undervisningen - en veileder*. Hentet 02.04.2016 fra https://iktsenteret.no/sites/iktsenteret.no/files/attachments/veileder_hensiktsmessig_bruk_bm_lav.pdf

Sevik, K. (2015). *Koding i skolen*. Hentet 02.05.2016 fra https://iktsenteret.no/sites/iktsenteret.no/files/attachments/koding_i_skolen_-_sikt-notat_nr_2_-_temakonferanse_2015.pdf

Trætteberg, H. (2014). *Objekter og klasser*. Hentet 14.04.2016 fra https://www.ntnu.no/wiki/display/tdt4100/Objekter+og+klasser

Word Wide Web Consortium. (2012). *A Short History of JavaScript.* Hentet 10.03.2016 fra https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript

Vedlegg

Vedlegg 1 Oppgavesett Scratch
Vedlegg 2 Oppgavesett Processing

Vedlegg

1 Oppgaveindeks

Denne oppgaveindeksen er en liste over alle oppgavene brukt under uttestingene ved Os og Strupe. Det ordinære oppgavesettet for Strupe ble produsert av Børre Stenseth. Resten av oppgavene ble produsert av bachelorgruppen.

For fulle oppgavetekster, se vedlegg 2, *Oppgavesett, Os* og vedlegg 3, *Oppgavesett, Strupe*.

1.1 Demoer:

Demo 1: https://scratch.mit.edu/projects/97085130/
Demo 2: https://scratch.mit.edu/projects/97085180/
Demo 3: https://scratch.mit.edu/projects/97085263/
Demo 4: https://scratch.mit.edu/projects/97085361/

Demo-programmene ble demonstrert live for klassen. Her ble det tatt imot innspill fra klassen om hvordan programmene skulle fullføres.

1.2 Øvinger:

Øving 1: https://scratch.mit.edu/projects/97150654/
Øving 2: https://scratch.mit.edu/projects/97151479/
Øving 3: https://scratch.mit.edu/projects/97152521/
Øving 4: https://scratch.mit.edu/projects/97153504/

Øvingene skulle løses på egenhånd av elevene. Hver øving hadde problemstillinger som reflekterte demo-oppgavene som ble demonstrert i forveien.

1.3 Oppgavesett, Os:

Oppgave 1: https://scratch.mit.edu/projects/95781503/
Oppgave 2: https://scratch.mit.edu/projects/95778540/
Oppgave 3: https://scratch.mit.edu/projects/97050713/
Oppgave 4: https://scratch.mit.edu/projects/97054015/
Oppgave 5: https://scratch.mit.edu/projects/97055019/

Oppgave 6: https://scratch.mit.edu/projects/97061994/

Fasit oppgave 4: https://scratch.mit.edu/projects/97055019/
Fasit oppgave 5: https://scratch.mit.edu/projects/97059656/
Fasit oppgave 6: https://scratch.mit.edu/projects/97061054/

1.4 Ubrukte oppgaver:

Prosessen for utvikling av undervisningsmateriale var tidkrevende, og gruppen endte opp med å lage prototyper for mange oppgaver som aldri ble videreutviklet, eller brukt i undervisningen. Hvorfor enkelte oppgaver ikke ble brukt blir diskutert i diskusjonskapittelet for Os.

34 ubrukte oppgaver er samlet i denne linken:

https://scratch.mit.edu/users/andersbo/projects/

2 Oppgavesett, Os

Oppgave 1:

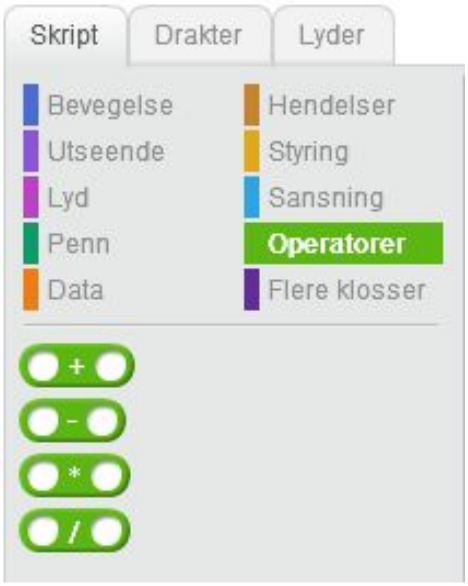
Gå til linken: https://scratch.mit.edu/projects/95781503/

Hesten Harald er veldig god til å gjøre om fra kilogram til hektogram. Trykk på det grønne flagget for å kjøre programmet. Velg antall kilogram du vil at hesten Harald skal gjøre om til hektogram.

a. Svaret du fikk var 10 ganger større enn tallet du skrev i kilo. Kan du forklare hvorfor?

Harald trenger hjelp til å gjøre om fra kilogram til gram. Trykk på “Se Inni” for å se hvordan programmet er bygget opp. Bruk dette til å svare på spørsmålene nedenfor.

a. Gjør endringer i programmet slik at Harald gjør om til gram i stedet for hektogram.
b. Kjør programmet på nytt, og be Harald om å omgjøre 5 kilogram til gram. Hva ble svaret?
c. Gjør endringer i programmet slik at Harald gjør om fra gram til kilogram. Hint: Du må endre det som kalles operator. Dette finner du under Operator-menyen, se bildet nedenfor.



a. Kjør programmet på nytt, og be Harald om å omgjøre 1000 gram til kilogram. Hva ble svaret?

Oppgave 2:

Gå til linken: https://scratch.mit.edu/projects/95778540/

Trykk på det grønne flagget for å kjøre programmet. Bruk venstre og høyre piltast for å minke og øke verdiene som omgjøres.

a. Bruk programmet for å finne ut hva 8 kilogram tilsvarer i hektogram.

Trykk på "Se Inni" for å se hvordan programmet fungerer.

a. Hvordan regnet programmet seg fram til svaret i hektogram?
b. Hva forteller dette deg om forholdet mellom kilogram og hektogram?
c. Bruk programmet til å finne ut hvor mange kilometer som tilsvarer 25 meter.
d. Hvordan regnet programmet seg fram til svaret i meter?
e. Hva forteller dette deg om forholdet mellom kilometer og meter?

Oppgave 3:

Gå til linken: https://scratch.mit.edu/projects/97050713/

Programmet i denne oppgaven er veldig lik programmet i oppgave 2, bortsett fra at måleenhetene er forskjellig. Men den viktigste forskjellen er at dette programmet ikke er ferdig laget, og det er din oppgave å fullføre det.

I programmet fra oppgave 2 ble verdiene regnet ut med disse operasjonene:

Men operasjon-delen av dette programmet er ikke fylt inn!

Byggeklossene du trenger for å fullføre programmet ligger nederst i skriptet:

a. Bruk disse byggeklossene for å fullføret programmet, slik at verdiene omgjøres riktig.
b. Kjør programmet og kontrollér at verdiene blir omgjort.

Oppgave 4:

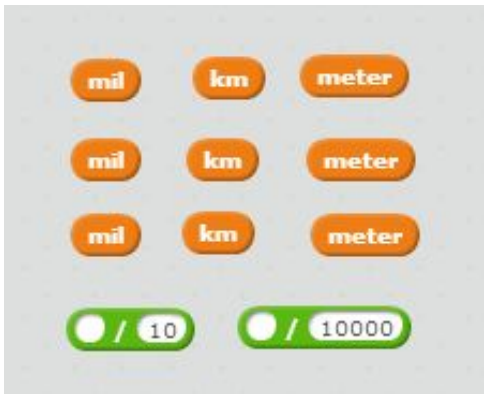
Gå til linken: https://scratch.mit.edu/projects/97054015/

Per går en tur hver dag. På mandag, tirsdag og onsdag gikk han disse avstandene:

Mandag: 2,5 kilometer.
Tirsdag: 950 meter.
Onsdag: 1,5 mil.

Programmet skal regne ut hvor langt Per har gått til sammen, og oppgi dette som meter, som kilometer, og som mil. Men programmet er ikke ferdig laget – klossene som skal legge sammen og omgjøre avstandene er ikke fullstendige:

Klossene som trengs for å fullføre programmet ligger løst i skriptet:



a. Fullfør programmet slik at det regner ut hvor langt Per gikk til sammen, uttrykket som både meter, kilometer og mil.
b. Hvor langt gikk Per til sammen? Oppgi svaret som meter, kilometer, og mil.

Oppgave 5:

Gå til linken: https://scratch.mit.edu/projects/97059689/

Per skal fylle en kjele med vann. Han fyller først på 250 ml, og deretter 7,2 dl.

a. Fullfør programmet slik at det regner ut hvor mange centiliter med vann Per har fylt i kjelen. I dette programmet blir du ikke forsynt med klossene du trenger – denne gangen må du hente disse selv fra menyene.

a. Hvor mange centiliter med vann hadde Per fylt?

Notat: Spent på om denne oppgaven blir for ambisiøs. Dette er den første oppgaven som ikke supplerer med bygge klossene som trengs. Tidligere oppgaver tok steget fra

analyse til endring i eksisterende programmer. Her går skrittet videre til å lage deler av programmet fra bunn av. Mulig det forventer for mye av elevene.

Oppgave 6:

Gå til linken: https://scratch.mit.edu/projects/97061994/

Vi har seks blokker:

Blokk 1 veier 250 g
Blokk 2 veier 4 hg
Blokk 3 veier 1,5 kg
Blokk 4 veier 0,8 kg
Blokk 5 veier 300 g
Blokk 6 veier 6 hg

Programmet skal regne ut hvor mange gram blokkene veier til sammen, men programmet er ikke ferdig laget. Klossene som må brukes for å fullføre programmet ligger løst i skriptet:

a. Gjør endringer i programmet slik at det regner ut hvor mange gram alle blokkene veier til sammen.

Gjør endringer i programmet slik at det regner ut hvor mange hektogram blokk 1, 4 og 6 veier til sammen.

3 Oppgavesett, Strupe

Her er det to oppgavesett. Det første er det oppgavesettet som ble laget av bachelorgruppens veileder, Børre Stenseth. Oppgave 9 og 10 i dette settet ble utelatt da de omhandler trigonometri som ikke er en del av ungdomskolepensum, mens oppgave 11 var en slags ekstraoppgave.

Dette er et utvalg av oppgaver som Børre Stensett satte sammen. Dette er de oppgavene som Odd-Tore, lærerne og Børre ble enige om å gi til elevene.

I tillegg ble det laget et annet, alternativt oppgavesett. Tanken med dette var å gi elevene en mulighet til å repetere ting de kanskje syntes var vanskelige i det opprinnelige oppgavesettet, i tillegg til å la elevene få bryne seg på andre geometriske figurer. Dette oppgavesettet ble delt ut i løpet av den siste økta, og elevene kunne velge mellom å bruke det opprinnelige settet eller det alternative. Resultatene dreier seg først og fremst om det opprinnelige oppgavesettet, men siden det alternative også nenves, velger vi å ta med en kort oversikt over alle oppgavene i dette settet. Begge oppgavesettene er tilgjengelig i filen processing.rar som kan lastes ned her: http://frigg.hiof.no/bo16-g12/files/processingoppgaver.rar

3. 1 Ordinært oppgavesett (av Børre Stenseth) (neste side):

Programmeringsoppgaver

Noen oppgaver som skal løses ved hjelp av programmeringsverktøyet Processing.

Alle oppgavene går ut på å tegne figurer på et ark i A4-format.

Tema for oppgavene er areal og volum.
Oppgavene er beregnet på 9. og 11. skoletrinn.

Du løser en oppgave ved å åpne skissen (sketch) med riktig navn (oppgavens navn) i katalogen **svar**, og skiver inn din kode i **minkode.**

Processing finner du på https://processing.org/

Her finner du også en omfattende og god beskrivelse av alle mulighetene i Processing. Det kan være litt slitsomt å finne fram her i starten. Derfor er det laget en liste over de vanligste funksjonene du trenger i vedlegg til dette oppgavesettet.

Lykke til

1. Ruter

Tegn to kvadrater, ett på 3x3 cm og ett på 5x5 cm.
De to skal ligge under hverandre med 1 cm avstand.

Du finner en løsning på denne oppgaven i katalogen **svar/ruter**.
For å bli kjent med programmeringen kan du leke med denne løsningen (endre størrelsen på kvadratene, flytte dem, legge til ett kvadrat, lage rektangler i stedet, osv).

Du kan bruke denne løsningen som mal, utgangspunkt når du skal løse de andre oppgavene nedenfor.
Fordelene med det er at du får satt opp A4-siden og du har en mønster du kan skrive koden din i setup(), draw() osv

2. Italia

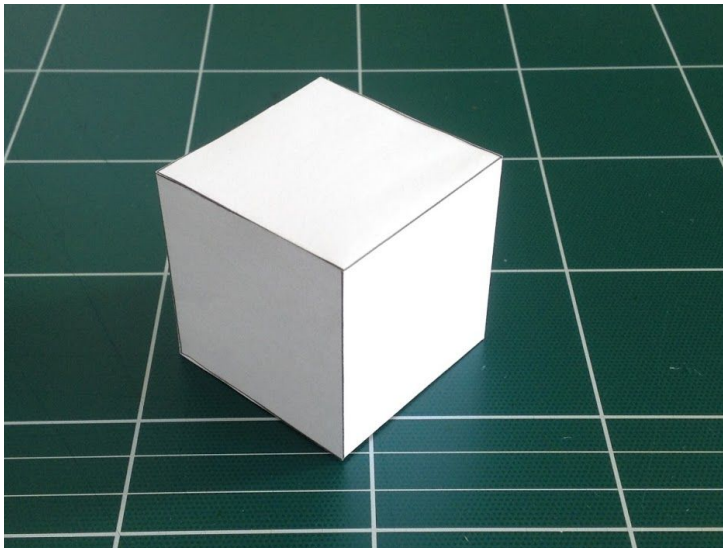
Litt oppvarming
Det italienske i figur X flagget ser slik ut:

Figur X: Det italienske flagget

og så står det i beskrivelsen på Wikipedia: Propotion 2:3. Det må vel bety at det er 2 enheter vertikalt og 3 enheter horisontalt.
Tegn flagget med en høyde på 4 cm.
Hint:
rødt: ***fill(255,0,0)***
*hvitt:****fill(255,255,255)***
grønt: ***fill(0,255,0)***

3. Terning1

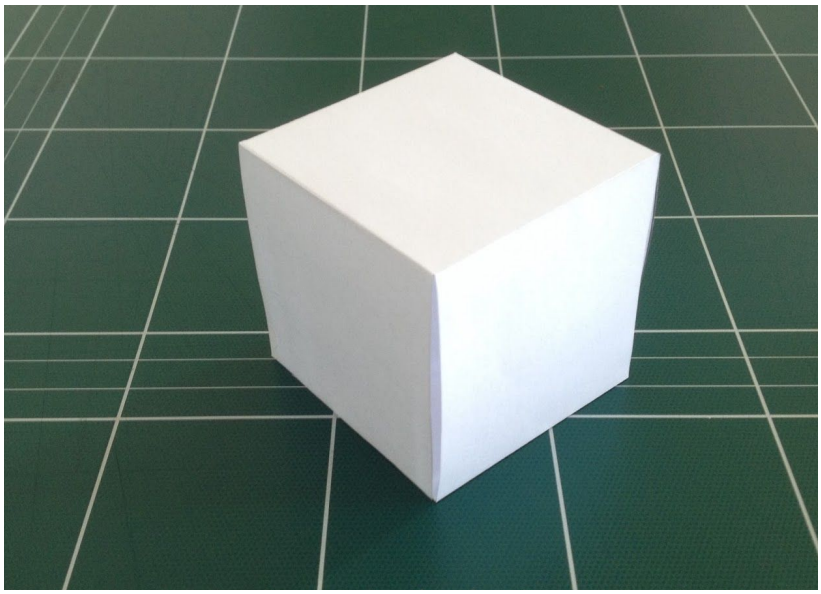
En terning har bredde, lengde og høyde 5cm.
Tegn terningen utbrettet slik at dersom vi klipper og bretter figuren til en 3-dimensjonal figur får vi en terning med sidekanter på 5 cm.



Figur X Endelig resultat Terning 1

4. Terning2

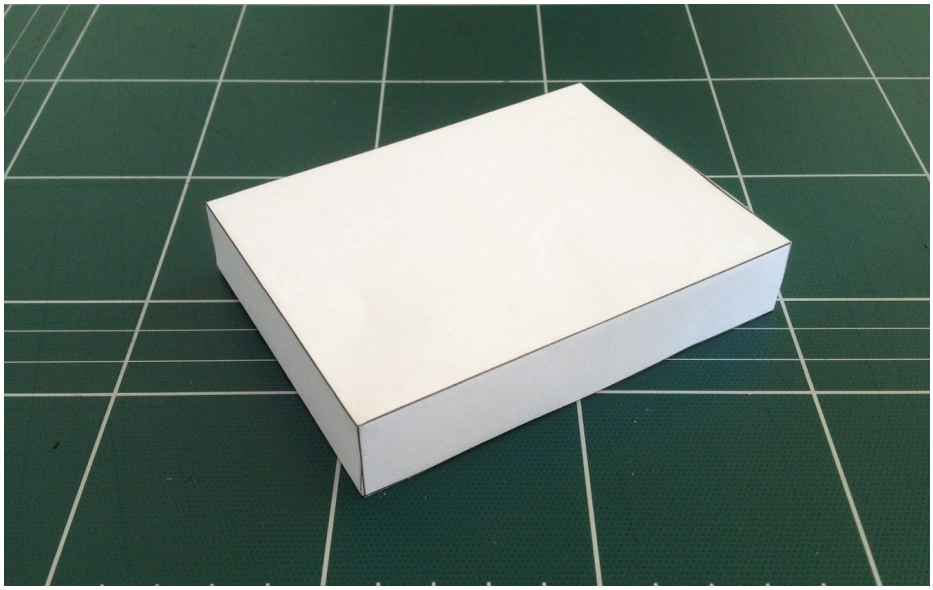
En terning har en samlet overflate på 220 kvadrat centimeter. Husk at alle sidekantene er like lange i en terning.
Tegn terningen utbrettet. Deretter skal du måle og se at det passer.
*Hint: Du kan finne kvadratroten av et tall med **sqrt(tallet**)*



Figur X Endelig resultat Terning 2

5. Eske1

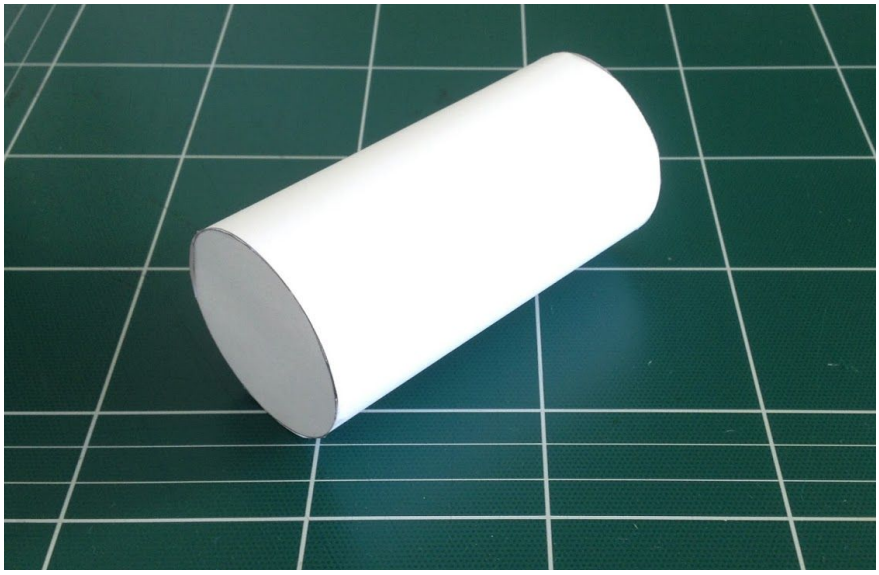
En eske har et gitt volum på 170 kubikk centimeter. Høyden er 2 cm og lengden er 8 cm.
Finn bredden og tegn esken utbrettet. Deretter skal du klippe og brette figuren til en 3-dimensjonal figur slik at du får en eske med riktige dimensjoner.



Figur X Endelig resultat Eske 1

6. Sylinder

En lukket sylinder (tettet i begge ender) har et gitt volum på 200 kubikk cm.
Bestem diameter og lengde og tegn sylinderen utbrettet. Deretter skal du klippe og sette sammen figuren til en sylinder. Mål så volum. Hvor nærme kom du det gitte volum?
Prøv en gang til slik at du kommer enda nærmere det gitte volum enn første gang.
Hint: Du kan tegne en sirkel med ***ellipse(x,y,d,d)*** *der sentrum er i x,y og diameteren er d.*



Figur X Endelig resultat Sylinder

7. Flagg

Tegn flere norske flagg i forskjellig størrelse på ett ark

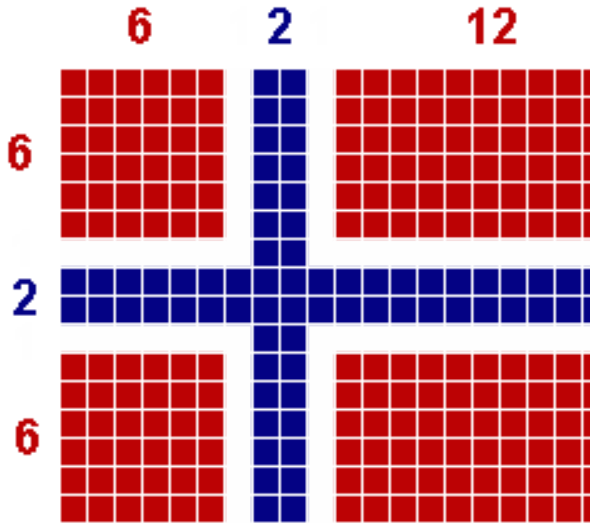
Det norske flagget (figur X) er bygget opp av enheter med forskjellig farge. Det er 22 enheter horisontalt og 16 enheter vertikalt.

Figuren til høyre viser dimensjonene. Du skal **ikke** tegne de små rutene hver for seg. Det skal være sammenhengende røde, hvite og blå områder slik som vi vanligvis ser flagget.

Hint:
Rødt: ***fill(255,0,0)***
Hvitt: ***fill(255)***
Blått: ***fill(0,0,255)***

Du kan lage en egen funksjon:
void tegnFlagg(float lengde){
}

Fiu



Figur X: Det norske flagget

som du kan bruke flere ganger.

Noen oppgaver for de som vil prøve seg med litt trigonometri

8. Trekanter

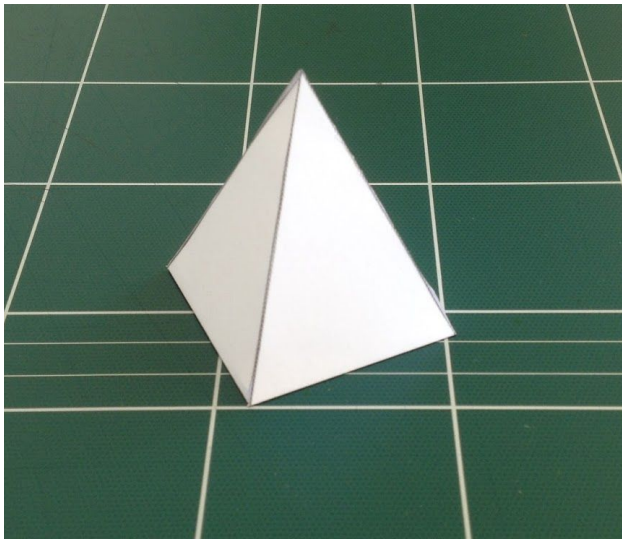
På samme ark skal du tegne

- En likesidet trekant, med sider 5cm
- En likebent trekant, med de to like sider på 5cm
- En trekant der alle sidene er forskjellig

Hint: Du kan tegne en trekant med punktene p1,p2,p3 slik:
triangle(p1x, p1y, p2x, p2y, p3x, p3y)
Du har tilgang til trigonometriske funksjoner som ***sin(v), cos(v), tan(v)*** *der v skal være oppgitt i radianer. Husk at 360 grader er 2PI (6.28) radianer.*

9. Pyramide1

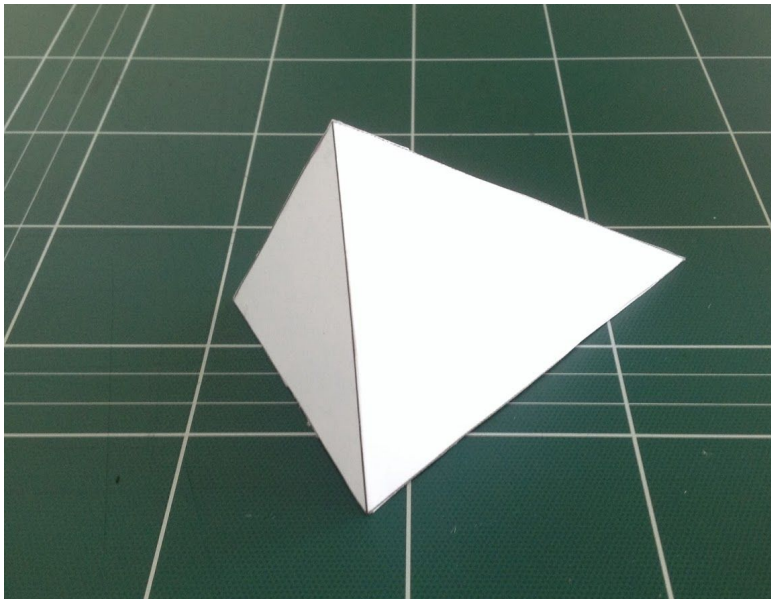
En pyramide har en kvadratisk grunnflate på 5x5 cm og høyde 6cm.
Tegn pyramiden utbrettet slik at dersom vi klipper og bretter figuren får vi en pyramide med riktige dimensjoner.
Hint: Du har tilgang på trigonometriske funksjoner, ***sin(v), cos(v), tan(v),*** *der v oppgis i radianer.*



Figur X Endelig resultat Pyramide 1

10. Pyramide2

En pyramide har et volum på 100 kubikk cm og en grunnflate som er en likesidet trekant med sider 8 cm.Tegn pyramiden utbrettet slik at dersom vi klipper og bretter figuren får vi en pyramide med riktige dimensjoner.

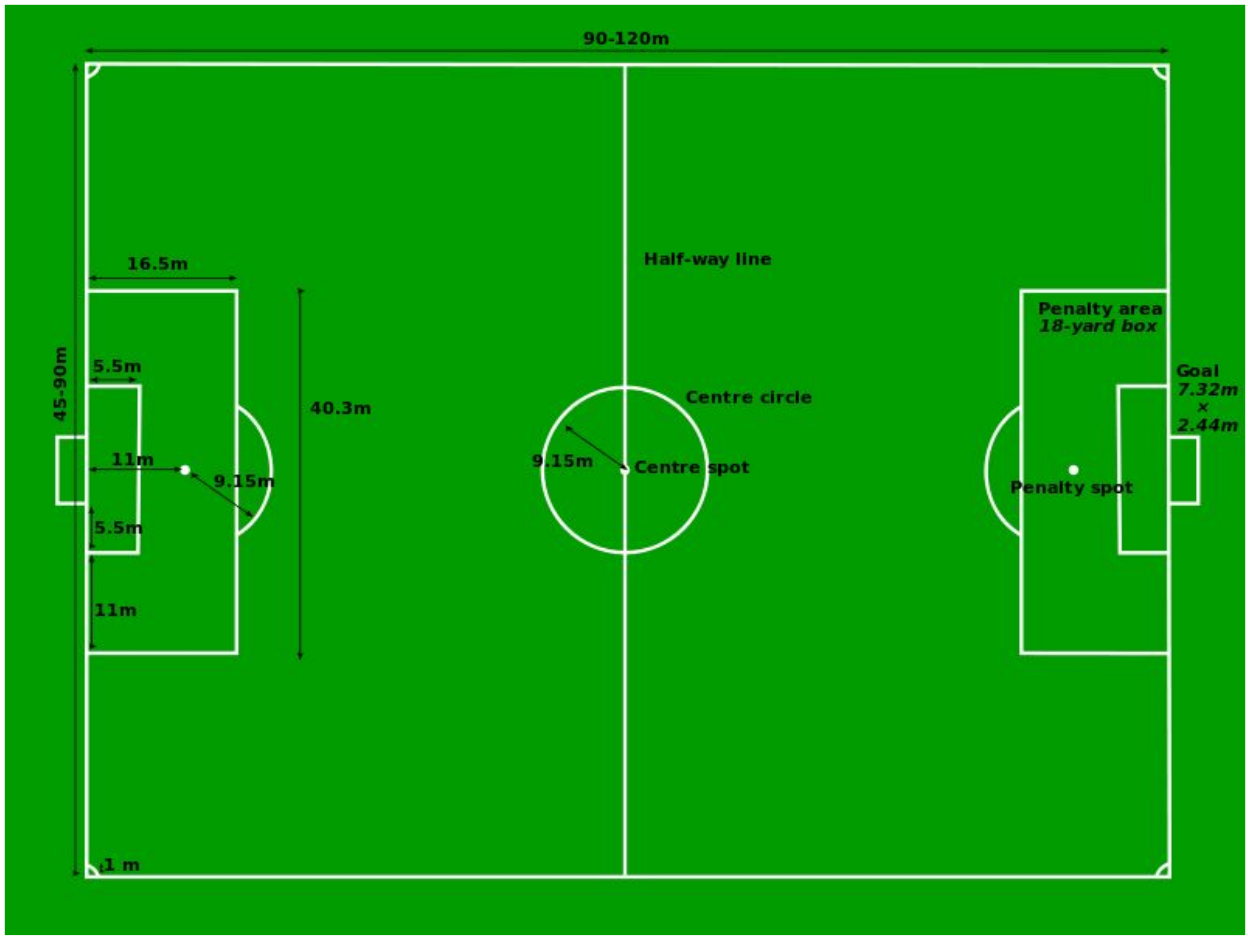


Figur X Endelig resultat Pyramide 2

For de som har alt for lite å gjøre:

11.Fotballbane

Her ser du tegning av en fotballbane (figur X på neste side).
Hentet fra https://no.wikipedia.org/wiki/Fotballbane



Figur X: Tegning av fotballbanen.

Tegn ut en fotballbane (64x100 m) som fyller mesteparten av et A4-ark.
Du trenger ikke sette på noe tekst, bare tegn alle linjene. Skriv ut i console hvor mange meter med linjer som skal til for å merke hele banen. Eller kanskje du vil lage en håndballbane, eller et kart over klasserommet.

12. Vedlegg: Litt hjelp i starten

Det finnes en masse muligheter i Processing. Det finnes en komplett oversikt under menyen "Help" i Processing-vinduet, velg "References". Det er såpass mye at det er litt forvirrende i starten. Nedenfor finner du det du trenger for å komme i gang med oppgavene.

Variabler (tabell X)

int antall=3	definerer et heltall
float vekt=2.6	definerer et desimaltall

Tabell X: Eksempler med bruk av variabeler

Geometriske funksjoner (tabell X)

rect(x1,y1,b,h)	tegner en firkant, et rektangel x1,y1 er normalt hjørnet oppe til venstre b er bredden og h er høyden
ellipse(x,y,b,h)	tegner en ellipse, en oval. x,y er sentrum, b er bredden og h er høyden. Hvis b=h blir det en sirkel med sentrum i (x,y)
triangle(x1, y1, x2, y2, x3, y3)	tegner en trekant med hjørner (x1,y1),(x2,y2) og (x3,y3)

Tabell X: Eksempler på bruk av metoder for geometriske funksjoner

Origo (tabell X)

translate(x,y)	flytter origo x til høyre og y nedover . Husk at positiv y-akse er nedover.

Tabell X: Eksempel på bruk av funksjon som har med origo å gjøre.

Farger (tabell X)

fill(r,g,b)	fyller figurene du tegner etterpå (firkant, ellipse, trekant) med en farge. du kan blande r: rød, g:grønn, b:blå. Alle kan være minst 0 og maks 255. fill(255,0,0) blir helt rødt , fill(0,0,255) blir helt blått, fill(255,255,0) blir gult. Prøv deg fram.
noFill()	ingen farge fylles i

stroke(r,g,b)	streken rundt figurene du tegner etterpå får denne fargeblandingen av r: rød, g:grønn, b:blå.
noStroke()	ingen omgivende strek

Tabell X: Fargeoperasjoner i Processing.

Hjelpeutskrift (tabell X)

println(a,b,c)	skriver det vi måtte ønske til console som er det lille vinduet under koden. Veldig greit når du skal teste noe

Tabell X: Utskrift til konsoll i Processing.

Alternativt oppgavesett i Processing for Strupe

Oppgave 1 – Enkel introduksjon

Lag et program som bare skriver "Hei verden".

Oppgave 2 – Rektangler

A. Fullfør funksjonen oppgave2A som lager et rektangel der 2 av sidene er 5 cm og de to andre sidene er 4 cm.
B. Gjør endringer i dette programmet slik at høyden på rektangelet er 8 cm og bredden 6,5 cm. Resultatet legger du i funksjonen oppgave2B.

Oppgave 3 – Kvadrater og kube/terning

A. Fullfør programmet slik at det tegner opp et kvadrat som har sidekanter på 4 cm. Dette legger du i funksjonen oppgave 3A.
B. Utvid funksjonen oppgave3B for å få programmet til å tegne en utbrettet terning, slik at man kunne brettet det sammen til en terning dersom det hadde blitt tatt en utskrift av det.

Oppgave 4 – Sirkel og sylinder

A. Fullfør et program som lager en sirkel der radiusen er 2 cm.
B. Utvid programmet slik at det med hjelp av sirkelen du kan lage en sylinder med volum på 100 og en radius lik den i oppgave 4A. Hva blir diameteren som skal brukes i sylinderen?

Oppgave 5 – Likebeinte trekanter og pyramide

A. Fullfør funksjonen oppgave5A slik at programmet lager en likebent trekant der to av sidene er 5 cm.
B. Gjør endringer i programmet slik at det lager en utbrettet pyramide. Bruk gjerne den samme trekanten som du lage i oppgave 5a) i flere eksemplarer. Pyramiden skal lages slik at det er et kvadrat i midten av den utbrettede pyramiden.

Oppgave 6: Rettvinklede trekanter og romber

Oppgaven kan ses på som en fortsettelse av oppgave 5 hvor man skulle lage trekanter som skulle brukes til å lage en pyramide.

A. Fullfør funksjonen oppgave7A. Funksjonen skal lage en trekant som har en rett vinkel. Kantene som går ut ifra den rette vinkelen skal være 3 og 4 cm. Hvor lang blir da den siste siden?
B. Bruk trekanten du lage i oppgave 7a) til å lage en rombe. Husk at i en rombe er alle sidene like lange.

Oppgave 7 – Flagg

Figur X under viser hvordan Monacos flagg ser ut (rødt øverst, hvitt nederst):

Figur X: Monacos flagg

Proporsjonen i flagget er 2:3
Fullfør Processing-programmet som tegner dette flagget, slik at det er 5 cm høyt.
Hint: Flagget består av 2 rektangler
Farg rektanglene slik:
fill(255,0,0); // rødt
fill(255,255,255); // hvitt. Kan også skrives fill(255);

Oppgave 8 – Likesidede femkanter og prismer med 5 kanter

A. Fullfør funksjonen oppgave8A som lager en femkant der alle sidene er like lange og hver side er 3 cm. *Hint: For å lage en slik femkant må man lage en liebeint trekant, tilsvarende den du lagde i oppgave 5A, med hjelp av funksjonen triangle. For å tegne resten av figuren, bruker du funksjonen quad.*
B. Fullfør funksjonen oppgave8B som lager et utbrettet prisme med 5 kanter. For å gjøre dette trenger du to eksemplarer femkanten du laget i oppgave 8A (topp og bunn). Som et hint for hvordan man lager kantene rundt figuren, kan man bruke vanlige kvadrater med funksjonen rect(). Alle sidekantene skal ha en lengde på 3 cm.

Oppgave 9 – Areal av en trekant

En trekant har et areal på 12 cm^2. Fullfør funksjonen oppgave9 som skal tegne en rettvinklet trekant med selvvalgte sidelengder. Til slutt skal du fullføre funksjonen som regner arealet av trekanten du har lagd. Klarer du å lage en rettvinklet trekant med et

areal på 12 cm^2? Husk at arealet for en trekant er (g*h)/2, der g er grunnlinja og h er høyden på trekanten.

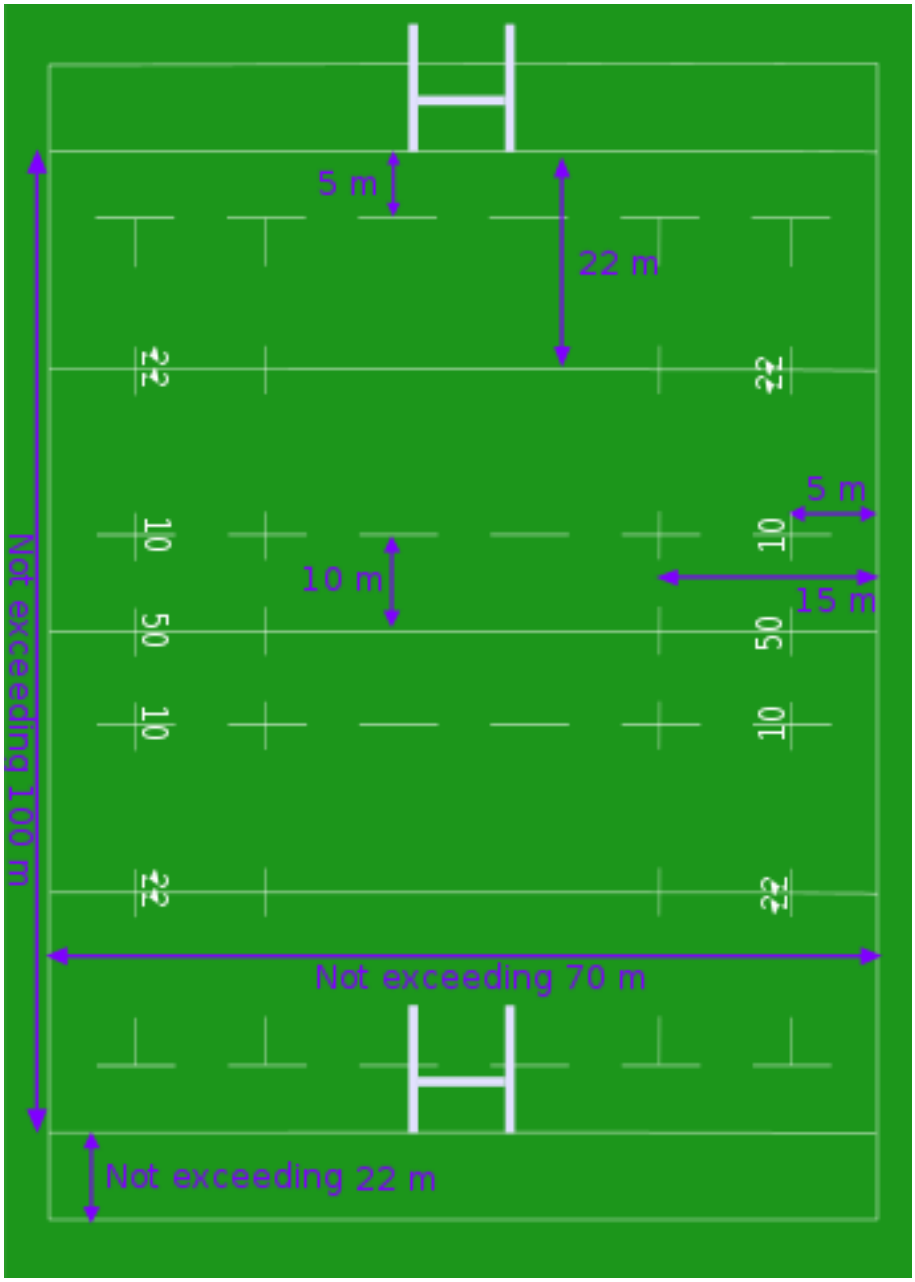
Oppgave 10 – Volum av et trekantet prisme

Et trekantet prisme har et volum på 60 cm^3. Bestem høyden på prismet og bestem deretter sidene på trekanten. I denne oppgaven, for enkelhets skyld skal trekanten være rettvinklet. For å kunne regne volumet til prismet du lager, må du først regne ut arealet til trekanten, slik du gjorde i oppgave 10. Deretter kan du regne ut volumet slik: V = G * h, der G er grunnflata/arealet til trekanten og h er høyden til prismet.
For å tegne trekanten, må du fullføre funksjonen *oppgave10*, samt kalle på funksjonene regnAreal og regnVolum.
Klarer du å lage et trekantet prisme som har et volum på 60 cm^3?
Hint: Resultatene av regnAreal og regnVolum må lagres i variabler. For å gjøre dette, skriver du: "brukerAreal = regnAreal(grunnlinje, trekanthøyde); og brukerVolum = regnVolum(brukerAreal, prismehøyde);"

Ekstraoppgave for de drevne som vil ha en utfordring

På neste side ser du et bilde av en rugbybane (figur X).

Fullfør programmet som tegner banen. Du trenger bare å tegne de hvite linjene, se bort ifra målene (de H-formede figurene) og tallene.
Har du lyst til å prøve deg på en fotballbane, håndballbane, osv. eller kanskje klasserommet ditt, må du gjerne gjøre det.

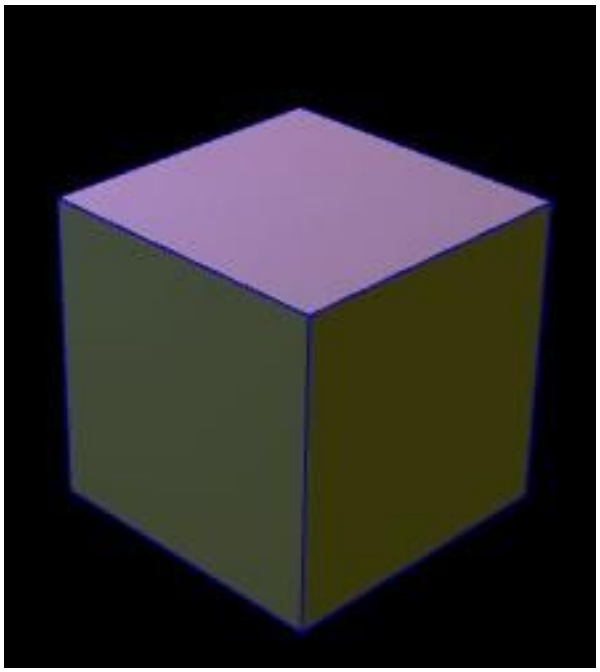


Figur X: Illustrasjon av en rugbybane med avstander målt i meter. Bildet er hentet fra Wikipedia:
https://en.wikipedia.org/wiki/Rugby_union#/media/File:RugbyPitchMetricDetailed.svg

Frivillig ekstraoppgave – kube i tre dimensjoner

Oppgaven følger samme konsept som de andre, du skal fullføre funksjonen som skal tegne opp alle sidene i en tredimensjonal kube. Husk at en kube i 3D har seks kanter. Denne funksjonen heter *tegnKube*
Alle kantene (kalt "side" i dette programmet) har både lengde og høyde som må fylles inn.

På forhånd er det opprettet 6 sideobjekter: Foran, bak, venstre, hoyre, topp og bunn. Du må initialisere disse (*hint: bruk kodeordet **new***) slik: "foran = new foran(X, Y);" der X er lengden og Y er høyden. Til slutt må du gjøre et kall på funksjonen *tegn*, men du må selv finne ut av hvilket klasseobjekt den tilhører (hint: Et kall på en funksjon som finnes i en annen klasse, skrives slik: "objekt.funksjon();", f.eks. foran.hentBredde();). Figur X under viser hvordan kubenkan se ut.



Figur X: Et eksempel på hvordan kuben kan se ut.

4 Gjennomføringsbeskrivelse, Os

Dette kapittelet tar for seg beskrivelser av selve utføringen av undervisningen. Detaljnivået er relativt høyt, for å sikre mest mulig nyttig informasjon for ProgPed, med tanke på eventuelle oppfølgingsprosjekter og lignende fremtidige undervisningsopplegg. Disse beskrivelsene er øktvise, og nokså spesifikke, i motsetning til det overordnede metodekapittelet.

Økt én:

Dato: 09.02.2016.
Tidspunkt: 11:45-13:15.

Undervisningsopplegget var lagt opp slik at i begynnelsen satt både A-klassen og B-klassen i samme rom, for å få en felles introduksjon til Scratch og programmering. Etter introduksjonen ble det gjennomgått en rekke demo-programmer (se vedlegg 1, *Oppgaveindeks*) live, med innspill fra klassen. Etter introduksjon og demo ble klassene splittet til hvert sitt klasserom. Aasmund Lunde tok ansvar for B-klassen, med hjelp fra Tobias Alvik Hagen. Odd Tore Kaufmann tok ansvar for A-klassen, med hjelp fra Anders Bolt-Evensen. Anders Heggestøyl gikk frem og tilbake mellom klasserommene, for å få et helhetlig perspektiv på de forskjellige klassenes undervisningsopplegg og fremgang.

Etter klassene ble splittet til hvert sitt rom, fikk elevene fire øvinger som de skulle løse på egenhånd – hver øving gikk ut på å sette sammen programmer som var ekvivalente med demo-programmene (se vedlegg 1, *Oppgaveindeks*) som ble demonstrert i forkant. Utførelsen av disse øvingene konkluderte økt én.

Økt to:

Dato: 10.02.2016.
Tidspunkt: 08:30-09:15.

I begynnelsen av økt to begynte begge klassene med en kort oppsummering av forrige økt, med litt repetisjon av demoprogrammene. Etter denne oppsummeringen hendte det vesentlige forskjeller i undervisningsopplegget mellom klassene. A-klassen begynne rett

på et oppgavesett laget av bachelorgruppa (se vedlegg 2, *Oppgavesett, Os*), mens i B-klassen ville Aasmund Lunde bruke tid på å gå gjennom variabel-konseptet. Forskjellen i gjennomgang av variabel-konseptet vil danne et viktig grunnlag for sammenligningen av de forkskjellige klassenes resultater.

Økt tre:

Dato: 17.02.2016.
Tidspunkt: 08:30-09:15.

A-klassen: Økten begynte med en kort repetisjon av de oppgavene som de fleste elevene ble ferdig med i økt to. Løsningen ble gjennomgått på storskjerm, med innspill fra klassen om hvordan den skulle løses. Etter repetisjonen ble klassen satt til å jobbe videre med oppgavesettet. Mot slutten av økten ble løsningen på de oppgavene som de fleste elevene hadde fått til gjennomgått i plenum, igjen med innspill fra elevene om hvordan oppgavene skulle løses.

B-klassen: I begynnelsen av økten startet elevene på oppgave 1 fra oppgavesettet. De ble testet underveis ved at Aasmund spurte ut om oppgavene. De jobbet seg opp til og med oppgave 4, da Aasmund ikke så ut til å skjønne hvordan denne oppgaven var tiltenkt. Avslutningsvis viste en elev en egenprodusert sketch, hvor man kunne styre figuren ved hjelp av piltastene.

Økt fire:

Dato: 01.03.2016.
Tidspunkt: 11:45-13:15.

A-klassen: Mer tid enn vanlig gikk til repetisjon av oppgaver fra forrige økt, ettersom vinterferien hadde forlenget tidsrommet mellom økt tre og økt fire. Etter repetisjonen ble elevene satt til å jobbe med oppgavesettet. Løsningen på oppgavene ble gjennomgått i plenum 30 minutter før økta var over. Den siste halvtimen fikk elevene i oppgave å utforske https://scratch.mit.edu/explore/, og forsøke å gjøre endringer i eksisterende programmer, eller å forklare hvordan deler av et eksisterende program fungerer. Alternativt fikk de prøve å lage sitt eget program fra bunn av. I forkant av oppgaven ble de fortalt at tre elever eller grupper på to skulle presentere hva de hadde laget og/eller lært i løpet av oppgaven. Økten ble avsluttet med presentasjon fra elever.

B-klassen: Det ble ingen repetisjon av oppgaver fra øktene før vinterferien. Elevene ble i starten av timen introdusert for oppgave 5. Da elevene svarte noe vagt på spørsmålene om omgjøring fra millimeter til meter, gikk læreren over til vanlig tavleundervinsning for å friske opp i omgjøringen mellom volumenheter. Klassen beveget seg etterhvert over til oppgave 6. Videre lot læreren elevene jobbe med egne oppgaver. Enten oppgaver de komponerte på egenhånd, eller i lag med andre, eller oppgaver de fant på https://scratch.mit.edu/explore/. Økten ble avsluttet ved å gå gjennom lærerens valgfrie hjemmelekse, gitt før vinterferien, et program som spurte om tilfeldige gangestykker fra en gangetabell. Tilslutt spurte læreren om tilbakemelding på selve programmeringsopplegget.

5 Gjennomføringsbeskrivelse, Strupe

Dette kapittelet tar for seg beskrivelser av selve utføringen av undervisningen. Detaljnivået er relativt høyt, for å sikre mest mulig nyttig informasjon for ProgPed, med tanke på eventuelle oppfølgingsprosjekter og lignende fremtidige undervisningsopplegg. Disse beskrivelsene er øktvise, og nokså spesifikke, i motsetning til det overordnede metodekapittelet.

Økt én:

Dato: 09.03.2016.
Tidspunkt: 12:15-14:00.

Økten begynte med en introduksjon av bachelorgruppa, og en forklaring på hvordan undervisningsopplegget var tenkt å utføres. Deretter en kort, muntlig introduksjon om hva programmering er, hva et dataprogram er, hvordan de lages, og hvordan maskiner utfører dem.

Etter introduksjonen ble Processing åpnet på storskjermen, og elevene ble bedt om å åpne Processing på sine egne maskiner. Det ble gitt en kort introduksjon av grensesnittet til Processing: Hvor man skriver, hvilken knapp man trykker for å kjøre et program, hvordan man endrer tekststørrelsen, dersom man har vanskeligheter med å lese teksten.

Det første som ble gjort i Processing var å kjøre et tomt program, for å illustrere at ingen ting skjedde, annet at et lite vindu dukket opp. Den første kommandoen ble derfor introdusert for elevene:

```
size(600,600);
```

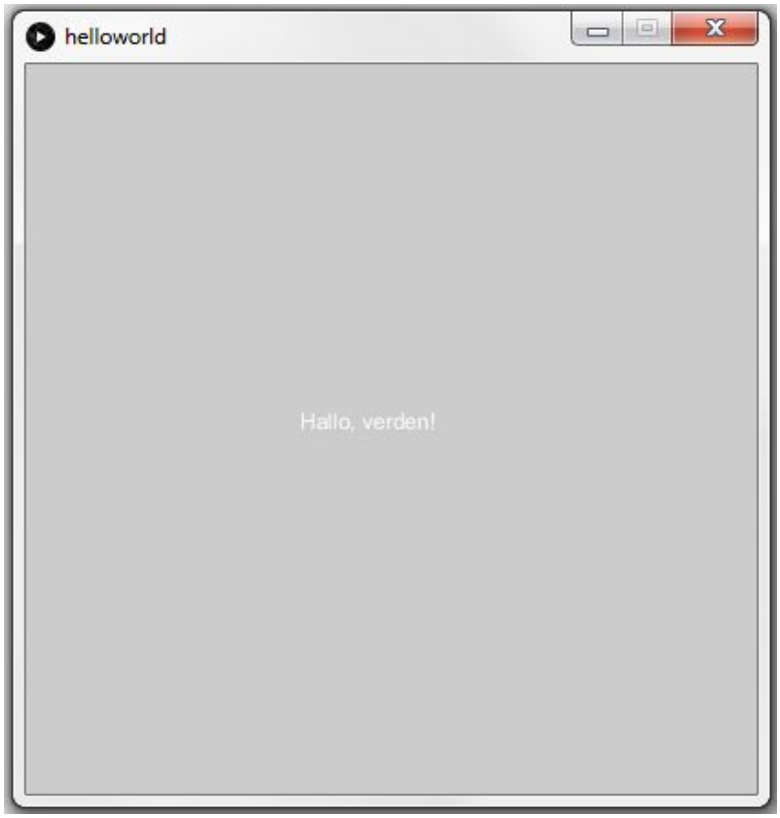
Her ble det forklart at kommandoen hadde et navn, hva parametre er, at parametre skilles med komma, og at semikolon markerer enden på en kommando – forklart som at det fungerer “som et punktum” i programmering. Programmet ble kjørt, både på storskjerm, og på hver av elevenes maskiner.

Det ble satt som mål at klassen i fellesskap skulle lage sitt første program. Det skulle skrive “Hallo, verden!”. Programmet ble utvidet med linjen:

```
text("Hallo, verden!",200,300);
```

Denne linjen krevde forklaringen på at forskjellige kommandoer kunne ha forskjellige antall og typer parametre. Det var også her koordinatsystemet måtte forklares, ettersom kommandoen inneholdt koordinatene til teksten. Fordi Processing bruker et koordinatsystem der origo er definert øvert til venstre, i motsetning til det ordinære koordinatsystemet, hvor origo er definert nederst til venstre, måtte forskjellen forklares.

Microsoft Paint ble brukt på storskjermen for å tegne opp et ordinært koordinatsystem, med eksempelkoordinaten (2,3). I tillegg ble det tegnet opp det omvendte koordinatsystemet som Processing bruker, med den samme eksempelkoordinaten (2,3). Denne tegningen ble brukt for å illustrere hvordan x-verdien oppfører seg likt på tvers av systemene, men at en høyere y-verdi i Processing betyr lenger nede på y-aksen. Programmet gav resultatet illustrert i figur 1 nedenfor:



Figur 1: Første utgave av "Hello World!"-program

For å gjøre teksten mer leselig, ble kommandoene `textSize(30);` og `fill(200,0,200);` introdusert.

Her ble forskjellen mellom de to fargetypene *fill* og *stroke* forklart, i tillegg til hvordan RGB-systemet ble brukt for å definere farger. Det ble forklart som at man kan lage farger med en gitt mengde rødt, en mengde grønt, og en mengde blått, og at hvert parameter reflekterte mengden til de ulike fargene. Det ble også forklart at minimum mengde av hver farge var 0, og at maksimum var 255. Hvorfor disse verdiene ble brukt ble bevisst hoppet over, for å ikke skape unødvendig forvirring.

Introduksjonen av disse kommandoene ble også brukt for å forklare hvordan rekkefølgen på kommandoene har betydning for programmets utførelse. Dette ble demonstrert ved først å legge til de nye linjene i bunn, og kjøre programmet. Da ingen ting skjedde, ble textSize og fill-linjene flyttet i forkant av text-kommandoen, og resultatet demonstrert.

Da det grunnleggende "Hallo, verden!"-programmet var ferdig, fikk klassen i oppgave å utvide programmet til å skrive en ny linje under "Hallo, verden!", med teksten "Hei igjen".

Flere elever fra klassen kom med gode innspill, om at man trengte en ny text-kommando, med høyere y-verdi i koordinatene. Da elevene ble utfordret til å skifte farge på den andre linja, men ikke den første, klarte de å identifisere at en ny fill-kommando måtte stå over den nye text-kommandoen. Resultatet av øvelsen vises i figur x.

Figur x: Resultat av "Hello World!"-øvelse

Møtereferater

Møte 1

...
...

Kick-off-møte, Bacheloroppgave

Dato: 14.01.2016
Tid: 13:00-15:45

Oppmøte: *Anders Heggestøyl, Tobias Alvik Hagen, Anders Bolt-Evensen, Børre Stenseth, Odd Tore Kaufmann*
Frafall: *Ingen.*
Leder: *Anders Heggestøyl*
Referent: *Tobias Alvik Hagen*

Agenda:

1. Planlegging
2. Forprosjektet
3. Annet

Punkt 1 Planlegging

Vi avtaler fremtidige møter til å være tirsdag kl 13:15
Børre ønsker at vi minimerer avhengigheten av verktøy for lesing av dokumenter, presentasjon av kildekode, etc. Vi bør priotere neste uke og komme igang. Gruppen blir enige om å bruke Trac som prosjektstyringsverktøy hvor Børre legge ut hva som gjelder av arbeidsoppgaver.

Punkt 2 Forprosjekt

I forprosjektet bør vi ha med komponenter som deltakelse i undervisning, observasjon, ulike utviklingsverktøy og hva som gjøres i andre land. Det bør være med argumenter på hvordan programmering knyttes opp mot selve undervisning, som matematikkfag eller opp mot andre fag. I sluttrapporten kan vi ta med hva som endret seg fra forprosjektet frem til slutten av prosjektet.

Punkt 3 Annet

En tricky bit med oppgavene til Processing er å skrive ut til nøyaktig A4-format. Det skal være mulig å måle med linjal på utskrift at det er riktig, men det kan oppstå noen skjevheter på milimeternivå. Vi sjekker opp i mulige løsninger på dette.

Neste møte: Torsdag 21.01.2016 - kl 14:00

Agenda:
- Sikring av kodeeksempler
- Forprosjektsrapport

Referent,

Tobias Alvik Hagen

6 Fremgangsrapport #1 26.01.2016 – 02.02.2016

Fremgangsbeskrivelse

For å få en bedre oversikt over vår fremgang i prosjektet, både internt i gruppa, samt som en måte å oppdatere veileder på, kan det være nyttig å se tilbake på planene vi la i aktivitetsplanen for denne perioden. Ved å sammenligne oppgaver som var satt til å fullføres innen denne rapportens dato (02.02.2016) med hvilke oppgaver som ble utført kan vi få en god oversikt over hvordan planer og realitet avviker eller samsvarer.

Her er vår aktivitetsplan for aktiviterer i perioden 26.01.2016 og 02.02.2016:

Aktivitet	Start est.	Slutt est.	Start reell	Slutt reell	Innsats estimat	Innsats reell	Ansvar
Avtale møte med Odd Tore	26.01	26.01	26.01	26.01	0.5	0.5	A.H.
Renskrive referater	26.01	26.01	26.01	N/A	1	N/A	T.H.
Fullføre Gantt og risikoanalyse	26.01	29.01	26.01	N/A	2	N/A	T.H.
Utvikle oppgaver og eksempler, Os	26.01	29.01	26.01	30.01	12	8	A.H.
Utvikle oppgaver og eksempler, Os	26.01	29.01	26.01	30.01	12	8	T.H.
Utvikle oppgaver og eksempler, Os	26.01	29.01	26.01	30.01	12	11	A.E.
Workshop Os	28.01	28.01	28.01	28.01	2	2	Alle
Workshop Strupe+VGS	29.01	29.01	29.01	29.01	2	2	Alle
Konkretisere omfang	30.01	02.02	02.02	02.02	1	1	A.H.
Videreutvikle og forbedre, Os	30.01	02.02	30.01	02.02	8	7.5	A.H.
Videreutvikle og forbedre, Os	30.01	02.02	30.01	02.02	8	2	T.H.
Videreutvikle og forbedre, Os	30.01	02.02	30.01	02.02	8	9	A.E.
Fremgangsrapport #1	02.02	02.02	02.02	02.02	2	2	A.H.

Grønn: Fullstendig.
Gul: Pågående arbeid.
Rød: Langt i fra fullstendig.

I denne perioden har vi hovedsaklig fokusert på Scratch, ettersom uttestingen hos Os er like rundt hjørnet. Planen var å utvikle en rekke førsteutkast til oppgaver, få feedback på dem fra Odd Tore og gjennom workshoppen, og så videreutvikle de oppgavene som viste potensial.

I den første bolken med utkast til oppgaver viste det seg at de fleste oppgavene kunne kategoriseres som to typer: 1) Kalkulator-oppgaver: Et program som mottar en input, og spytter den ut som en annen måleenhet. Og 2) “Tekstbok”-oppgaver: Oppgaver som like så godt kunne stått i en tekstbok, og som ikke benytter noen av fordelene ved programmering. Eksempel på dette er programmer som ber eleven om å regne ut en omgjøring, og oppgir hvorvidt svaret er riktig eller ikke. Ingen av disse typene er hensiktsmessige, og gir ikke grunn til å bruke programmering som verktøy.

Dersom elevene aldri blir oppfordret til å se i koden, gjør vi noe feil. Et viktig verktøy for å gjøre disse “kalkulator-oppgavene” nyttige er god bruk av oppgavetekst. Oppgave 1 i det vedlagte oppgavesettet illustrerer hvordan oppgavetekst kan gi nytte til en slik oppgave. Dette var Odd Tore enig i. Anders Heggestøyl kommer i neste ukesperiode å skrive en del dokumentasjon om filosofi/retningslinjer for Scratch-oppgaver for bruk i barneskolen.

Apropos oppgavetekst. Vi har foreslått at måten for utdeling av oppgaver bør foregå gjennom at hver elev får ett dokument som inneholder oppgavene, og at oppgaveteksten linker til Scratch-programmene online. I følge Harald er det bestemt at det er online-versjonen som vil bli tatt i bruk, så dette vil være en god løsning for å gjøre logistikken litt enklere. Det blir mye enklere for elevene å gå direkte til en link i oppgaveteksten, enn finne fram til forskjellige lokalfiler som de enten har måttet laste ned, eller som ligger bortgjemt i en mappe. Det betyr også at det eneste vi trenger å distribuere til elevene er ett dokument som inneholder oppgavetekstene (eventuelt til og med bare en link til oppgavesettet).

Oppgavesettet som er vedlagt i samme mailen som denne rapporten inneholder en del nye konsepter for hvordan programmeringen kan være mer til nytte for matematikkforståelsen. For eksempel hvordan vi kan la løse kode-klosser være liggende i programmet, som elevene må sette på plass for å fullføre programmet. Odd Tore synes dette var retningen å gå, og har gitt masse nyttig feedback på oppgavene mtp oppbygging og formuleringer, noe vi skal jobbe videre med i neste ukesperiode. Odd Tore gav også en rekke tesktbok-oppgaver som vi kunne bruke som inspirasjon til å komme opp med flere av disse “out of the box thinking”-konseptene.

Planer for neste ukesperiode: Anders Heggestøyl videreutvikler oppgavesett basert på feedback fra Odd Tore, oppfyller noen oppgavebestillinger, og skriver dokumentasjon for filosofi/retningslinjer for Scratch-oppgaver. Tobias Alvik Hagen jobber med en introduserende demo, som er ment å brukes i forkant av faktiske oppgaver, for å demonstrere enkle konsepter i Scratch steg for steg (basert på feedback fra workshop med Os). Fordi VGS-lærerne uttrykket bekymring over at tegn-og-print-oppgavene i Processing tok for lang tid, skal Anders Bolt-Evensen utforske andre løsninger for visualisering av tredimensjonale objekter i Processing basert på flater som elevene skal regne seg fram til. Fordi Tobias ikke fullførte alle sine aktiviteter for denne ukesperioden vil de uferdige aktivitetene også videreføres til den neste ukesperioden.

Begge workshoppene gav verdifull informasjon om hva som fungerer, og hva som bør forbedres. Vi har en rekke punkter basert på hva som ble sagt under workshoppene som vi kommer til å jobbe videre med.

Utfordringer

Det er åpenbart at variabel-konseptet vil være problematisk å introdusere i barneskolen. En mulig løsning på dette vil være å unngå oppgaver som krever at elever oppretter sine egne variabler, og å heller forsyne dem med variablene de trenger på forhånd. Dette vil være et argument for oppgaver der programmet har løse byggeklosser, som elevene må bruke for å fullføre/utvide programmet.

Aasmund Lunde påpekte en del aspekter ved programmene som kan være unødvendig forvirrende for elevene: For eksempel “hide variabel”-klosser, og andre kontroll-kommandoer som ikke er spesielt intuitive. Her vil en løsning være å “gjemme bort” delene av koden som barna ikke trenger å se. Dette kan gjøres ved å f.eks. plassere kontroll-kode til å være tilhørende bakgrunnsbildet, og den relevante koden tilhørende sprite 1. Det er skript tilhørende sprite 1 som vises som default, og man vil ikke finne den resterende koden med mindre man leter etter den.

Den største kritikken fra workshoppene var fra VGS, om at tegn-og-klipp-oppgavene tok for lang tid, og at de ikke nødvendigvis hadde nok fokus på matematikken. Anders Bolt-Evensen kommer til å se på mulige alternativer for å forbedre det logistiske aspektet, i tillegg til mer fokus på matematikkforståelse.

Konklusjon

Alt i alt ligger vi omtrent der vi hadde forventet. Både workshops og Odd Tore har gitt verdifull feedback, som vi bruker i neste ukesperiode for å videreutvikle. Forhåpentligvis har vi i løpet av neste tirsdag et solid forslag til oppgavesett, samt et førsteutkast til en demo/introduksjonsbit.

Produktivitetsnivået har økt fra forprosjektet, men kan økes videre. De fleste aktiviteter for neste ukesperiode er allerede definert, med ansvar fordelt, men det dukker helt sikkert opp et par uventede aktiviteter underveis.

Snart blir det tid for å skifte fokus fra utvikling av undervisningsmateriale til mer dokumentasjon og teori.

Neste fremgangsrapport sendes 09.02.2016.

7 Fremgangsrapport #2 03.02.2016 – 17.02.2016

Fremgangsbeskrivelse

For å få en bedre oversikt over vår fremgang i prosjektet, både internt i gruppa, samt som en måte å oppdatere veileder på, kan det være nyttig å se tilbake på planene vi la i aktivitetsplanen for denne perioden. Ved å sammenligne oppgaver som var satt til å fullføres innen denne rapportens dato (17.02.2016) med hvilke oppgaver som ble utført kan vi få en god oversikt over hvordan planer og realitet avviker eller samsvarer.

Her er vår aktivitetsplan for aktiviterer i perioden 03.02.2016 og 17.02.2016:

Aktivitet	Start est.	Slutt est.	Start reell	Slutt reell	Innsats estimat	Innstats reell	Ansvar
Forbedre oppgavesett basert på feedback fra Odd Tore	03.02	07.02	03.02	08.02	3	3.5	A.H.
Oppgavebestilling Odd Tore	03.02	07.02	03.02	08.02	5	2	A.H.
Oppgaver basert på inspirasjon-dok.	03.02	07.02	03.02	07.02	5	5	A.E.
Demo/tutorial-materiale Scratch	03.02	07.02	03.02	07.02	8	6.5	T.H.
3D-animasjon i processing, flater	05.02	09.02	02.02	02.02	5	5.5	A.E.
Dokumentasjon filosofi/retningslinjer	07.02	09.02	09.02	11.02	4	4	A.H.
Planlegging m. Odd Tore og Aasmund Lunde	N/A	N/A	08.02	08.02	N/A	0.5	A.H.
Undervisning Os #1	09.02	09.02	09.02	09.02	2	2	Alle
Undervisning Os #2	10.02	10.02	10.02	10.02	2	2	Alle
Dokumentere resultater, Os	10.02	12.02	10.02	15.02	1.5	1.5	Alle
Veiledning Børre	09.02	09.02	10.02	10.02	1	1	Alle
Skype-møte	N/A	N/A	15.02	15.02	1	1	Alle
Undervisning Os #3	16.02	16.02	N/A	N/A	2	N/A	Alle

Undervisning Os #4	17.02	17.02	17.02	17.02	2	2	Alle
Dokumentere metode, Os	17.02	17.02	17.02	17.02	2	2	A.H.
Dokumentere resultater, Os	17.02	17.02	17.02	17.02	1.5	1.5	Alle
Tilpasse oppgavesett	N/A	10.02	10.02	10.02	N/A	1	A.H.
Fremgangsrapport #2	16.02	16.02	17.02	17.02	2	2	A.H.

Grønn: Fullstendig.
Gul: Pågående arbeid.
Rød: Langt i fra fullstendig.

Denne perioden har stort sett dreiet seg om undervisningen på Os. Tre av fire økter er nå i boks, og resultatene har vært positivt overraskende. Vi endte opp med å kjøre en felles introduksjon for begge klassene i begynnelsen av den første økten, og så splittet vi opp i to klasserom. Undervisningsopplegget ble noe annerledes for A-klassen enn det det var for B-klassen. I B-klassen brukte Aasmund resten av økten på å bruke egne eksempler for å gå gjennom variabel-konseptet i detalj. I midlertid hoppet A-klassen rett på øvinger etter introduksjonsbiten – de fikk kun en 5-minutters forklaing på variabler fra Bolt.

B-klassen begynnte derfor ikke på det reelle oppgavesettet før i dag (tredje økt), mens A-klassen gjorde oppg 1 og 2 i andre økt, og 3 og 4 i tredje økt. Etter fjerde økt vil det bli interessant å sammenligne hvor godt hver av klassene løste oppgavesettet. Fjerde økt, hvor begge klassene forhåpentligvis vil være innom oppg 4, 5 og kanskje 6 vil være det viktigste grunnlaget for sammenligning, ettersom disse har mye variabelbruk.

Oppgavesettet kan sees her:
https://docs.google.com/document/d/1oT-Fnk5fZIgmoVKQ2sWnmcCj-TQhkg-8REml-7PNIk/edit

Etter veiledningen forrige onsdag har vi et ganske klart bilde over hvordan sluttdokumentet vil se ut. Vi har begynt å dokumentere metode og resultater for de undervisningsøktene vi allerede er gjennom. Tobias dokumenterte utelukkende for B-klassen, Bolt utelukkende for A-klassen, og Anders H gikk frem og tilbake mellom rommene, og holdt et overordnet perspektiv for å knytte det sammen.

Planer for neste ukesperiode: Fokus på dokumentasjon. Spesielt med tanke på de tre øktene som er ferdige på Os, men også begynnte å titte på tooling, forbedre disposisjon til å matche nye planer, og forberede Strupe-uttestingen.

Utfordringer

I det som skulle være økt 3 (tirsdag 16.02) møtte vi på et teknisk problem: MIT sin Scratch-server var nede. Dette varte fra da timen begynte, og frem til rett før timen var ferdig. Det ble derfor ikke noe Scratch denne dagen, og økt 3 ble flyttet til onsdagen, som skulle være økt 4. Heldigvis får vi forstatt mulighet til å kjøre økt 4, som kan være den viktigste økten, men dette blir ikke før tirsdag etter vinterferien.

Vår backup-plan var å ha installeren til den lokale versjonen klar på minnepinne. Én av ulempene ved offline-versjonen er at den krever Adobe Air, noe Scratch-installeren også fikser. Det vi ikke var forberedt på var at denne installeren også trengte kommunikasjon med MIT, som var nede… I dette tilfellet måtte både Scratch og Adobe Air vært installert på hver eneste maskin på forhånd, i tilfelle nedetid fra MIT sin side.

Konklusjon

Selv om det var synd at én økt ble forskjøvet av de tekniske vanskelighetene, var de tre øktene vi har hatt så langt veldig lovende. Elevene tok stoffet til seg overraskende fort, og de har vært veldig entusiastiske. Mange hender i været når ting taes i plenum.

Før vinterferien blir det fokus på dokumentasjon. Vi er også snart i overgangen fra Os og Scratch til Strupe og Processing. Vi har en del å tenke på når det kommer til hvordan undervisningen på Strupe skal gjøres.

Neste fremgangsrapport sendes 01.03.2016.

8 Fremgangsrapport #3 18.02.2016 – 09.03.2016

Fremgangsbeskrivelse

For å få en bedre oversikt over vår fremgang i prosjektet, både internt i gruppa, samt som en måte å oppdatere veileder på, kan det være nyttig å se tilbake på planene vi la i aktivitetsplanen for denne perioden. Ved å sammenligne oppgaver som var satt til å fullføres innen denne rapportens dato (09.03.2016) med hvilke oppgaver som ble utført kan vi få en god oversikt over hvordan planer og realitet avviker eller samsvarer.

Her er vår aktivitetsplan for aktiviterer i perioden 18.02.2016 og 09.03.2016:

Aktivitet	Start est.	Slutt est.	Start reell	Slutt reell	Innsats estimat	Innsats reell	Ansvar
Introduksjon, Os	25.02	28.02	N/A	N/A	2	N/A	T.H.
Metode, Os	25.02	28.02	27.02	01.03	2	2.5	A.H.
Resultater, Os	25.02	03.03	25.02	N/A	8	4.5	A.H.
Diskusjon, Os	03.03	08.03	02.03	N/A	12	4	Alle.
Konsepter til processing	25.02	29.02	27.02	04.03	8	10	A.E.
Fullføre oppgavesett	29.02	03.03	29.02	04.03	4	4	A.E.
Undervisning Os #4	01.03	01.03	01.03	01.03	2	2	Alle.
Introduksjon hoveddokument	28.02	03.03	N/A	N/A	4	N/A	T.H.
Teori	01.03	08.03	01.03	N/A	10	3	T.H.
Erfaringer og løsninger MIT	28.02	03.03	26.02	26.02	2	2	A.H.
Utveksle resultater med Kristina	29.02	29.02	29.02	29.02	0.5	0.5	T.H.
Oppdatere dokumenter + linker, frigg	03.03	03.03	09.03	09.03	1	1	A.H.
Evaluering av verktøy	25.02	08.03	25.02	0N/A	15	5	T.H.
Mulig ekstra økt Os	02.03	02.03	N/A	N/A	2	N/A	Alle
Undervisning Strupe #1	08.03	08.03	09.03	09.03	2	2	Alle
Fremgangsrapport #3	08.03	08.03	09.03	09.03	2	2	A.H.

Grønn: Fullstendig.
Gul: Pågående arbeid.
Rød: Langt i fra fullstendig.

Perioden begynnte med å fullføre den fjerde og siste økta med Os. Dette var økta som var ment å fullføres før vinterferien, men ble forskjøvet grunnet de tekniske problemene. Den meste tiden har gått til dokumentasjon for Os-casen, forberedelse for Strupe, og allerede en undervisningsøkt ved Strupe.

Vi er ikke helt ferdig med dokumentasjon for Os, men vi klarer å fullføre det før fredag, når vi skal levere førsteutkastet til hoveddokumentet. Strupe kom noe tidligere enn forventet – vi hadde regnet med første økt på fredag 11.03, men den kom allerede på mandag (ble utsatt til tirsdag i siste liten). Vi var heller ikke klar over at det var vi som skulle kjøre hele undervisningsopplegget, så en del tid gikk til å forberede undervisningen ved Strupe.

Vi har alt klart før neste økt på Strupe til fredag, så vi behøver ikke bruke mer tid på forberedelser. Frem til fredag er førsteprioritet å få Os-dokumentasjonen 100% ferdig, samt en introduksjon til sluttrapporten. Etter fredag fortsetter vi med undervisning og dokumentasjon ved Strupe, og Tobias ser på teori og evaluering av andre verktøy.

Utfordringer

Den første Strupe-økta kom litt brått på, noe som gikk ut over tiden som var dedikert til å fullføre Os-dokumentasjonen. Vi klarer allikevel å få dette på plass før innleveringen til fredag.

Det var også en del surr fra lærerne ved Strupe. De kansellerte én av øktene på grunn av bibliotekbesøk, noe vi ikke fikk vite før en time før økta var ment å starte. Dette førte til at Odd Tore ikke fikk blitt med på økt én, ettersom vi måtte flytte økta til dagen etter. Vi var heller ikke forberedt på at økt én måtte skje i kantina, i stedet for det ordinære klasserommet, noe som førte til at vi måtte laste ned processing på en skole-maskin som var kompatibel med docking-stasjonen de brukte for prosjektoren i kantina.

Konklusjon

Vi ligger litt bak der vi ønsket å være med Os-dokumentasjonen, men vi klarer å fullføre det før innleveringen på fredag. På den lyse siden er vi allerede ferdig med en

undervisningsøkt ved Strupe, noe som gikk veldig bra, og vi har det meste av forarbeid for de neste øktene klart. Vi endte også opp med noe dokumentasjon for Strupe tidligere enn vi hadde antatt. Nå har vi to dager på å fullføre førsteutkastet til sluttrapporten, og så blir fokus på dokumentasjon for Strupe, evaluering av verktøy, og drøfting for hvilke deler av mattepensumet som kan dra mest nytte av programmering.

Neste fremgangsrapport sendes 22.03.2016.